Normalization of Historical Texts with Neural Network Models

Inaugural-Dissertation

zur

Erlangung des Grades eines Doktors der Philosophie

in der

Fakultät für Philologie

der

RUHR-UNIVERSITÄT BOCHUM

vorgelegt

von

Marcel Bollmann

Gedruckt mit der Genehmigung der Fakultät für Philologie der Ruhr-Universität Bochum

Referent: . . . . . Prof. Dr. Stefanie Dipper

Koreferent: . . . . . Prof. Dr. Barbara Plank

Tag der mündlichen Prüfung: 20.06.2018

Abstract

With the increasing availability of digitized resources of historical documents, interest in effective natural language processing (NLP) for these documents is on the rise. However, the abundance of variant spellings makes them challenging to work with both for human users and for NLP tools. Normalization to contemporary spelling is often proposed as a solution. This work investigates the suitability of a neural encoder–decoder architecture for automatic normalization of historical language data. The neural network is extensively tuned and improved by the application of techniques such as beam search and model ensembling. Nonetheless, in a large-scale evaluation on datasets from eight different languages, the proposed model is usually outperformed by a previously established method using character-based statistical machine translation (CSMT).

Historische Dokumente werden zunehmend in digitalisierter Form verfügbar gemacht. Häufig sind sie jedoch durch eine Fülle von Schreibvarianten gekennzeichnet, welche die Anwendung computerlinguistischer Methoden (bzw. NLP-Tools) schwierig gestalten. Ein häufig verwendeter Ansatz ist die Normalisierung dieser Varianten auf moderne Schreibweisen. Die vorliegende Arbeit untersucht die Anwendung neuronaler Encoder-Decoder-Modelle für die automatische Normalisierung historischer Sprachdaten. In einer umfassenden Auswertung auf historischen Korpora in acht verschiedenen Sprachen zeigt sich, dass das verwendete Modell – trotz zahlreicher Anpassungen und Verbesserungen wie z.B. Beam Search und Ensembling – meist eine schlechtere Normalisierungs-Genauigkeit hat als etablierte Methoden, die auf statistischer maschineller Übersetzung beruhen.

Acknowledgements

When I began to work on this thesis, I wondered how other graduates could fill their acknowledgements with so many people. Five years later, I know.

First and foremost, I would like to thank my supervisor, Stefanie Dipper, for her continuous support. Stefanie always encouraged me and provided both the necessary guidance and freedom to pursue my research ideas. Without her, I might not have had a lot of the opportunities and experiences that brought me to where I am today. So, a heartfelt thank-you!

Thanks also go to my former colleagues in Bochum, particularly Julia Krasselt, Florian Petran, and Adam Roussel, for countless brisk and inspiring discussions. Special thanks to Katharina Bort and all student annotators who provided me with data for my first experiments and always accommodated my occasional "urgent" requests.

My cordial thanks to Barbara Plank for agreeing to co-supervise this thesis and for many helpful comments. Sincere thanks also go to Anders Søgaard for hosting me for a research visit and sharing many valuable insights on neural networks. More generally, thanks to everyone who provided comments, shared their datasets, or discussed my research with me; this list of names will certainly be incomplete, but includes Fabian Barteld, Joachim Bingel, Christian Chiarcos, Grzegorz Chrupała, Tomaž Erjavec, Dirk Hovy, Bryan Jurish, Katharina Kann, Nikola Ljubešić, Rita Marquilhas, Eva Pettersson, Paul Rayson, Yves Scherrer, Uwe Springmann, and everyone who came up to me after a conference talk or at a poster presentation. Further thanks go to Johannes Bjerva, Ana Valeria González, João M. Martins, and Kaja Verhoeven for assisting me with translations.

Finally, thanks to all of my family and friends—Juliane, Georg, Melanie, Miriana, Tobias, Dominika, Max, Marlene, to name just a few—for making my life easier at the right times, for distracting me from my work when I needed it, and for putting up with my moods when I hit yet another obstacle during my research. A PhD can certainly be an arduous journey, and it is best not done alone.

Contents

List of Acronyms

CER character error rate

CSMT character-based statistical machine translation

HNR historical/normalized type ratio

IR information retrieval

LCS longest common subsequence

LSTM long short-term memory

MFN most frequent normalization

MLP multi-layer perceptron

MTL multi-task learning

NLP natural language processing

OCR optical character recognition

PMI pointwise mutual information

POS part-of-speech

pp percentage points

RNN recurrent neural network

ROPE range of practical equivalence

sHNR standardized historical/normalized type ratio

SMT statistical machine translation

tf–idf term frequency–inverse document frequency

TPE tree-structured Parzen estimator

TTR type/token ratio

List of Figures

List of Tables

Zusammenfassung

Durch eine zunehmende Anzahl von Digitalisierungsprojekten werden mehr und mehr historische Dokumente einer breiten Öffentlichkeit zugänglich gemacht. Handschriften und Bücher, die vormals nur persönlich in Bibliotheken zu begutachten waren, können nun in digitaler Form leicht verbreitet werden. Praktisch nutzbar sind diese Ressourcen vor allem, wenn sie Dokumente nicht bloß in Bildform enthalten (z.B. als Scan), sondern auch in (maschinenlesbarer) Textform bereitstellen. Das ermöglicht etwa die Volltextsuche nach bestimmten Schlüsselwörtern innerhalb der Daten oder ihre Weiterverarbeitung mithilfe von NLP-Tools,[1] z.B. zur automatischen Wortarten-Annotation (POS-Tagging[2]).

Ein großes Hindernis dabei ist jedoch oft die sprachliche Variation. Je älter die historischen Dokumente sind, desto mehr weichen sie üblicherweise von der heutigen Standardsprache ab. Dies kann alle sprachlichen Ebenen betreffen, wie etwa die Syntax, die Morphologie oder das Lexikon. Die größte Bedeutung kommt jedoch oft der *Schreibvariation* zu. Die standardisierte Orthographie ist in den meisten Sprachen eine recht junge Erfindung; historische Texte sind nicht selten geprägt von unzähligen Schreibvarianten, die etwa von dialektalen Einflüssen oder den individuellen Präferenzen des Verfassers bzw. Schreibers stammen können. Laing (1994, S. 123) berichtet etwa, dass in einem Korpus des spätmittelalterlichen Englisch[3] über 500 Varianten der Präposition *through* ‘durch’ dokumentiert sind.

Eine hohe Frequenz von Schreibvarianten reduziert die praktische Nutzbarkeit der Daten ungemein: so ist es etwa für eine Volltextsuche äußerst hinderlich, alle möglichen Schreibvarianten des gesuchten Wortes kennen und explizit angeben zu müssen. Auch die Genauigkeit von NLP-Tools, die auf Daten der modernen Standardsprache trainiert sind, nimmt auf historischen Daten meist deutlich ab (vgl. Rayson u. a., 2007; Scheible u. a., 2011b). Dass Schreibvarianten dabei eine besonders zentrale Rolle einnehmen, wird unter anderem daran deutlich, dass sich schon frühe Arbeiten zur computergestützten Analyse historischer Texte mit dem Problem der Schreibvariation beschäftigen (z.B. Fix, 1980; Koller, 1983; Klein, 1991).

Für die praktische Nutzbarkeit von historischen Texten ist eine effiziente Behandlung von Schreibvariation daher von höchstem Interesse.

Normalisierung

Eine Möglichkeit der Behandlung von Schreibvariation in historischen Texten ist die der *Normalisierung*. Damit ist gemeint, verschiedene Schreibvarianten desselben Wortes auf eine

[1] NLP = Natural Language Processing
[2] POS-Tagging = Part-of-speech-Tagging
[3] LALME; A Linguistic Atlas of Late Mediaeval English

eindeutige *normalisierte Form* abzubilden. In der Praxis ist dies zumeist die äquivalente Wortform in moderner Orthographie; so könnten etwa die historischen Schreibungen *fraw*, *frauwe*, *fraẅ*, *frowe*, *vrawe* usw. allesamt auf die moderne Wortform *Frau* abgebildet werden. Ähnliche Ansätze sind in der Literatur auch unter den Begriffen *Modernisierung* und *Kanonikalisierung* zu finden.

Die Normalisierung auf moderne Standardschreibung bietet viele Vorteile:

1. Sie reduziert die Schreibvarianz, was jeglichen NLP-Anwendungen auf diesen Daten zugutekommt.
2. Sie vereinfacht Suchanfragen und erleichtert Nutzern das Verständnis der Daten, da spezielle Kenntnisse der historischen Schreibpraxis nicht mehr zwingend erforderlich sind.
3. Sie vereinfacht bzw. ermöglicht die Anwendung von Tools und Ressourcen, die für die moderne Standardsprache entwickelt wurden, auf den historischen Daten.

Bei all diesen Punkten ist zu beachten, dass eine Normalisierung oft nicht alle Nuancen und Besonderheiten der historischen Sprachstufe adäquat wiedergeben kann. Daher sollte sie keinesfalls als „Ersatz“ für die historischen Schreibungen angesehen werden, sondern als *zusätzliche* Annotationsebene bzw. als Hilfsmittel für den Nutzer und für NLP-Anwendungen.

Die Herangehensweise, eine Wortform auf ihre „moderne Standardschreibung“ zu normalisieren, birgt in der Praxis einige Tücken. So stellt sich z.B. in stark flektierenden Sprachen wie dem Deutschen die Frage, ob bei der Normalisierung auch Flexionsanpassungen vorgenommen werden sollen. Soll beispielsweise die Phrase *alle ſtain* als *alle Stein* oder *alle Steine* normalisiert werden? *Stein* ist die orthographisch ähnlichste moderne Wortform zu *ſtain*; aus dem Kontext ergibt sich jedoch, dass wir im Neuhochdeutschen hier *Steine* erwarten würden. Bisherige Forschungsprojekte haben diese Frage unterschiedlich beantwortet; so wählt das Anselm-Korpus (Schultz-Balluff und Dipper, 2013a; Wegera, 2014) etwa ersteren Weg, während das RIDGES-Korpus (Odebrecht u. a., 2016) die zweite Lösung bevorzugt. Weitere häufige Probleme sind etwa die Behandlung von Eigennamen oder von extinkten Wortformen (z.B. *Zehern* für neuhochdeutsch *Tränen*).[4]

Automatisierung

Erkennt man prinzipiell den Nutzen einer Normalisierung an, so ist die nächste Frage, mit welchen Methoden eine Normalisierung automatisch erzeugt werden kann. Bisherige Ansätze zu diesem Thema lassen sich grob in folgende Bereiche einteilen:[5]

1. Tokenbasierte Ersetzung mit Hilfe eines „Wörterbuchs“ (Rayson u. a., 2005; Bollmann, 2012).
2. Regelbasierte Verfahren, die entweder mit manuell definierten Regeln arbeiten (Fix, 1980; Koller, 1983) oder Regeln automatisch aus Trainingsdaten ableiten können (Ernst-Gerlach und Fuhr, 2006; Bollmann u. a., 2011b).

[4] Kapitel 2 widmet sich diesen grundsätzlichen Fragen der Normalisierung ausführlich.
[5] Kapitel 4 bespricht diese Ansätze ausführlicher.

3. Anwendung von Distanzmaßen, um mit Hilfe eines Lexikons die moderne Wortform mit der geringsten Distanz zur historischen Ausgangsform zu finden (Robertson und Willett, 1993; Kempken u. a., 2006; Pettersson, Megyesi und Nivre, 2013).

4. Statistische Verfahren zur maschinellen Übersetzung (character-based statistical machine translation, CSMT), die auf Buchstabenebene angewandt werden, um eine historische Wortform in eine Normalisierung zu „übersetzen“ (Pettersson, Megyesi und Tiedemann, 2013; Scherrer und Erjavec, 2013; Ljubešić u. a., 2016b).

5. Lernverfahren aus dem Bereich der neuronalen Netze, oft auch als „Deep Learning“ bezeichnet (Bollmann u. a., 2017; Korchagina, 2017).

Insbesondere die Anwendung von neuronalen Netzen für diese Aufgabe ist noch vergleichsweise wenig erforscht; dabei haben neuronale Netze in den letzten Jahren enorme Popularität erlangt und in sehr vielen NLP-Anwendungen gute bis herausragende Ergebnisse erzielt (vgl. Goldberg, 2017, Abs. 1.3 für eine umfassende Übersicht).

Die ausführliche Untersuchung eines neuronalen Netzes für die automatische Normalisierung ist daher der Kernpunkt dieser Arbeit. Für einen Vergleich mit bereits etablierten Systemen wähle ich das Norma-Tool (Bollmann, 2012), welches Verfahren aus den Bereichen 1–3 implementiert, sowie das Tool cSMTiser (basierend auf Ljubešić u. a., 2016b), welches den CSMT-Ansatz (Bereich 4) benutzt.

Korpora

Um Systeme, die mit Ansätzen des maschinellen Lernens arbeiten, einsetzen zu können, werden zunächst *Trainingsdaten* benötigt. Auch für die effiziente maschinelle Evaluation eines automatischen Normalisierungsverfahrens sind manuell geprüfte „Golddaten“ unerlässlich. Um alle hier untersuchten Verfahren auf einer möglichst diversen Menge historischer Texte trainieren und evaluieren zu können, ziehe ich historische Korpora aus acht verschiedenen Sprachen heran: Deutsch, Englisch, Isländisch, Portugiesisch, Schwedisch, Slowenisch, Spanisch, sowie Ungarisch.[6]

Die Korpora decken unterschiedliche Zeiträume vom 14. bis zum 19. Jahrhundert ab, enthalten unterschiedliche Textgenres wie z.B. religiöse und wissenschaftliche Abhandlungen, amtliche Dokumente, oder persönliche Korrespondenzen, und haben einen Umfang von ca. 55.000 bis 325.000 Wörtern (vgl. Tabelle 3.1).

Die neuronale Encoder-Decoder-Architektur

Neuronale Netze bilden eine Klasse von maschinellen Lernverfahren, die auf einer Verkettung vieler einzelner, meist nicht-linearer, Funktionen („künstliche Neuronen“) basieren. Diese

[6] Insgesamt ergeben sich zehn verschiedene Datensets, da ich für das Deutsche gleich zwei Korpora heranziehe und das slowenische Korpus in zwei verschiedene Sprachstufen geteilt ist. Kapitel 3 widmet sich der detaillierten Beschreibung aller Korpora.

Funktionen haben Parameter („Gewichtungen“), die während des Trainings modifiziert werden. Für eine allgemeine Einführung empfiehlt sich Chollet (2017) oder Goldberg (2017).[7]

Encoder-Decoder-Modelle sind insbesondere im Bereich der maschinellen Übersetzung populär geworden (Cho, Merrienboer, Gülçehre u. a., 2014; Sutskever u. a., 2014). Sie bestehen im Wesentlichen aus zwei Komponenten:[8]

1. dem *Encoder,* der eine Eingabesequenz erhält und in einen numerischen Vektor umwandelt bzw. *kodiert;* und

2. dem *Decoder,* der die vom Encoder erzeugte Vektorrepräsentation erhält und in eine Ausgabesequenz *dekodiert.*

Im Fall der Normalisierung ist die Eingabesequenz eine historische Wortform, die als Folge von einzelnen Buchstaben bzw. Zeichen repräsentiert wird; die Ausgabesequenz ist entsprechend die zugehörige normalisierte Wortform. Dieses Modell wird in Abbildung 6.2 illustriert.

Eine Schwierigkeit beim Einsatz neuronaler Netze besteht darin, die optimale Architektur für einen gegebenen Einsatzzweck zu finden. Ich beschränke mich hier auf die Untersuchung von Encoder-Decoder-Modellen, die aus sogenannten LSTM-Komponenten[9] (Hochreiter und Schmidhuber, 1997) bestehen. Dabei führe ich eine ausgiebige *Hyperparameter-Optimierung* durch, um die Anzahl an LSTM-Ebenen in Encoder und Decoder, ihre Dimensionalität (d.h. die Anzahl künstlicher Neuronen pro Ebene), sowie die Parameter „Dropout“ und „Learning Rate“ des Trainingsverfahrens zu optimieren.[10]

Des Weiteren untersuche ich eine Reihe potentieller Verbesserungen des Encoder-Decoder-Modells: den Attention-Mechanismus, das Beam-Search-Decoding, sowie die Konstruktion eines Ensembles aus fünf unabhängig voneinander trainierten Modellen. Ich zeige anhand von Evaluationen auf einer Teilmenge der historischen Datensets, dass alle diese Mechanismen im Schnitt die Akkuratheit des Modells (gemessen als prozentualer Anteil der korrekt normalisierten Wörter) verbessern. Die Verwendung eines lexikalischen Filters, der nur die Erzeugung von Wörtern aus einem vorgegebenen modernen Lexikon erlaubt, zeigte hingegen nur in Einzelfällen einen positiven Effekt.

Multi-Task Learning

Neben dem oben beschriebenen Modell, welches jeweils auf einem einzelnen Datenset trainiert und evaluiert wird, experimentiere ich außerdem mit dem Training auf zwei Datensets gleichzeitig.[11] Dies geschieht mit Methoden des *Multi-Task Learning* (MTL; Caruana, 1993), welches auf der Idee basiert, dass ähnliche oder verwandte Aufgaben besser zusammen gelernt werden können als unabhängig voneinander.

[7] In dieser Arbeit gibt Kapitel 5 eine grundlegende Übersicht.
[8] Das Encoder-Decoder-Modell wird in Kapitel 6 behandelt.
[9] Long Short-Term Memory; eine häufig verwendete Komponente neuronaler Netze, die speziell zur Verarbeitung langer, sequentieller Eingaben (z.B. Wort- oder Zeichenfolgen) entwickelt wurde.
[10] Vgl. Abschnitt 6.2.5 für eine Kurzbeschreibung der letztendlich verwendeten Konfiguration.
[11] Dieser Ansatz wird in Kapitel 8 behandelt.

Im Kontext dieser Arbeit betrachte ich die Normalisierung auf unterschiedlichen Sprachen bzw. Datensets als „verwandte Aufgaben“ im Sinne des MTL. Hierzu teste ich drei verschiedene Ansätze, MTL mit dem oben vorgestellten Encoder-Decoder-Modell umzusetzen. Der beste dieser Ansätze modifiziert das Modell, indem für jedes Datenset ein separater „Prediction Layer“ verwendet wird; dies ist die letzte Ebene des Decoders, welche die normalisierte Zeichenfolge vorhersagt. Wird das Modell beispielsweise auf Englisch und Spanisch gleichzeitig trainiert, so durchlaufen die Daten für beide Sprachen denselben Encoder und Decoder, mit Ausnahme der allerletzten Ebene, welche sprachspezifisch ist. Dies zwingt sowohl den Encoder als auch den Decoder, sprachunabhängige Repräsentationen zu lernen, während durch die separaten „Prediction Layer“ einem Teil des Netzes ermöglicht wird, sprachspezifische Transformationen zu lernen. Abbildung 8.1 illustriert dieses Modell.

Ich evaluiere diesen MTL-Ansatz auf allen paarweisen Kombinationen der historischen Datensets. Ziel dieser Evaluation ist es, herauszufinden, (i) ob das parallele Training auf zwei Datensets bessere Modelle hervorbringt, und (ii) welche Datensets dabei am meisten voneinander profitieren. Es stellt sich heraus, dass MTL nur in Einzelfällen hilfreich ist. Hauptkriterium dabei ist die Größe des Trainingssets: kleinere Datensets können stark vom MTL-Verfahren profitieren, während die Modelle bei größeren Datensets mit MTL oft sogar schlechtere Normalisierungen produzieren. Die spezifische Kombination der Datensets – ob also z.B. verwandte oder weiter entfernte Sprachen kombiniert werden – scheint dabei nur eine untergeordnete Rolle zu spielen.

Evaluation

In einer vergleichenden Evaluation trainiere und evaluiere ich Norma, cSMTiser, und das vorgestellte Encoder-Decoder-Modell separat auf jedem der zehn historischen Datensets.[12] Das primäre Evaluationskriterium ist dabei die „Word Accuracy“, d.h. der prozentuale Anteil der korrekt normalisierten Wörter. Ergebnis dieser Evaluation ist, dass alle drei Ansätze oft recht nah beieinander liegen, das CSMT-Modell jedoch auf fast allen Datensets das beste Ergebnis liefert. Die absolute „Word Accuracy“ liegt bei CSMT je nach Datenset zwischen 87% und 96% (vgl. Tabelle 10.2).

Neben dieser quantitativen Beurteilung der Normalisierungsmethoden widme ich mich auch der Frage, wie die Qualität der fehlerhaften Normalisierungen genauer bewertet werden kann, und ob die verschiedenen Methoden sich in dieser Hinsicht unterscheiden. Eine manuelle Fehleranalyse von zufälligen Stichproben fehlerhafter Normalisierungen zeigt, dass oft mehr als die Hälfte dieser Fälle durchaus brauchbare Vorschläge enthalten. So wird z.B. das deutsche *fůchent* plausibel als *suchend* normalisiert, aber dennoch als Fehler gewertet, da in den Korpusdaten *sucht* als Gold-Normalisierung angegeben ist.

Als besonders vielversprechender Ansatz stellt sich in diesem Zusammenhang das *Stemming* heraus, d.h. die Reduktion von Wortformen auf ihre Wortstämme. Eine Evaluation auf Basis der Wortstämme kann Aufschluss darüber geben, wieviele Normalisierungsfehler lediglich auf Unterschiede in Flexionsendungen o.ä. zurückzuführen sind. So zeigt sich z.B., dass dies für

[12] Kapitel 7 präsentiert ausführliche Vergleiche und Analysen der verschiedenen Normalisierungsansätze auf Development-Daten, während Kapitel 10 die Ergebnisse durch eine Evaluation auf Test-Daten verifiziert.

das Spanische auf bis zu 40% der fehlerhaften Normalisierungen des Encoder-Decoder-Modells zutrifft. Ein automatischer Stemming-Algorithmus ist für viele Sprachen verfügbar, weshalb diese Evaluationsmethode meist mit wenig Aufwand angewandt werden kann.

Eine weitere Analyse legt nahe, dass das Encoder-Decoder-Modell von einer besseren Modellierung der normalisierten Wortformen, z.B. durch ein zeichenbasiertes Sprachmodell oder bidirektionales Decoding, profitieren könnte. Ansonsten zeigen die meisten Analysen jedoch vor allem Unterschiede zwischen den Datensets auf, während die einzelnen Normalisierungsmethoden sich dabei weniger unterscheiden. So gibt es beispielsweise eine große Schnittmenge zwischen den fehlerhaften Normalisierungen des neuronalen Netzes und cSMTiser, d.h. beide Methoden haben größtenteils dieselben Wortformen richtig zu normalisieren gelernt.

Zuletzt evaluiere ich die Methoden in einem Trainingsszenario mit nur 5.000 Tokens pro Datenset.[13] Ziel dabei ist, herauszufinden, wie die Methoden sich bei einer geringeren Menge an Trainingsdaten verhalten, wie sie in der Praxis sehr wahrscheinlich ist. Als Ergebnis kann man festhalten, dass das Encoder-Decoder-Modell hier deutlich besser abschneidet, insbesondere wenn es mit Multi-Task Learning und lexikalischem Filter verwendet wird. Das CSMT-Modell liefert jedoch auch hier äußerst gute Resultate.

Fazit

Ich untersuche in dieser Dissertation die Anwendbarkeit eines neuronalen Encoder-Decoder-Modells auf die automatische Normalisierung historischer Texte. Dabei verwende ich historische Korpora aus acht verschiedenen Sprachen, um das Modell zu evaluieren und mit anderen Ansätzen zu vergleichen. Trotz ausgiebiger Optimierung des neuronalen Netzes durch Hyperparameter-Tuning und zusätzliche Techniken wie Attention-Mechanismus und Ensembling, die jeweils für sich betrachtet das Modell allesamt verbessern, erzielt es letztendlich keine Verbesserungen gegenüber einem bestehenden Ansatz, der auf statistischer maschineller Übersetzung (CSMT) basiert.

Obwohl dies in gewisser Weise ein „Negativresultat“ für das präsentierte Encoder-Decoder-Modell ist, glaube ich, dass diese Dissertation nicht zuletzt aus folgenden Gründen einen wichtigen Beitrag leistet:

1. Sie präsentiert die umfangreichste Evaluation automatischer Normalisierung, die – nach meinem besten Wissen und Gewissen – bis dato durchgeführt wurde, sowohl in Bezug auf die getesteten Sprachen als auch auf die verglichenen Methoden.
2. Die Ergebnisse stehen in Kontrast zum derzeit in der Literatur zu beobachtenden Trend, dass neuronale Netze auf zahlreichen NLP-Anwendungen den „klassischen“ Methoden überlegen sind, z.B. im Bereich der maschinellen Übersetzung (Bahdanau u. a., 2014; Wu u. a., 2016) oder auch der historischen Normalisierung (Bollmann u. a., 2017; Korchagina, 2017).
3. Die Arbeit erforscht neuartige Methoden zur (qualitativen) Analyse der automatischen Normalisierung, von der zukünftige Arbeiten profitieren können.

[13] Dieses Szenario wird in Kapitel 9 eingeführt.

Foreword

or How to read this thesis

This publication is a slightly revised and updated version of my PhD thesis, which was first submitted in February 2018. Like most PhD theses, it is a lengthy work that most people would probably not want to read from start to finish. Nonetheless, I believe that several parts of this work can be useful to various audiences. If you are reading this and are pressed for time, here are my thoughts as to which parts might be most interesting to you.

- **If you are building or working with historical corpora** and want to know more about *what* normalization is, *why* we should do it, and *how* existing corpora of historical documents handle it, you might be particularly interested in Chapter 1 (for a general overview), Chapter 2 (for an in-depth look at the normalization task), and Chapter 3 (for a concrete discussion of the historical datasets I used in my experiments).
- **If you want to perform normalization automatically** and would like to know what approaches there are and how they perform, take a look at Chapter 4 (for a systematic overview of previous work), the summary of my comparative analysis in Sec. 7.8, as well as the evaluation in Chapter 10 (for some concrete numbers).
- **If you are interested in neural networks for NLP or building better automatic normalization systems,** you might want to read about the encoder–decoder model I used (Chapter 6), the quantitative and qualitative analysis of its normalization performance (Chapter 7), and possibly my evaluation of multi-task learning and low-resource scenarios (Chapters 8 and 9). For an even quicker overview of my results and challenges for future work, take a look at the conclusion in Chapter 11.

Finally, most chapters end with a summary section that recaps their main findings.

Translations of non-English quotes and of examples from the datasets, when they appear, are mostly my own. For languages that I did not have sufficient knowledge of, I consulted native speakers or based my translation on a careful consultation of multiple dictionary and/or translation resources. Nonetheless, it is possible that some translations—particularly of normalization examples—ended up being not completely accurate. All remaining mistakes or inaccuracies are solely my own.

Chapter 1

Introduction

Ich woulde nu an defer ſtunt
Gerne hoiren van dynes ſelues munt
Wat dyme kynde zu leyde is geſcheyn
Dattu mit dynen ougen hais geſeyn[1]
— From the *N1509* text of the *Anselm Corpus*

The ongoing digitization efforts of libraries and researchers are making more and more historical documents available to the general public. Written records that date back hundreds of years are usually kept and preserved at libraries; however, sometimes the access to these documents is highly restricted because, e.g., the paper might be brittle, or already damaged due to water, mold, or other external influences. Consequently, actually viewing and working with these records can be a difficult task. Digitization does not only help to preserve these documents, but also to easily distribute them in electronic form.

Many research projects aim to create digital editions of historical documents that do not simply consist of scans of the pages, but are provided in textual form, potentially with additional annotations such as part-of-speech (POS) tags. The advantages of such resources are numerous: (i) they enable reception of the texts without the difficulties of reading old typefaces or handwriting; (ii) they enable search queries to be performed on the texts, either based on word forms or on linguistic features; (iii) they allow for automatic analyses of the texts, e.g., stylistic analysis, authorship attribution, or analysis of linguistic structures; (iv) they allow for comparative and diachronic analyses of language development; and so on. All of these factors open up new ways of doing humanities research that has traditionally been carried out by manual examination and analysis of physical documents, giving rise to the umbrella term *digital humanities* that has been growing in popularity in recent years (e.g., Svensson, 2010; Berry, 2012; Berry and Fagerjord, 2017).

Examples for historical corpora or research projects in the digital humanities are plentiful and span many different languages, time periods, text genres, and research questions. The *ARCHER* corpus is a representative corpus of historical English texts from multiple genres (Biber et al., 1994);[2] the *Corpus of Historical American English (COHA)* is a balanced corpus for investigating language change and American culture and society (Davies, 2012);[3] the *Anselm Project*

[1] "I would now, in this hour, / like to hear from your own mouth / what suffering has befallen your child / that you have seen with your own eyes." (Saint Anselm speaking to the Virgin Mary; my translation.)
[2] http://www.projects.alc.manchester.ac.uk/archer/
[3] https://corpus.byu.edu/coha/

creates a digital resource of all German records of a specific medieval treatise (Schultz-Balluff and Dipper, 2013b);[4] various *Reference Corpora* have been and are being created for various stages of historical German;[5] the *InterGramm* project investigates the language elaboration of Middle Low German;[6] the *Gender and Work* project researches the living conditions of men and women in medieval Sweden by analyzing historical documents (Fiebranz et al., 2011);[7] the *P.S. (Post Scriptum)* project provides a collection of private letters from Early Modern Portuguese and Spanish;[8] the *goo300k* project constructs a corpus of historical Slovene (Erjavec, 2012);[9] and many more such projects exist.

Transcribing, annotating, and/or analyzing a large amount of historical documents is typically performed with the aid of natural language processing (NLP) tools, as performing these task manually is immensely time-consuming and therefore often not feasible. The *Reference Corpus of Middle High German,* for example, limits some of its texts to extracts of 20,000 tokens, even though it is the result of multiple research projects spanning more than a decade of work, and some software-based automatization was already used in the process (Klein and Dipper, 2016). This illustrates how difficult it would be to make the *entirety* of documented texts in Middle High German available in this way; and the number of historical documents from Early New High German is yet considerably higher. More and better NLP tools for this type of data are required to make the greater part of historical documents accessible for further research.

Note that "historical text" in this context is not a universally defined category. While the Swedish *Gender and Work* corpus covers texts from the early 16th century to about 1800, the *Corpus of Historical American English* contains texts from the 1810s to the 2000s. The date range of a resource can be influenced by specific research interests—e.g., analyzing a particular language stage—or by the existence (and accessibility) of suitable documents. From an NLP perspective, text typically becomes more challenging to process the more different it is from the contemporary language. Since language evolves slowly over time, this typically means that text will be more difficult to process the older it is. For the purposes of my analyses, I will not consider texts that are younger than the 19th century—though this is arguably an arbitrary boundary.

I hope to have provided some background and motivation for what historical text is, what the benefits are of making these texts available in digitized, textual form, and why NLP tools are a crucial component for this. In the following sections, I will discuss the specific challenges for NLP when working with historical data (Sec. 1.1), the potential ways to address them (Sec. 1.2), and which particular approach I will investigate within this thesis (Sec. 1.3). Finally, Sec. 1.4 summarizes the main contributions of this thesis, and Sec. 1.5 gives an overview of its structure.

[4] https://www.linguistics.rub.de/anselm/
[5] Old German: http://www.deutschdiachrondigital.de/home/?lang=en
Middle High German: https://www.linguistics.rub.de/rem/
Low German: https://vs1.corpora.uni-hamburg.de/ren/index_en.html
Early New High German: http://www.ruhr-uni-bochum.de/wegera/ref/
[6] https://www.uni-paderborn.de/en/research-projects/intergramm/project/
[7] http://gaw.hist.uu.se/?languageId=1
[8] http://ps.clul.ul.pt/
[9] http://nl.ijs.si/imp/index-en.html

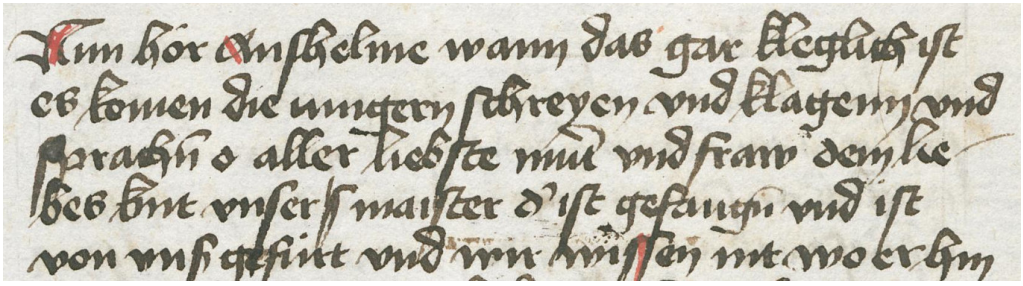


Figure 1.1: Extract from a manuscript of 15th century German, showing parts of text *B3* of the *Anselm Corpus* (Source: Staatsbibliothek zu Berlin – PK; http://resolver.staatsbibliothek-berlin.de/SBB00009D8D00000000)

1.1 Challenges for NLP on historical data

Before historical texts can be processed by NLP tools, they need to be digitized in textual form. "Textual" is used here primarily as an opposite to "graphical", i.e., scans or other photographic reproductions of physical pages. Textual representations can be obtained either from manual transcriptions or by optical character recognition (OCR) tools applied to scans of the documents; the latter approach comes with its own set of problems, though, e.g., when historical typefaces are used or the source document is a manuscript, as in Figure 1.1.[10] While this step is of course crucial in obtaining a digitized representation of a text, I will not consider it further here, rather focussing on the challenges that come afterwards.

After converting the historical document to digital text, we can—in principle—start applying NLP tools to it. This could be search tools, POS taggers, syntactic parsers, named entity recognition software, etc., usually depending on the type of research question we are interested in. In practice, however, this proves to be challenging, as most existing NLP tools are developed for *contemporary* languages. Historical language often differs significantly from its modern equivalent in several aspects that make a naive application of these tools problematic. Rayson et al. (2007) report that the accuracy of an English POS tagger dropped from 96% to 82% when applied to data from Early Modern English. Similarly, Scheible et al. (2011b) obtain a POS tagging accuracy of only 69.6% when applying a Standard German POS tagger to Early Modern German texts.

First of all, there is the problem of tokenization: it is common for NLP tools to expect their input to be tokenized and divided into sentences. For modern text, this usually involves splitting off punctuation marks from words, using spaces to split a text into tokens, and disambiguating sentence-final punctuation from other types (e.g., abbreviation markings).[11] In historical text, punctuation marks may be used quite differently from modern conventions; e.g., in the *Anselm Corpus,* there is often no sentence-ending punctuation at all. Interword spacing is also not as straightforward: e.g., a compound word might be written with or without a space between its constituents; and particularly in the case of manuscripts, spacing can also be influenced by spatial constraints of the page (Bollmann et al., 2011a).

[10] For a detailed discussion of digitization, see Chapter 4 of Piotrowski (2012).

[11] This is a slightly simplified description, and some other steps can be included in tokenization as well, such as splitting up some morphologically complex units (e.g., splitting *don't* into *do n't*).

Historical language can also differ from its modern variant in various linguistic aspects such as syntax, semantics, morphology, and lexicon. Inflectional processes may have changed; semantics of certain words may have shifted; lexemes may have become extinct; and so on.[12] Naturally, all of these issues can make it challenging to apply tools or resources for modern language to this type of data. The most prevalent issue, however, is arguably that of spelling variation, which I will discuss in the following section.

1.1.1 Spelling variation

Since historical language has typically not yet undergone a standardization process, it is not uncommon to find many different spelling variants for the same word form. Laing (1994) writes:

> With Middle English we are dealing with periods when there was no generally accepted standard written variety of the vernacular manifesting stable and consistent orthographic conventions. What we would identify as answering to 'one and the same word' in Modern English may appear in a medieval text in many different forms. Sometimes the variety is astonishing; the data in *LALME*[13] indicate, for instance, that for *it*, the personal pronoun, 45 different forms are recorded, *she* has 64 and the preposition *through* more than 500. (Laing, 1994, p. 123)

This variety of spellings can, at least to some degree, be observed for most historical languages. For example, in the Anselm dataset of Early New High German (introduced in Sec. 3.1.2), there are 36 different forms of the conjunction *und*, 47 for the personal pronoun *sie*, and 53 for the particle/preposition/adverb *zu*. The lack of an established standard means that spellings can be affected by dialectal influences or individual preferences of the writer; naturally, clerical errors can also be a factor. In many cases, this makes variant spelling the most common and substantial difference to contemporary texts.

Importantly, though, spelling variation is of a different nature than most other categories in which historical language can differ from the modern one. Lexical and grammatical changes do not make historical language *inherently* more difficult to process—on the contrary, we can argue that it could just be treated as a separate language in its own right, for which separate NLP tools can be built. Practical matters of implementation aside, there is no theoretical reason why we could not create tools that handle the syntactical, morphological, and lexical properties of historical language just as well as for modern language.

The same cannot be said for spelling variation, however. As the name implies, spelling variation introduces additional *variance*. This has consequences for almost all further processing of the data. Consider the case of POS tagging (or any other labeling task): if a given word occurs in ten different spelling variants, all of which are equally common, the amount of training data required to label all of these variants correctly increases tenfold compared to a dataset without this spelling variation. In a domain that already suffers from sparse amounts of annotated training data, this is a severe problem.

[12] In Chapter 2, I will look at these issues in more detail.
[13] A Linguistic Atlas of Late Mediaeval English

Even if we had large amounts of historical data for training our tools, treating all spelling variants of a word as separate entities means missing out on useful information: if two tokens are essentially identical except for their spelling—or, in the words of Laing (1994), "answering to 'one and the same word' "—this knowledge can and should be exploited by a learning algorithm. It should also be considered for search queries; having to know and individually specify all spelling variants in order to find all instances of a word is, after all, highly impractical.

Some of the earliest computer-assisted analyses of historical texts explicitly handle the issue of spelling variation. Usually, this is done by *normalizing* the variant spellings to a single form. Fix (1980) describes such a normalization approach as a preprocessing step for lemmatization of Old Icelandic; Koller (1983) presents a method for normalization of Old German; Klein (1991) uses an index of normalized word forms to facilitate (pre-)lemmatization of Middle High German. The focus on spelling variation in these early works again highlights the significance of this aspect.

Similar arguments can be made for the problem of tokenization: e.g., the inconsistent use of spacing can also introduce variance, and the absence or irregular use of punctuation marks can introduce ambiguity with regard to phrase or sentence boundaries. However, problematic interword spacing typically occurs much less frequently than spelling variation within a word, and it is also considerably easier to mark up manually during the transcription stage, as many corpora have done (e.g., Simon, 2014; Klein and Dipper, 2016; Odebrecht et al., 2016).

For these reasons, the aspect of *spelling variation* and how to handle it for natural language processing is the main focus of this thesis.

1.2 Possible solutions

If historical language is different enough from modern language that it poses problems for existing NLP tools, what can be done to make NLP "work" on historical data? The approaches to this problem can broadly be fit into three main categories: (i) retraining the tool; (ii) domain adaptation; and (iii) data adaptation.

The first solution is to *retrain* existing tools or machine-learning models in a supervised way on manually annotated training data from the historical domain. This requires that a sufficient amount of training data for the given task is available, which is often not the case. Besides the time-consuming aspect of manual annotation, historical data may also require expert annotators who are knowledgable in the particularities of the historical language stage.

Furthermore, manually annotated training data alone does not address the problem of increased variance. As discussed above, the inconsistencies in spelling alone increase the required amount of training data significantly. It is also unlikely that any set of labeled training data can capture all forms of spelling variants that might occur in other, previously unseen texts, meaning that models trained this way might not be able to generalize well in practice.

The second option is to view NLP for historical text as a *domain adaptation* problem. Here, we assume that historical and modern language stages are just two different domains of one language, and that we can leverage the labeled data available for the modern (or "source") domain to build tools that perform well on the historical (or "target") domain. The most common

approach is to combine labeled data from the source domain (which is assumed to be available in large quantities) with either labeled or unlabeled data from the target domain (see, e.g., Daumé III and Marcu, 2006; Daumé III, 2007). This has the benefit of reducing or eliminating the requirement to create manually annotated resources for the target domain, but also does not explicitly address the variance problem discussed above.

The third option is to perform *data adaptation*. By this, I am referring to all methods that transform the historical (target) data to make it look more similar to the modern (source) data. The most common transformation here is *(spelling) normalization*, also called *canonicalization*: the mapping of historical spelling variants to a canonical form, usually their contemporary equivalent (e.g., Piotrowski, 2012, Ch. 6). This approach eliminates the variation in spelling. The following is an example from a historical English text and its normalization in the *Innsbruck Letter Corpus* (introduced in more detail in Sec. 3.1.1):

(1) þe quene was ryght gretly displisyd with us both
the queen was right greatly displeased with us both

Often, the aim of normalization is to apply existing tools—that have been trained on modern data—to the normalized historical text. While spelling normalization is the most common form of data adaptation, it is also conceivable to perform normalization on other levels, such as morphology or syntax.

This list of options is not exhaustive: Firstly, the three approaches are not mutually exclusive and can be combined, e.g., by performing a canonicalization step to reduce the spelling variance before retraining on the canonicalized data. Secondly, many variations on these approaches are conceivable: e.g., producing artificial training data for the historical domain by "adapting" modern data (Hana et al., 2011); or explicitly modifying tools by adding knowledge of the historical domain, such as modifying the lexicon, tokenizer, and affixation module of a POS tagger (Sánchez-Marco et al., 2011). In general, though, most approaches that do more than just retraining tools on new data can be categorized as *adapting the tool, adapting the data,* or a mixture of these two.

1.2.1 Arguments for normalization

I strongly favor the data adaptation approach in the form of *normalization.* This is mainly for three reasons:

1. It addresses the issue of variance.

   This is helpful for all downstream applications, regardless of which approach is chosen, i.e., applying an existing tool to the data, retraining a tool on the normalized data, building a new specialized tool, etc.

2. It provides useful information to all users of the data, not just NLP tools.

   Normalization, when used to complement the original historical data, can provide helpful assistance for everyone working with the data, by reducing the barrier of understanding,

facilitating search queries, etc. Even if not all nuances of the original text will be preserved in a normalization, I believe there is a net benefit to be gained from it.

3. It facilitates the application of existing tools and resources.

   Tools such as POS taggers and parsers might perform reasonably well on a normalized text, depending on how different the historical language is apart from the spelling aspect; entries from lexical or semantic databases could be linked to normalized word forms; and so on.

For some researchers, the third aspect—reusing existing tools—is the main motivation for performing normalization, which in this context is only seen as an intermediate step in a larger processing chain. While I do agree this can be a useful aspect, I actually believe it is the least important of the three. Normalization can provide useful information on its own, both to NLP tools and to (human) users of the data.[14]

1.3 Automatic normalization

If normalization is a useful annotation layer for historical data, the next question is how we can produce it automatically. After all, manually normalizing a full corpus of texts can be just as time-consuming as creating any other type of annotation; shifting the annotation effort from, e.g., POS tagging to normalization achieves a reduction of the spelling variation (with all the benefits described above), but is still an inefficient way to process large amounts of texts.

1.3.1 From rules to machine translation

Automatic spelling normalization has a long history. Early approaches were usually based on hand-crafted rules that encode regular spelling transformations (e.g., Fix, 1980; Koller, 1983). These are easy to implement from a technical standpoint, but require expert knowledge of the language and are inflexible with regard to new data that might show different characteristics. Nonetheless, they can be very effective if the spelling changes are mostly regular. Later work also explores methods to derive replacement rules automatically from training data (Koolen et al., 2006; Bollmann et al., 2011b).

Many approaches are based on the idea that most historical spelling variants are "close" to their modern equivalents by some form of string distance metric, and can therefore be normalized by finding the closest modern equivalent in a lexicon (Hauser and Schulz, 2007; Jurish, 2010a; Pettersson, Megyesi, and Nivre, 2013). These come with their own drawbacks, though, as they often rely on a comprehensive lexicon of the modern target language, which is unlikely to cover all proper nouns, compounds, or other specialized vocabulary that can be encountered in a text. They also fail in cases where the underlying assumption does not hold, e.g., when a historical word form is highly similar to a modern word that is completely unrelated.

In recent years, many works on historical normalization have utilized character-based statistical machine translation (CSMT) (e.g., Sánchez-Martínez et al., 2013; Scherrer and Erjavec, 2013;

[14] Sec. 2.1 will elaborate on this aspect.

Pettersson, Megyesi, and Tiedemann, 2013; Ljubešić et al., 2016b; Schneider et al., 2017). Their approach is to reuse existing software for statistical machine translation, which has long been the state-of-the-art approach for machine translation and is therefore thoroughly tested and optimized, and model normalization as the translation of character sequences. In other words, instead of translating a sentence consisting of words as the atomic units, the technique is used to translate a word consisting of characters. In most cases, CSMT was shown to perform better than previous work; it can currently be considered the state-of-the-art approach for historical normalization.

1.3.2 Neural networks

In the area of machine learning, *neural networks* have received an enormous amount of attention in the last few years, often under the term *deep learning*. This is also true for natural language processing: according to Young et al. (2017), up to 70% of long papers at large-scale NLP conferences in 2016/2017 covered deep learning methods. Goldberg (2017) gives an introduction to neural networks for NLP and cites an impressive number of tasks where they have been successful; the following is an (incomplete) excerpt:

> Fully connected feed-forward neural networks [...] provide benefits for many language tasks, including the very well basic *[sic]* task of language modeling, CCG supertagging, dialog state tracking, and pre-ordering for statistical machine translation. [...]
>
> Networks with convolutional and pooling layers [...] show promising results on many tasks, including document classification, short-text categorization, sentiment classification, relation-type classification between entities, event detection, paraphrase identification, semantic role labeling, question answering, predicting box-office revenues of movies based on critic reviews, modeling text interestingness, and modeling the relation between character-sequences and part-of-speech tags. [...]
>
> Recurrent models have been shown to produce very strong results for language modeling, as well as for sequence tagging, machine translation, parsing, and many other tasks including noisy text normalization, dialog state tracking, response generation, and modeling the relation between character sequences and part-of-speech tags.
>
> (Goldberg, 2017, p. 4 f.)[15]

Even though they have been applied to "noisy text normalization" in the context of social media text (e.g., Chrupała, 2014), neural networks have so far rarely been used for historical normalization, despite the fact that there is an obvious candidate architecture for this task: character-based neural machine translation (Ling et al., 2015). The state of the art for machine translation has shifted from statistical to neural models (Bahdanau et al., 2014; Wu et al., 2016),

[15] The quoted passage also contains footnotes with citations for every mentioned task; they have been left out here for brevity.

so if character-based statistical machine translation (CSMT) is the state of the art for historical normalization, could character-based neural machine translation perform even better?

A potential problem is that neural networks are typically said to work best when large amounts of training data are available; e.g., Chollet (2017, Sec. 1.3) cites the availability of "very large datasets" as one of the factors for the success of deep learning. Datasets for historical normalization, on the other hand, are comparatively small. The *GerManC-GS* corpus of Early Modern German contains about 50,000 tokens (Scheible et al., 2011a); the *Reference corpus of historical Slovene goo300k* has about 300,000 tokens (Erjavec, 2012); the *HGDS corpus of Old Hungarian* has about 2.2 million tokens, but the majority of them are automatically normalized, with only a small fraction of manually checked normalizations (Simon, 2014). In comparison, Wu et al. (2016) train an English-to-French machine translation system on 36 million sentences.

However, previous work suggests that neural models may be suitable for the normalization task despite the small training sets. Bollmann and Søgaard (2016) and Bollmann et al. (2017) train neural network models on small datasets of historical German, containing between 2,000 and 11,000 tokens, and show that the resulting models can outperform the previously established Norma tool (Bollmann, 2012). Korchagina (2017) finds that a neural machine translation system trained on about 70,000 German words performs better than both Norma and a statistical machine translation system. Still, to the best of my knowledge, all work on neural historical normalization so far has only been evaluated on German datasets, and Korchagina (2017) presents the only direct comparison of neural models to CSMT.

Due to the general success of deep learning and the promising results of these previous studies, I believe that a more thorough investigation of the suitability of neural networks for historical normalization is warranted.

1.4 Aim of this thesis

My aim in this thesis is to apply an encoder–decoder neural network model, inspired by work in neural machine translation (and introduced in more detail in Chapter 6), to the historical normalization task, evaluate it on a large selection of corpora from different languages, and compare its performance with previously established normalization systems. To this end, I will do the following:

- Optimize the encoder–decoder architecture for the historical normalization task.

  This includes tuning the number of neural network layers used in the model, tuning other hyperparameters to find optimal settings for the task, and adding various techniques that are shown to mostly improve the normalization accuracy, such as the attention mechanism, beam search decoding, and model ensembling (cf. Chapter 6).

- Perform a comparative evaluation and analysis on development data.

  Evaluation is performed on ten datasets from eight languages, chosen mostly based on the availability of gold-standard normalization data (cf. Chapter 3). The output of the encoder–decoder model is compared to that of other normalization tools, which are selected to be representative of different previously established normalization methods

(cf. Chapter 4). Besides the commonly measured word accuracy, the evaluation and analysis will also focus on more fine-grained ways to assess the quality of the automatic normalization output (cf. Chapter 7).

- Investigate the models' performances under multi-task learning and low-resource training.

  Techniques from multi-task learning are used to train the encoder–decoder model on two datasets in parallel, with the aim of improving the performance on each single dataset (cf. Chapter 8). In the low-resource scenario, I repeat some of the previous evaluations and analyses after training the models on only a small portion of the datasets, simulating a common use case where only little training data is available (cf. Chapter 9). In both instances, my aim is to find out if and how the performances of the models differ compared to the full evaluation.

- Repeat selected evaluations on held-out test data to confirm the observations of the previous analyses (cf. Chapter 10).

Overall, the current state-of-the-art normalization approach based on character-based statistical machine translation (CSMT) is shown to perform better than the proposed neural network model in most scenarios, including the low-resource training scenario. The qualitative analyses provide some useful insights into the individual characteristics of the datasets, but mostly find no major differences between the models—i.e., the CSMT and neural network models mostly learn to normalize the same word forms and patterns. While this is in many ways a "negative result" for neural networks, I believe this thesis is an important contribution to the field for three reasons:

1. It is, to the best of my knowledge, the most extensive evaluation and comparison of methods for automatic historical normalization so far. It covers more datasets and languages than previous evaluations and compares normalization systems representing a broad variety of previously established approaches.

2. It runs contrary to the widely reported successes of neural networks and the associated notion that they mostly outperform "classical" approaches. It also stands in contrast to previously published results for historical normalization that found advantages for neural networks (e.g., Korchagina, 2017).

3. It explores novel ways to analyze the output of automatic normalization systems that can benefit future work.

In essence, the *thesis statement* that I am going to substantiate in the following chapters is:

> Compared to an encoder–decoder neural network architecture that has been extensively tuned and optimized, character-based statistical machine translation still remains the better overall choice for the historical normalization task, as it outperforms the neural model in most scenarios when evaluated on a diverse set of ten datasets from eight languages.

1.5 Structure of this thesis

The structure of this thesis is as follows:

- Chapter 2 discusses the normalization task. It explores what normalization in the context of historical text is and what it is not, what challenges there are for deciding on a "best" or "gold-standard" normalization, and how different corpora have chosen to address them.
- Chapter 3 presents the corpora used in my experiments. It describes the historical datasets, documents any preprocessing decisions, and explores ways to analyze and compare the spelling variation found in them. Additionally, it introduces the contemporary datasets that are used in some of the experiments.
- Chapter 4 explores methods for performing normalization automatically. It gives an overview of previous work in this area and describes the specific tools I have chosen to include in the later evaluations.
- Chapter 5 introduces neural networks. It explains the basic concepts of neural network models and how they are trained, as well as documenting any relevant implementation details.
- Chapter 6 describes the encoder–decoder architecture. It compares encoder–decoder models with and without an attention mechanism, performs hyperparameter tuning to find the best hyperparameters for the model and the training procedure, and analyzes properties of the model such as the variance of its performance under repeated training runs.
- Chapter 7 evaluates, analyzes, and compares the normalization methods on development data from the historical datasets. It explores various ways to analyze and measure the quality of errors the different models make, e.g., by comparing different evaluation measures, performing a manual error classification, applying a stemming algorithm to the data, comparing the distribution of errors, etc.
- Chapter 8 introduces multi-task learning as a way to train the encoder–decoder model on multiple datasets in parallel. It describes different architectures for multi-task learning and compares them by performing pairwise training experiments, i.e., training on two historical datasets simultaneously.
- Chapter 9 describes experiments in low-resource training, i.e., training with small datasets. It compares the different normalization methods in this scenario, investigates how their performance differs compared to the full training scenario, and examines a training setup where a small dataset is paired with a larger dataset using multi-task learning.
- Chapter 10 performs a final evaluation on held-out test data from the historical datasets. It repeats some of the experiments from previous chapters to confirm that the observed effects and conclusions remain the same when tested on previously unused parts of the data.
- Chapter 11 concludes by summarizing the results, suggesting ways to improve the neural network model presented here, and discussing possible directions for future research.

Chapter 2

Principles of normalization

> *In certain, perhaps in the majority of philological circles, normalization and its twin term, regularization, are almost taboo words, and editors who normalize their texts are liable to be ostracized.*
>
> — Markus (2000, p. 181)

Dealing with linguistic variation in historical documents has always been a concern for scholars in the humanities. Texts written in older stages of a language, often in manuscript form and influenced by the scribe's dialect, pose several challenges for contemporary readers, for instance due to the usage of unfamiliar spellings, characters, or abbreviations. It is therefore not surprising that adjusting parts of a text to ease comprehension, such as resolving abbreviations or unifying spelling variants, has a long tradition; Markus (2000) reports that this was common practice for university students of medieval German back in the 1960's.

This practice is most often called "normalization", but many related terms exist, such as "regularization", "modernization", or "canonicalization", with varying definitions regarding the extent and nature of the adjustments. Intuitively, the concept appears to be easy to define: when we see phrases like *whych he wryteth* or *aduise and councell*, we can recognize them as an archaic form of writing English, and determine that their modern-day equivalents would be *which he writes* and *advise and counsel*, respectively. In other words, a form like *wryteth* can be seen as a historical spelling variant which could be changed—or "normalized"—to *writes* in order to conform to modern spelling conventions.

Delving deeper into the topic, the concepts of variation and normalization are not always so intuitive and clear. Deviant word forms can stem from the lack of orthographic conventions, but also from phonological, morphological, or lexical differences of the writer's language—should all of these variations be smoothed over, or only some of them? How should extinct lexemes be normalized that do not have a direct modern equivalent? Syntactic and morphosyntactic variation can raise further questions, e.g., should a genitive noun phrase be changed to accusative if that is the norm of the modern language? These issues do not always just concern word forms in isolation, as changing the case or gender of a noun could necessitate similar changes in articles, adjectives, or pronouns that agree with it. Furthermore, it might not even be easily decidable whether a deviant word form is the result of a different grammatical case, phonological influences of the writer's dialect, or just a clerical error.

There is no single "best" way to handle these issues, as the most desirable treatment will always depend on the intended goal of the normalization. Scholars of medieval studies might be

content with a minimal approach that only resolves abbreviations and standardizes spelling, since they will be somewhat familiar with the morphological and syntactic peculiarities of the historical language stage. NLP practitioners looking to apply modern tools for part-of-speech tagging, syntactic parsing, or semantic analysis, might appreciate a more radical modernization of the source material.

The remainder of this chapter explores these issues in more detail. Sec. 2.1 first discusses why normalization can be useful in the context of historical texts. Sec. 2.2 looks at choices during the digitization process of historical documents and their effect on the (digitized) source data. Sec. 2.3 analyzes how normalization can be defined and how it differs from related terms such as regularization or modernization, while Sec. 2.4 takes a closer look at challenges for the normalization task and how existing guidelines and historical corpora choose to handle them. Finally, Sec. 2.5 concludes and describes which view of the normalization task I will adopt for the purposes of this thesis.

2.1 Why normalization?

Historical documents are a valuable resource for researchers in many fields, such as historians, sociologists, literary scientists, linguists, lexicologists, and others. However, the raw source material, i.e., historical manuscripts or prints, can be difficult to study: handwriting can be hard to read; they can contain abbreviations or characters that are no longer used; the spelling of words can be different from modern orthography; and they can exhibit a wide variety of other linguistic differences due to dialectal features, or simply due to language development since the time they were written. Critical editions are often used to study these texts, not only because they compile information from several different sources of the same text, but also because editorial adjustments are often made to reduce the above-mentioned difficulties, helping the reader to understand the material. Indeed, Markus (2000, p. 184) argues that "unless readers have specialised in palaeographic mediaeval practices, they need normalizing editorial help."

There are several arguments to be made not to rely on editors alone, though. First of all, editorial decisions are not always made transparent, and details of the source material may sometimes be lost in editions. The decision of what or what not to preserve is made by the editor, typically with regard to what they consider relevant for the reader. However, what is relevant strongly depends on the individual research question—a sociologist studying medieval societies will probably not be interested in linguistic peculiarities of the source text, while they might be the prime interest of a historical linguist studying language change. Normalization, if performed automatically by a software tool or provided as an annotation layer, enables scholars to work directly with the unedited source material, while still benefiting from the help of normalized word forms.

Secondly, the ongoing digitization efforts of libraries and corporations (such as Google Books) are making a large amount of historical documents readily accessible to the public, sometimes for the first time. These digital resources open up many new avenues of research that was previously impossible or difficult to do: search queries can quickly find all occurrences of a word or phrase (Rogers and Willett, 1991, e.g., Ernst-Gerlach and Fuhr, 2006); word frequency lists can be extracted automatically (Baron et al., 2009); NLP tools can provide automatic annotations

in order to quickly identify and extract certain linguistic structures, e.g., relative clauses (Hundt et al., 2012) or verb phrases (Fiebranz et al., 2011; Pettersson et al., 2014b; Krasselt, 2017); and so on. However, on texts with variant and inconsistent spelling, word form queries are more difficult to perform, word frequency lists are inaccurate, supervised machine learning models are harder to train, and tools already developed for modern language stages are likely to perform poorly. Creating manual annotations or critical editions is a time-consuming process, and simply not conceivable to do for such a large body of material.

It is important to recognize that normalization, in the sense presented here, is a tool, not intended to substitute the source material, but rather to complement it. As an added layer of annotation in a corpus or a digital edition, it can assist in reading, comprehending, and searching within a document. It facilitates automatic analyses that can be based on the normalized form. As a preprocessing step for NLP tasks, it can significantly improve their accuracy, or enable the (re-)utilization of existing tools developed for the modern standard language. As such, normalization is an important step to making historical documents accessible not only in a literal sense, but in a practical sense that enables meaningful applications to be based on them.

2.2 Digitization

Historical documents always originate in physical form, either as printed texts or as handwritten manuscripts. Before they can be processed by computational means, they therefore need to be digitized. These digitized versions consequently become the source texts that are the starting point for our normalization efforts. How the digitization is performed (e.g., OCR vs. manual transcription) and which decisions were made during this stage (e.g., level of detail to preserve) can therefore influence aspects of the normalization process.

Digitizing historical documents, at least for our purposes, means converting them to a string of text (as opposed to, say, images of the individual pages). OCR techniques can be used to extract a string representation from images of text, but they are difficult to apply to manuscripts, and also face challenges on printed historical documents (cf. Piotrowski, 2012, Ch. 4).[1] When OCR is used, there is usually a manual post-processing step (e.g., Simon, 2014)—if there is not, the normalization process will likely also have to deal with errors introduced by the OCR program, which may be of a very different nature than the variation originating from the text's original scribe.

When digitized texts are based on printed editions, they may already contain changes introduced by the editor, regularizing some aspects of the writing such as diacritics, capitalization, or common character substitutions such as ⟨j⟩ for ⟨i⟩ (Dipper and Schultz-Balluff, 2013). When multiple sources from different editors form the basis of a corpus, they might even follow different editorial principles—Simon (2014) reports that for their corpus of Hungarian codices, the editions had to be manually standardized so identical characters had a common representation. Editorial decisions can also be made by the corpus creators themselves, as in the Post Scriptum corpus, where punctuation characters have already been standardized in the transcription; e.g.,

[1] There is, however, active research on improving OCR on historical documents (e.g., see Berg-Kirkpatrick and Klein, 2014; Springmann and Lüdeling, 2017), so this situation might change in the future.

a paragraph mark in the form of a double slash is transcribed as a full stop (Vaamonde, 2017). Digitized texts that have been (pre-)edited in some way are—at least in parts—likely to be easier to normalize than those which are transcribed verbatim.

Often, historical texts are transcribed manually by experts of the source language. The level of detail preserved in these transcriptions can vary, too, especially when they are based on manuscripts, which may contain arbitrary amounts of variation of letter forms, embellishments, or abbreviation marks conceived by the writer. Furthermore, (pre-)editorial decisions similar to printed editions can also be made during this stage. Transcriptions in the Reference Corpus of Middle High German, for example, conflate the "r rotunda" ⟨ꝛ⟩ and the plain ⟨*r*⟩, while keeping the distinction between plain ⟨*s*⟩ and round ⟨*ſ*⟩ (*Korpushandbuch DDD-Mittelhochdeutsch* 2014).

There is also the issue of tokenization, i.e. marking up token boundaries. Spacing in historical texts can be just as variable as other spelling characteristics, and does not always correspond to modern usage. In many corpora, the original spacing is preserved, but exceptions from word boundaries according to modern orthography (e.g., two separate words that should be written as one word in modern writing, or vice versa) are manually marked up in some form (Simon, 2014; Klein and Dipper, 2016; Odebrecht et al., 2016). Others keep the original spacing and only encode modernized word boundaries at the normalization layer (Archer et al., 2015). It depends on these decisions whether the task of normalization also has to account for modernizing the tokenization or can already presuppose it. In manuscripts, it might not always be easy to decide when the spacing between two letters is wide enough to be recorded as a space; cases like these can be up to subjective interpretation, and for practical reasons, an editorial decision usually needs to be made (cf. Marttila, 2014, p. 440).

When digitizing historical texts, there is always a tension between the desire to preserve as many details from the source material as possible, in order to allow varied and nuanced research questions to be investigated on the data, and concerns of usability. For this reason, some corpora even offer multiple layers of representations; e.g., the Anselm corpus offers a "simplified" layer that maps certain historical characters to modern equivalents, such as mapping ⟨*ſ*⟩ to ⟨*s*⟩ (Dipper and Schultz-Balluff, 2013). The RIDGES Herbology Corpus offers a "clean" layer which performs similar mappings, but also differs in tokenization, providing modern word boundaries rather than the original ones (Odebrecht et al., 2016).

Decisions made during the digitization process can have a huge impact on the suitability of a resource for linguistic analyses. However, they are also important from an NLP perspective, since the digitized material effectively becomes the source material which all further processing is based on. If the digitized texts already come with modern token boundaries or simplified/standardized characters, this will probably facilitate further processing of the data. The downside is that results achieved on this type of data will not necessarily be transferable to new data that has not been prepared in such a way. Lastly, these factors should also be kept in mind when comparing results from different corpora, as different results might not only originate from linguistic differences, but also from these choices made during digitization.

2.3 Defining normalization

So far, we have discussed why normalization can be useful, but apart from setting the vague goal of eliminating variant word forms, we have not defined what exactly "normalization" should entail. In this section, I will discuss how the term is used in the literature, and how it relates to canonicalization, modernization, regularization, or standardization.

In its most literal sense, *normalization* can be defined as "the change of deviant word forms according to the norm of a standard" (Markus, 1997, p. 220); similar definitions can be found throughout the literature. But what should the standard be? Most authors agree that two main approaches can be distinguished (e.g., Fix, 1980; Koller, 1983; Markus, 2000):

1. defining a standard based on text-internal criteria, e.g., identifying the variant that is most commonly used, or is seen as most consistent with regard to similar word forms, in a given work or the body of all works by an author; or
2. relying on an external resource, such as a grammar or a lexicon, to provide the standard language which the normalization should adhere to.

Bowers (1989) calls the first approach *regularization* and only the latter *normalization*, and I will adopt this terminology here, though not all authors strictly follow this distinction.

Regularization in this sense reduces the variance arising from inconsistencies within a text or corpus without relying on any external resources. It lends itself to unsupervised approaches to spelling variation that aim to find or cluster all variants of a word form, but do not necessarily require them to be mapped to modern forms (e.g., Giusti et al., 2007; Barteld et al., 2015). However, when a mapping to a single variant is desired, it is not clear which of the variants should be chosen for this purpose. Simply choosing the variant that occurs most often is an option, but might not result in a regularization that shows consistent spelling characteristics across different word forms. Choosing the style of a well-known author is an alternative, but not necessarily practical, as Markus (2000) notes:

> In English, not even Chaucer's version of Middle English is homogeneous and well-known enough among Anglicists to function as the optimal norm. That role is best played by Present-day English. (Markus, 2000, p. 185)

The latter case, i.e., normalization according to an external standard, raises the question what the external standard should be. This can be a matter of which lexicon or grammar should be considered the most comprehensive or authoritative, but, more fundamentally, starts with the question whether the resource should be a *historical* or a *modern* one. Lenders et al. (1973), in their study of Middle High German texts, choose to use both: the Middle High German lexicon by Matthias Lexer (Lexer, 1992) as a reference for the historical word form; and the Duden as the reference for their Modern German equivalents. They argue that this approach circumvents difficulties due to word forms being archaic or extinct (in which case they might not be covered by the modern lexicon), or a more recent addition to the language (and therefore not yet covered by the historical lexicon). It also allows for approaching corpus analyses from the perspective of either language stage. For many NLP applications, normalization to the modern language variety will be preferable, since this allows utilizing existing resources, which will typically be more plentiful for the present-day language than for historical stages.

Jurish (2011) uses the term *canonicalization* to refer to the mapping of unknown words to "extant canonical cognates" (p. 2), which amounts to normalization with the modern language as its target. He chooses this term as it "has established itself in the domain of information processing." Baron and Rayson (2009) refer to the same concept as *standardization*, though the term itself leaves open which standard it refers to.

Modernization clearly refers to the modern language as the desired standard, and is used occasionally in this sense (e.g., Sánchez-Martínez et al., 2013; Scherrer and Erjavec, 2013). Historically, the term is also associated with more radical changes to the source material to conform to standards of a publisher (Bowers, 1989) or to provide a full translation of words or phrases (Markus, 2000). Krasselt et al. (2015) distinguish two layers of normalization that they call "normalization" and "modernization", the latter of which covers more radical adjustments that could be seen as closer to a full translation, such as providing a modern, semantically equivalent word form if the historical one underwent a significant semantic shift.

In this work, I will use the term *normalization* in its sense of normalizing to a modern standard language, which also appears to be the most common usage in the literature (e.g., Markus, 1997; Oravecz et al., 2010; Hendrickx and Marquilhas, 2011; Reynaert et al., 2012; Amoia and Martínez, 2013; Pettersson, Megyesi, and Nivre, 2013; Archer et al., 2015; Krasselt et al., 2015; Etxeberria et al., 2016; Ljubešić et al., 2016b; Schneider et al., 2017). This definition, however, only provides the guiding principle that its target word forms should conform to a modern standard—it does not define how to arrive at that standard, i.e., what criteria should be applied to perform the mapping of historical word forms to modern ones. Indeed, the extent of changes that normalization should cover is not easy to define. While Marttila (2014) speaks of a "policy of minimal intervention", the introduction of a separate "modernization" layer by Krasselt et al. (2015) shows that there is a tension between staying close to the source text and producing a normalization that is "modern enough" for its intended purpose, e.g., being understandable to a contemporary reader, or resulting in high accuracy with modern NLP tools. In the following section, I will look more closely at the various challenges for deciding on a normalized form and how different corpora and guidelines chose to approach them.

2.4 Guidelines and challenges

The most striking feature of historical texts is often their lack of a standard orthography. If normalization is the mapping of historical word forms to their modern equivalents, it seems like a simple mapping of variant spellings to their standard form in modern orthography. This assumption could hold true if orthography was the *only* feature that distinguished historical and modern texts; but language change and dialectal influences can affect all areas of language, including morphology, syntax, or semantics, and all of these will be reflected in the writing.

Furthermore, language change is continuous: there is not always a clear boundary between historical variants and correct/acceptable spellings. For example, the German dative ending *-e* in phrases such as *im Walde* 'in the forest' is considered somewhat archaic, but still perfectly understandable to contemporary speakers, and even preferred in certain fixed phrases. Normalizing *Walde* > *Wald* could therefore be seen as unnecessary; on the other hand, having two

interchangeable forms of the same morphological word (i.e., the dative singular of *Wald*) seems like the very essence of "variation" that normalization is intended to reduce.

Extinct word forms or affixes pose an even greater challenge, as they cannot be normalized to a modern word form on a purely graphematic level. It is not obvious what the appropriate normalization should be in such cases—adjusting only the spelling results in an artificial form that does not actually exist in the modern language, while replacing it with an appropriate modern lexeme or affix constitutes a change way beyond the graphematic level, and is much closer to a translation of the text.

In a supervised machine learning setting that uses existing, normalized datasets as gold-standard data, it is easy to neglect these considerations and define the task pragmatically by the content of the datasets used in the experiments. Ultimately, the goal of such an experimental setup (or, at least, the easily *measurable* goal) is to learn to reproduce the type of annotations of the training set on the data of the test set. However, since the details of the normalization task can vary between different datasets, the performance of machine learning algorithms can also vary due to that. I believe it is beneficial to be aware of these criteria when evaluating automatic normalization methods, especially when comparing results obtained on different datasets.

In the following subsections, I will examine various criteria that need to be taken into account for normalization, summarize how existing corpora and normalization guidelines choose to handle them, and discuss what impact they have for NLP applications.

2.4.1 Spelling and phonology

By far the most common type of normalization is *spelling normalization* in the strictest sense of the term: eliminating variant spellings of otherwise identical word forms. This is usually the main criterion in any normalization dataset, and such a salient one that the whole process is often called "spelling normalization".

The aim of spelling normalization is to produce word forms that conform to the modern, standard orthography of the language in question, and often, the notion of "modern" or "contemporary" orthography is simply assumed to be well-defined (e.g., Rögnvaldsson et al., 2012; Erjavec, 2015). Other datasets employ a *dictionary criterion*, stating, for example, that "the normalised version should be a word form that is likely to be present in a modern language dictionary" (Pettersson, 2016, p. 50). When a particular dictionary is specified, this establishes an objective point of reference for deciding whether a given historical word form should be normalized, and whether a given modern word form can be used as a normalization. For German, Duden[2] is often used as the dictionary of choice (Krasselt et al., 2015; Odebrecht et al., 2016); for English, Marttila (2014) uses the online version of the Oxford English Dictionary.[3]

Purely graphematic operations include normalizing character pairs that are sometimes used interchangeably, such as ⟨*i*⟩/⟨*j*⟩, ⟨*i*⟩/⟨*y*⟩ or ⟨*u*⟩/⟨*v*⟩, in word forms like German *jre* > *ihre* or the English *aduise* > *advise*. Replacing characters that are no longer in use, such as ⟨*ſ*⟩ in place of modern ⟨*s*⟩, can also fall under this category, unless these spellings were already regularized

[2] http://www.duden.de/woerterbuch
[3] http://www.oed.com/

during the digitization process (cf. Sec. 2.2). The same applies to certain abbreviations found in handwriting, such as a bar over a character used to represent a following nasal consonant (e.g., *einē* for *einen*). Diacritics are another common case; e.g., superscribed ‘e’ as in ⟨å⟩ in place of modern umlaut ⟨ä⟩ in German, or ⟨o̧⟩ in place of ⟨ő⟩ in Hungarian (Simon, 2014).

Graphematic criteria are often grouped with phonological ones, since both are closely related, and their normalizations can mostly be unambiguously determined. For the Anselm corpus, Krasselt et al. (2015, p. 17) define that normalized word forms may be derived “exclusively via phonological and/or graphematic equivalence operations.” For example, while the alternation *czu* > *zu* ‘to’ would count as a graphematic operation since it is not phonologically motivated, pairs like *zwelf* > *zwölf* ‘twelve’ or *sunne* > *Sonne* ‘sun’ are the result of vowel rounding or lowering, respectively, between Early New High German and Modern German. Vaamonde and Magro (2017) list many similar examples for Spanish and Portuguese, e.g., the common lenition of [b] to [v] reflected in writing in word pairs such as *binho* > *vinho* ‘wine’, the elision of [d] in *aministrar* > *administrar* ‘to administrate’, and many others.

An important property of spelling variation is that the correct normalization cannot always be determined without looking at word context. This can be seen in Early Modern English word pairs like *bee* > *be*, *doe* > *do*, or *then* > *than*, where we can only determine by looking at the context whether the word needs any normalization at all, i.e., whether *bee* is used as a noun describing the flying insect, or as a variant spelling of the verb *to be* (Archer et al., 2015, p. 16). Another example is the normalization of *my mistres eyes* to *my mistress's eyes* (Archer et al., 2015, p. 13), where the normalization to a genitive form is only triggered by the contextual usage of the word within the noun phrase. Common examples in German include *jrē* ‘their’, which can be either accusative *ihren* or dative *ihrem*, and *dz*, which can be either the definite article *das* or the conjunction *dass*.

Proper nouns

Proper nouns are a special case for spelling normalization: they cannot typically be found in a dictionary, raising the question of how to determine a reference spelling, and they can preserve archaisms that have not always changed with the rest of the language, so applying modern spelling rules is not always appropriate. Especially for personal names, their exact spelling can also be an individual choice.

Not all normalization guidelines explicitly mention their treatment of proper nouns, but those that do adopt a variety of positions: Archer et al. (2015) do not normalize names (such as *Darbye*, *North Baiely*) at all; Pettersson (2016), on the other hand, does normalize place names (such as *Upsala* > *Uppsala*) but not personal names, reasoning that the latter do not necessarily have unique spellings in contemporary Swedish, while place names do have a standardized modern form. Krasselt et al. (2015) and Simon (2014) both choose to normalize proper nouns, including personal names, though both are mainly concerned with religious texts—here, a modern Bible translation can often be used to provide a reference spelling.

Capitalization

The usage of capital letters can be another source of high variance. In most languages, we would expect them at the beginning of sentences or in proper nouns today, but this usage is often not established or inconsistent in older texts. While capitalization is sometimes used for highlighting, it often lacks any clear function, appearing in the middle of words, or being used due to individual preferences of the writer (Markus, 1997, p. 213).

Most normalization guidelines do not explicitly describe how they treat capitalization. Marttila (2014) only modifies capitalization for proper nouns, which are always capitalized. Belz et al. (2017) mention that they adjust capitalization according to modern orthographic rules, but do not modify it sentence-initially.[4] Vaamonde and Magro (2017) also mention the adjustment of capitalization as one of the principles for normalization, although without detailing their exact approach.[5]

2.4.2 Morphology and morphosyntax

Variation in historical language stages can also extend to morphological processes, such as inflectional patterns and/or affixes. On a morphosyntactic level, the selection of case and gender may also be subject to variation. These processes go beyond the mere level of spelling, and there are different approaches that either preserve these variations or normalize them as well.

Historical German has many inflectional patterns that differ from modern German, e.g., the feminine accusative ending *-n* (*in die Nasen* > *in die Nase*), the dative *-e* (*in dem puche* > *in dem Buch*), infinitives with *ge-* prefix (*gesein* > *sein*), and many others. They are normalized in both the RIDGES corpus (Belz et al., 2017, p. 43) and the Anselm corpus, although the latter introduces a second normalization layer (called "modernization") to keep more fine-grained information: when there is a matching surface form for the historical variant, this is recorded as the normalization, while any morphological adjustments are only made in the modernization. An example for this is the phrase *dese wort*, which is first normalized to *diese Wort*—since *Wort* is a valid Modern German word form—before being further modified to the plural form *Worte* on the separate modernization layer (Krasselt et al., 2015, p. 19 ff.).

For some of these cases, it is unclear whether they should be considered historical variants that need normalization. The dative *-e*, for example, is considered obsolete in modern German, yet still survives in many idiomatic or fixed phrases (e.g., *im Bilde sein* 'to be in the know', *am Fuße des Berges* 'at the foot of the mountain') and is intelligible to contemporary speakers. Nonetheless, both Anselm and RIDGES choose to normalize it to a form without the *-e* suffix.

[4] "Wortbildung und Großschreibung, die nicht der modernen Orthographieregeln [sic] entsprechen, werden angeglichen." ('Word formation and capitalization not conforming to modern orthographic rules are adjusted.') (Belz et al., 2017, p. 177, my translation)
"Satzanfänge bleiben kleingeschrieben, wenn sie im Original auch kleingeschrieben sind." ('Beginnings of a sentence remain lower-cased when they are lower-cased in the original as well.') (Belz et al., 2017, p. 43, my translation)

[5] "La normalización ortográfica de las formas originales, incluyendo la acentuación y la inserción de mayúsculas donde corresponda." ('Orthographic normalization of the original forms, including accents and the insertion of capitalization where appropriate.') (Vaamonde and Magro, 2017, p. 5, my translation)

Another example is *ward*, a preterite form of *werden* ('to become', also used as an auxiliary indicating passive voice), which is considered archaic or "poetic" compared to the standard form *wurde*. In RIDGES, it is preserved in the normalization, while the Anselm corpus also allows *ward* in the normalization, but changes it to *wurde* in the extra modernization layer (Krasselt et al., 2015, p. 21).

Derivational morphemes are another, related source of variation. Belz et al. (2017) describe the word form *stachelecht* for modern *stachelig* 'thorny' as using an extinct derivational morpheme *-echt*, which they normalize to the appropriate modern equivalent, here *-ig*. The adjectival/adverbial suffix *-lich* is often used in places where it is considered ungrammatical today, such as in *volkomenlich*, which would be *vollkommen* 'complete(ly)' in modern German. Krasselt et al. (2015) treat these cases like extinct word forms (cf. Sec. 2.4.3), only using the modern *vollkommen* on the second modernization layer. In the RIDGES corpus, treatment of these cases depends on whether the base lexeme can be used as a valid word form on its own: on the one hand, we can find examples like *krefftiglich* > *kräftig* 'strong' where the normalization simply drops the suffix; on the other hand, forms like *manniglich* and *gemeiniglich* are treated like extinct word forms (and normalized graphematically only, cf. Sec. 2.4.3) since **mannig* and **gemeinig* do not constitute accepted modern German words.

While the RIDGES corpus normalizes morphological variation whenever possible, it also follows the rule that "case and gender inflection are not normalized to modern German forms" (Odebrecht et al., 2016, p. 9). An example for gender variation is the Early New High German neuter noun *das Milz* 'the spleen', which is a feminine noun (*die Milz*) in Modern German. Case inflection refers to morphosyntactic adjustments within a phrase, as in *man trinke des wassers* 'one should drink the water' (Belz et al., 2017, p. 187), which uses a genitive noun phrase (*des Wassers*) where modern German syntax requires the accusative case (*das Wasser*). While RIDGES does not adjust these examples for case and/or gender at all, the Anselm corpus utilizes the extra modernization layer for this purpose.

So far, all examples in this section have been for German, which is not surprising given its rich morphology, though similar examples exist in other languages as well. Pettersson (2016) gives an Old Swedish inflected verb form *tillbragte* 'spent (time)', which uses an archaic inflectional paradigm and is therefore normalized to modern *tillbringade*. Early Modern English has the second-person singular verb ending *-st* in forms like *wouldst* or *didst*, which Archer et al. (2015) decide to normalize to *would* and *did*, respectively, although they mention that some corpora leave forms like *doth* and *hath* unchanged (p. 14).

Vaamonde and Magro (2017) describe similar cases in verb inflection of Old Spanish, but choose to only normalize them on a graphematic level:[6] examples include the second-person preterite verb form *digistes* > *dijistes*, which would actually be *dijiste* in modern Spanish, or the *-des* suffix in second-person plural forms such as *cantabades*, which is replaced by *-(e)is* in modern Spanish (*cantabais*). The corpus of Old Hungarian codices takes a similar approach, "preserving all words and morphemes, even those which do not exist in Modern Hungarian" (Simon, 2014, p. 8). As an example, they give the word form *fekette* 'laid', an inflected adverbial participle

[6] "Todas las formas marcadas como *verb_paradigm* no se modernizan a su correspondiente forma estándar. Solo se normaliza la grafía[.]" ('All the forms marked as *verb_paradigm* are not modernized to their corresponding standard form. Only the spelling is normalized.') (Vaamonde and Magro, 2017, p. 27, my translation)

which shows subject agreement, while adverbial participles in Modern Hungarian are not inflected in any way.

For research in historical language development, preserving morphological and morphosyntactic features that have fallen out of use is certainly desirable, since it enables analyses that would otherwise not be possible. From an NLP perspective, any linguistic feature that is not seen in modern language data, and therefore in the training data for most of our tools, poses an additional challenge. Particularly, normalizations that retain archaic morphemes or inflection patterns no longer satisfy the dictionary criterion (cf. Sec. 2.4.1), which can be a drawback when relying on an NLP tool that utilizes a contemporary dictionary or wordlist.

2.4.3 Lexicon and semantics

Language change can also affect the lexicon: word forms found in historical texts may have become extinct in contemporary language, or might have changed their meaning so that they would no longer be used or considered acceptable in the given context. Similar to morphological change, different degrees of handling these cases are conceivable: keeping the historical lexeme; normalizing it on the level of spelling only; replacing it with the modern word form that is most similar in meaning; or a mixture of these approaches in form of a multi-layered annotation.

All corpora and guidelines I investigated choose to keep extinct or archaic forms, offering at least one layer of normalization that retains the lexeme, but typically standardizes the spelling in some way. Archer et al. (2015) keep archaic word forms like *oft*, *morrow*, or the personal pronoun *thine*, although they do normalize different spellings of these forms to a single spelling variant, e.g., *ofte* > *oft* and *thyne* > *thine*. In a similar vein, Marttila (2014, p. 445) describes normalizing the individual components of "nonce words or spontaneous formations", citing *ontrusse* > *on-truss* (meaning 'to truss upon') as an example. Simon (2014) normalizes only the spelling of extinct word forms from Old Hungarian, such as *ýsa* > *isa* 'certainly', where Modern Hungarian would use *bizony*; Erjavec (2012) does the same for Slovene.[7] One possible challenge with this approach is that there is no contemporary dictionary that can be relied upon for the "correct" spelling of these word forms, and therefore some level of subjectivity might be involved in these decisions. Consequently, Pettersson (2016, p. 50) writes that she normalizes archaic word forms "according to intuition", citing the Swedish example *brofougde* > *brofogde*, which refers to an occupational title ('surveyor of bridges') that is no longer in use.

Extinct word forms can be handled more naturally when a *historical* dictionary is used as the point of reference. Marttila (2014) uses the *Middle English Dictionary*[8] for this purpose; Krasselt et al. (2015) use the Middle High German dictionary by Lexer (1992).[9] This ensures that all spelling variants are mapped to one common form, but leads to normalizations which neither consist of an extant lexeme nor conform to modern spelling rules; e.g., in the case of the Anselm corpus, *zehern* > *zeheren* 'tears' or *gutleichen* > *guotlichen* 'friendly'. Here, the additional modernization layer is used instead to provide a modern translation of these words (here, *Tränen* and *freundlich*). Importantly, though, choosing this route means there is no layer

[7] "Where the word does not exist anymore in the contemporary language only its spelling is modernised." (Erjavec, 2012, p. 2259)

[8] https://quod.lib.umich.edu/m/med/

[9] http://www.woerterbuchnetz.de/lexer

that keeps the historical lexeme while modernizing the spelling, which is the most commonly found approach in other corpora.

Apart from the Anselm corpus, many other corpora also provide a second level of annotation that provides glosses or translations. For German, the RIDGES corpus offers an "explanation" column for extinct word forms, but otherwise only normalizes them graphematically as in *Vergefz* > *Vergess* (Belz et al., 2017, p. 43). The same approach is taken by Erjavec (2015) for Slovene. For Spanish and Portuguese, Vaamonde and Magro (2017) also provide an additional field for normalized spellings of "variant lexemes", but do *not* extend this to archaic or extinct lexemes. In particular, they require that variant word forms marked in this way must stem from the same lemma as its standard form.[10]

Since most corpora only normalize archaic word forms for spelling, they do not modify words that still exist in the contemporary language, but have undergone a significant semantic shift. When a gloss or translation is provided, this can be extended to instances of semantic shift as well—Erjavec (2015) explicitly mentions doing this. Krasselt et al. (2015) list many examples for changes in meaning, such as *wib* > *Weib* 'woman', which is considered archaic or even derogatory today, but historically used in a neutral fashion. It is therefore given the additional modernization *Frau* as its closest neutral equivalent in Modern German.

As with morphological changes, there are many arguments in favor of preserving archaic or extinct word forms: not only does it enable studies on lexical change, but it also avoids losing nuances of meaning, as there might not always be a perfectly equivalent modern translation. However, this category arguably poses the greatest challenge for NLP applications, mostly because they cannot draw on any modern resources (like corpora or dictionaries) when processing them. From the perspective of automatic normalization, for example, extinct lexemes with normalized spelling cannot be matched or verified against an existing modern resource. Using the most appropriate extant equivalent as the target normalization, on the other hand, changes the nature of the normalization task considerably: contrary to most types of variation, this is a word-level operation that can no longer be reasonably broken down to character-level transformations.

2.4.4 Syntax and punctuation

Syntactical variation is not typically considered for normalization. Since one common goal of normalization is to provide a basis for further automatic processing of the text, often involving further annotations which should be projected onto the source tokens, it is desirable to restrict the normalization to a purely token-level annotation of the source text. Instead, syntactical changes—particularly involving word ordering—can be seen as the boundary between normalization and a full translation.

Punctuation is a similar case. On the one hand, it also shows a high degree of variance: some historical texts do not contain punctuation marks at all, or only very sparsely; and if they do, their usage can differ significantly from modern conventions. However, the modern usage

[10] "La forma no estándar y la forma estándar deben compartir un mismo étimo y un mismo lema." ('The standard and non-standard forms must share the same source word and lemma.') (Vaamonde and Magro, 2017, p. 20, my translation)

of punctuation is often based on syntactical considerations, and correctly placing, changing, or removing a punctuation mark is dependent on a potentially large context window. This is contrary to the normalization of alphabetic tokens, which is mostly driven by the surface form of the token itself.

If a punctuation mark should be inserted according to modern rules where none is present in the historical text, it is not obvious how this annotation should be recorded, since there is no source element to annotate. Vaamonde (2017) includes the adjustment of punctuation in the normalization step, and in the case of insertions, inserts an empty element into the source text which is then "normalized" with the respective modern punctuation mark. On the other hand, if only existing punctuation marks are normalized, the usefulness of punctuation within the normalized text is limited.

2.5 Conclusion

"Normalization" in the context of historical texts can be defined in many different ways, and the most desirable definition always depends on its intended purpose. Typically, they involve a trade-off between preserving features of the source material that are deemed interesting or relevant for future research, and improving the usability and/or accuracy of NLP tools on the data. In some cases, corpora contain multiple layers of normalization that enable the user to choose the one they consider most appropriate for their use case. Importantly, variation in historical language can go beyond the mere surface form of a word, and extend to areas such as morphology, semantics, and syntax.

In the context of this thesis, I am mostly interested in normalization both as an aid for users of a corpus and as a way to enable or improve the application of downstream NLP applications, such as part-of-speech taggers or parsers. This is best achieved by normalizing to the modern standard variety of the language in question, which is also the most common approach, and the most common usage of the term *normalization* (cf. Sec. 2.3).

To this end, most guidelines and/or corpora characterize the task of normalization as the token-level mapping of historical variant word forms to a single, unified word form, which should be graphematically close to the original form and covered by a contemporary dictionary whenever possible. Exceptions are made mainly for tokens containing archaisms, i.e., extinct lexemes or morphemes, or archaic inflection patterns. In these cases, most corpora choose to preserve the archaic feature and normalize the spelling only, even though the resulting word form will not be covered by a modern dictionary. There is no consensus on proper nouns; they are sometimes normalized, sometimes not. Morphosyntactic, syntactic, and semantic variation is usually *not* considered; this distinguishes normalization from a full translation task.

I do not consider capitalization as part of the normalization task, but rather treat it as a separate problem. To that effect, I will consider all data to be case-insensitive—in practice, that means lowercasing all word forms. First of all, the nature of capitalization is different from other aspects of writing, as it is an additional feature of a character that does not change the intrinsic value it represents. Indeed, it is not clear how much ambiguity is introduced by discarding normalization. In German, common nouns are capitalized as a rule, and while it is possible to find many example phrases in German that become ambiguous without proper

capitalization,[11] these tend to be rather artificial and rare (Müller, 2016, p. 53 f.), leading Hoberg and Hoberg (1975) to conclude that capitalization is not a necessary feature of the written language.[12] Secondly, when capitalization is used for proper nouns, restoring it essentially amounts to performing named entity recognition; sentence-initial capitalization, on the other hand, requires syntactic analysis to find sentence boundaries, but no other part of normalization includes syntactic considerations. Lastly, it is questionable whether correct capitalization can be successfully learned from spelling normalization data, since (i) the model has to learn an additional feature—capitalization—on top of the character transformations themselves, and (ii) uppercase characters occur much less frequently than lowercase characters, amplifying the already challenging problem of training data sparsity. Restoring correct capitalization, often called *truecasing*, is also known as a separate problem in the context of machine translation, and algorithms proposed to perform this task (e.g., Vlad Lita et al., 2003; Wang et al., 2006; Susanto et al., 2016) can conceivably also be used on spelling normalization output.

For similar reasons, I also exclude punctuation marks from my definition of normalization. Their placement is also syntactic in nature, making it impracticable to represent the task of modernizing them by means of character-level transformations. This is particularly true when the normalization introduces modern punctuation marks that have no equivalent in the historical text. Even when such cases are disallowed, including punctuation marks potentially distorts the evaluation: when a historical text already uses them similarly to modern conventions, their normalization is usually trivial, leading to easy gains in normalization accuracy that other texts with fewer or no punctuation marks do not get.

[11] A common example is *der gefangene Floh* 'the captured flea' vs. *der Gefangene floh* 'the prisoner fled'.

[12] "Weder die Struktur der deutschen Sprache an sich noch die – äußerst geringe – Zahl der bei Kleinschreibung auftretenden Doppeldeutigkeiten machen die Großschreibung erforderlich." ('Neither the structure of the German language itself nor the – extremely low – number of ambiguities occuring with lower-case writing necessitate the use of capitalization.') (Hoberg and Hoberg, 1975, p. 167, my translation)

Chapter 3

Corpora

> *Computerized corpora can be said to have revolutionized the study of the history of English. [... They] give us an opportunity to master huge quantities of textual material, to collect and sort evidence with a speed and level of accuracy that the scholars of earlier decades could only have dreamt of.*
>
> — Rissanen (2000)

All systems for automatic normalization discussed and evaluated in this thesis are trained and evaluated on manually prepared gold-standard datasets. This chapter presents all datasets that are used in the later experiments and discusses their properties.

Sec. 3.1 introduces the historical datasets for the experiments, the types of texts they contain, and further references in case there is previous work on these datasets. Sec. 3.2 summarizes preprocessing decisions that were applied to all datasets, while Sec. 3.3 introduces a method to generate character-aligned versions of the datasets that are used for some analyses. Sec. 3.4 explores ways to analyze and measure certain properties of the datasets, such as the ambiguity of mapping historical to normalized tokens and the similarity between datasets. Finally, Sec. 3.5 also introduces contemporary language resources that are used in some experimental setups to aid the normalization process.

3.1 Historical datasets

For supervised learning and evaluation of normalization, we rely on historical text collections that provide gold-standard normalizations on a token level. "Gold-standard" means that normalizations must have been created manually, or at least manually checked and corrected after an automatic normalization step. This serves to ensure that the normalization is somewhat reliable and our models do not just learn to reproduce another machine learning model's output; although it is not always clear how consistent these annotations actually are, as inter-annotator agreement is rarely performed (but cf. Bollmann et al., 2016). Furthermore, as Chapter 2 discussed in detail, the exact guidelines used to produce the normalization can and do differ between projects. "Token-level" means that normalizations should be aligned to the historical word forms or provided as annotations to them; datasets that simply provide two separate versions of a text (historical and normalized) are not considered, as the models used in my experiments operate on a token level, and adding an automatic word alignment step is potentially error-prone.

Dataset	Language	Corpus	Corpus Reference
DE_A	German	Anselm	Wegera (2014)
DE_R	German	RIDGES	Odebrecht et al. (2016)
EN	English	ICAMET	Markus (1999)
ES	Spanish	Post Scriptum	Vaamonde (2017)
HU	Hungarian	HGDS	Simon (2014)
IS	Icelandic	IcePaHC	Rögnvaldsson et al. (2012)
PT	Portuguese	Post Scriptum	Vaamonde (2017)
SL_B	Slovene (Bohorič)	goo300k	Erjavec (2012)
SL_G	Slovene (Gaj)	goo300k	Erjavec (2012)
SV	Swedish	Gender and Work	Fiebranz et al. (2011)

Dataset	Time Period	Genre	Tokens			Dataset Reference
			Train	Dev	Test	
DE_A	14^{th}–16^{th} c.	Religion	233,947	45,996	45,999	Bollmann et al. (2017)
DE_R	1482–1652	Science	41,857	9,712	9,587	–
EN	1386–1698	Letters	147,826	16,334	17,644	Pettersson (2016)
ES	15^{th}–19^{th} c.	Letters	97,320	11,650	12,479	–
HU	1440–1541	Religion	134,028	16,707	16,779	Pettersson (2016)
IS	15^{th} c.	Religion	49,633	6,109	6,037	Pettersson (2016)
PT	15^{th}–19^{th} c.	Letters	222,525	26,749	27,078	–
SL_B	1750–1840s	Mixed	50,023	5,841	5,969	Ljubešić et al. (2016b)
SL_G	1840s–1899	Mixed	161,211	20,878	21,493	Ljubešić et al. (2016b)
SV	1527–1812	Mixed	24,458	2,245	29,184	Pettersson (2016)

Table 3.1: Overview of historical datasets, giving some details about the source corpus and texts as well as an abbreviation used throughout this work ("Dataset"), a reference that describes the corpus in more detail ("Corpus Reference"), and a reference from which the dataset and splits were taken, if applicable ("Dataset Reference"); number of tokens refers to final dataset splits after preprocessing (cf. Sec. 3.2).

Previous work on historical normalization is often concerned with performance on one particular target language, although comparisons exist between different domains and language stages (e.g. Ljubešić et al., 2016b) or between different writers and dialects (e.g. Bollmann et al., 2011a; Bollmann and Søgaard, 2016). In this work, the focus will be on comparisons across different languages; to this effect, I will use datasets from eight different languages, and mostly train and evaluate on a single dataset per language (with few exceptions). While cross-language comparisons of automatic normalization methods have been done before (e.g. Etxeberria et al., 2016; Pettersson et al., 2014a; Pettersson, 2016), this is—to the best of my knowledge—the most extensive evaluation performed thus far.

The languages used in my experiments are English, German, Hungarian, Icelandic, Portuguese, Slovene, Spanish, and Swedish. Whenever possible, I have re-used available datasets (including their splits into training, development, and test sets) from previous work, both for better comparability and easier reproduction of results. The English, Hungarian, Icelandic, and Swedish datasets are taken from Pettersson (2016).[1] The German Anselm dataset is based on the data used in Bollmann et al. (2017). The Slovene dataset is taken from Ljubešić et al. (2016b). The Spanish and Portuguese datasets are described in Vaamonde (2017) but have, to the best of my knowledge, not been used in normalization experiments so far. The same applies to the German RIDGES dataset (Odebrecht et al., 2016).

Table 3.1 gives an overview of all historical datasets and the size of their training/development/test sets after preprocessing (cf. Sec. 3.2). The following extracts show sample passages from each dataset along with the annotated normalization:

(DE_A) deſe wort ſpricht vnſer liber here iheſus criſtus czu eyme iczlychen menſchen
diese wort spricht unser lieber herr jesus christus zu einem ieteslichen menschen

(DE_R) ſeind ſÿ doch alle auſz den vier elementen gemiſchet vnd eins feüchter deñ das ander
sind sie doch alle aus den vier elementen gemischt und eins feuchter denn das andere

(EN) whan your graciouse erthely persoune from your inward spirit ys dessolued
when your gracious earthly person from your inward spirit is dissolved

(ES) anque tomeys mui mucho travajo tengola guardada pa quando dios sea servido
aunque toméis muy mucho trabajo téngola guardada para cuando dios sea servido

(HU) o̩ zauoc èſmėǵ felèmèluē kèzdėnç̣ ſirńoc èlmēnèc èzèkèt tolga ez a noemi azeɔt iǫuǫ
ő szavuk ismét felemelvén kezdének sírniuk elmenjek ezeket toldja ez a noémi azért jöve

(IS) þá sem hanz gödverk voru i og þá vrdu hanns gödverk miklu þýngre enn ill
þá sem hans góðverk voru í og þá urðu hans góðverk miklu þyngri en ill

(PT) cõ a poenetencia que lhe derão pera avisar aos snres do sancto oficio
com a penitência que lhe deram para avisar aos senhores do santo ofício

(SL_B) ter ne bodi nevéren zhe ſe zherna perſt premozhi tezhe od nje rjav mòk
ter ne bodi neveren če se črna prst premoči teče od nje rjav mok

(SL_G) in privéže na vsak konec niti drobtino kruha in verže vse kokóšem breskevno vkuhanje lovre
in priveže na vsak konec niti drobtino kruha in vrže vse kokošim breskvino vkuhanje lovre

(SV) blef av rätten afsagdt det en syyn och rådhgångh nu nästkommande wårdagh hållas
blev av rätten avsagt det en syn och rådgång nu nästkommande vårdag hållas

[1] Many thanks to Eva Pettersson for kindly providing me with these datasets.

The following sections describe each dataset in more detail.

3.1.1 English

The historical English data comes from the *Letter Corpus of ICAMET*, the *Innsbruck Computer Archive of Machine-Readable English Texts* (Markus, 1999), also referred to as the *Innsbruck Letter Corpus*.[2] It consists of 469 complete letters written between 1386 and 1698, totaling about 182,000 words.

Markus (1997, 2000) discusses some aspects of normalizing texts from the Prose Corpus of ICAMET, but little information can be found that pertains to the manual normalizations in the Letter Corpus. Some editorial decisions have already been applied to the source texts, e.g., replacing ⟨*ſ*⟩ with ⟨*s*⟩, although other historical characters such as the letter thorn ⟨*þ*⟩ are retained.

However, inspecting the data reveals some of the properties of the manual normalization. For example, we can see that extinct word forms are replaced with modern ones:

(1) And I lete hym wete he that putte it downe chull pay therefore
And I let him know he that put it down shall pay therefore

(2) for þe good of pees betwix boþe reaumes
for the good of peace between both realms

In Ex. (1), the word form *wete* (from Middle English *witan*, cognate with German *wissen* 'to know') has been replaced with its modern semantic equivalent *know*. Lexemes that are archaic, but not necessarily extinct, can also be replaced, such as *betwix* in Ex. (2), which is normalized as *between* even though *betwixt* arguably still exists in present-day English.[3]

The normalization also takes inflection and context into account, as Ex. (3) shows:

(3) and yf enythyng be theryn to myche or to litell
and if anything is therein too much or too little

Here, *be* could conceivably be left unmodified in the normalization, but is normalized as the inflected form *is* instead. The normalization *to* > *too* can only be derived from the context, as *to* is a valid modern word, but the context makes it clear that this is not the intended word in this case.

The training, development, and test splits for this dataset are taken from Pettersson (2016).

[2] https://www.uibk.ac.at/anglistik/projects/icamet/

[3] It is attested in various dictionaries such as the Cambridge Dictionary or Merriam-Webster; see, e.g.:
https://dictionary.cambridge.org/dictionary/english/betwixt
https://www.merriam-webster.com/dictionary/betwixt

3.1.2 German

For the German experiments, I use two different datasets derived from two different corpora: the Anselm corpus and the RIDGES Herbology corpus. While both contain texts in Early New High German from a similar time period, they differ significantly in at least two aspects: (i) *text genre,* as the Anselm corpus contains religious texts while RIDGES is a corpus of scientific texts; and (ii) *normalization guidelines,* as the two corpora differ in how they treat inflectional variation, archaisms, and tokenization.

Anselm Corpus

The corpus *St. Anselmi Fragen an Maria* ("Questions of Saint Anselm to [the Virgin] Mary"), or just *Anselm Corpus* for short, is a collection of several written records of a medieval treatise (Schultz-Balluff and Dipper, 2013a; Wegera, 2014).[4] The final corpus will consist of up to 69 versions in Early New High German, written between the 14th and 16th centuries in various dialectal regions.

At the time of my experiments, the corpus is not yet officially released. Therefore, I use preliminary versions of a subset of the corpus, consisting of 46 texts exported on June 14, 2017, with a total size of about 326,000 tokens.[5] This selection is mostly identical to that used in previous experiments which trained and evaluated on texts individually (Bollmann and Søgaard, 2016; Bollmann et al., 2017); in those experiments, the first 1,000 tokens of each text were used as the test set, the next 1,000 tokens as the development set, and the remainder as the training set. This split was chosen over a randomized sample due to the semi-parallel nature of the texts, causing them to have a substantial overlap in vocabulary and structure. By always taking the test/dev splits from the beginning of each text, the training sets will never contain those parts of the texts and some of the vocabulary that is specific to them, hopefully resulting in a slightly less biased evaluation.

For the experiments in this work, I concatenate all splits from the individual texts into a single dataset. However, since the test sets have already been evaluated on in previous work, but the development sets have not been utilized so far, the roles of those sets are now switched. Therefore, my development set consists of the first 1,000 tokens from each text (previously used as test sets in Bollmann and Søgaard (2016) and Bollmann et al. (2017)), while the test set consists of the following 1,000 tokens from each text (the previously unused development sets). The training set is again built from the remaining parts of the texts.

An important feature of the corpus is the distinction between two tokenization layers, *diplomatic*—i.e., as found in the original source text—and *modernized.* These two layers differ whenever word boundaries in the historical text do not align with expected boundaries in contemporary German, no matter if this discrepancy is due to idiosyncracies of the document or due to linguistic change. This means that the modernized tokenization already resolves some of the issues that normalization might otherwise have to solve: for example, the historical

[4] https://www.linguistics.rub.de/anselm/

[5] The texts used are *b, B, B2, B3, Ba, Ba2, Be, D3, D4, H, Hk, Ka, KÄ1492, KJ1499, Le, M, M2, M3, M4, M5, M6, M7, M8, M9, M10, Me, n, N, N2, N3, N4, N1500, N1509, N1514, s1495, s1496/97, Sa, Sa2, Sb, SG, St, St2, Stu, T, W,* and *We.*

foltu 'should you' needs to be normalized as two words, *sollst du*, in modern German, but the modernized tokenization layer already splits the source word into two parts:

(4) ſolt u
sollst du
should you

This arguably simplifies the normalization task a bit. However, since all annotations—including the normalizations—are attached to tokens in the modernized tokenization, this is the layer that will be used here.

Krasselt et al. (2015) describe the normalization guidelines that were used to produce the gold-standard normalizations. In addition to the normalization, they introduce a *modernization* layer that offers a more radical adjustment of the source word in the following three cases: (i) inflectional changes (i.e., the closest normalization results in a word form that would be inflected differently in contemporary German); (ii) semantic shift (i.e., the word is used differently today and would not be appropriate in the given context); and (iii) extinct word forms. In those cases, the normalization provides the closest modern equivalent on a graphematic level, while the modernization also adjusts the inflection or uses a completely different lexeme. Since most corpora and guidelines take a more conservative approach and err on the side of staying closer to the historical word form (cf. Secs. 2.4.2 and 2.4.3), I do not use the modernization layer in any of these cases, and use only the normalization as the gold-standard data.

The Anselm corpus also provides different layers of character representations, notably one "UTF" layer with Unicode representations of the historical word forms, and one "simplified" layer. The latter only consists of characters in the modern German alphabet, simplifying the original tokens by mapping, e.g., *ſ* > *s*, but also implementing heuristics to map a nasal bar to one of *(e)n/(e)m*. Since this arguably constitutes a form of pre-normalization, I choose to only use the original representation. Furthermore, since longer passages of foreign-language material (e.g., Latin passages) are explicitly marked up in the corpus, I filter them before constructing the splits.

To illustrate the nature of the corpus and its normalization, consider the following examples, which are taken from the development sets of different texts (*B2, Hk, SG*, and *W*, respectively):

(5) a. sante anſhelm der bad vnſer liebe frauwe von hymelriche alczü lange zijt
sankt anselm der bat unser liebe frau von himmelreich allzu lange zeit
b. sant anſhelmus der pat vnſer frawen von hymelreich lange zeit
sankt anselm der bat unser frauen von himmelreich lange zeit
c. sant anſelm batt vnſer liebē frowen lang zit
sankt anselm bat unser lieben frauen lang zeit
d. sannd anſhalm pat unſer frawn von himlreich lange czeit
sankt anselm bat unser frauen von himmelreich lange zeit

Saint Anselm asked our (dear) lady (from heaven) for a long time

It is not difficult to find such examples of semi-parallel text passages across different texts in the corpus. However, they also demonstrate the high amount of variation among them: *Frau* 'woman/lady' is spelled as *frauwe*, *frawen*, *frowen*, or *frawn*; in total, there are 80 different variants of *Frau(en)* in the dataset, with 22 of them occuring at least 10 times.

The examples also show the difficulties of normalizing a highly inflected language such as German: the modern equivalent of the accusative noun phrase 'our dear lady' would be *unsere liebe Frau*; however, the historical sources in Ex. (5) all use *vnſer*/*unſer* without the final *–e*, and *frawn*/*frawen*/*frowen* suggests the modern plural form *Frauen*. Since Krasselt et al. (2015) require the normalization to be based purely on graphematic and phonological adjustments, they are *unser Frauen* in these cases, while the expected *unsere Frau* is only annotated on the modernization layer.

RIDGES Herbology Corpus

The project *Register in Diachronic German Science (RIDGES)*[6] is concerned with the analysis of scientific language from the mid-15th to the 20th century. To this end, the *RIDGES Herbology Corpus* contains a collection of "herbal treatises, lectures, and scientific texts" (Odebrecht et al., 2016); in version 6.0, it contains 50 excerpts from the time period 1482–1914 (Lüdeling et al., 2017).

The dataset used here is made up of 16 texts from 1482 to 1652; newer texts were deliberately excluded as they tend to show significantly less variation. From each text, 70% of all sentences are randomly sampled to be included in the training set, while another 15% each are used for the development and test sets.[7]

Details about the normalization process are described in Odebrecht et al. (2016, p. 9 ff.); further information can be found in the official corpus documentation (Belz et al., 2017). Their principles differ in subtle ways from those of the Anselm corpus. While the latter normalizes extinct lexemes to a standard historical form provided by a dictionary, the RIDGES corpus normalizes them to modern German orthography without changing the lexeme (e.g., *vinſtere* > *Finstere*, instead of the modern *Finsternis*). However, this does not apply to extinct morphemes, which are replaced with modern equivalents if possible; or removed, as in the case of the obsolete adverbial suffix *-lich*, e.g., *machtigklich* > *mächtig*. The same example is treated like an extinct word form in the Anselm corpus (Krasselt et al., 2015) and given the normalization *mehticlich*, a Middle High German dictionary form given by Lexer (1992).

Furthermore, while there is no morphosyntactical normalization and no adjustment of gender or grammatical case (Belz et al., 2017, p. 43), the RIDGES guidelines are more flexible with regard to smaller morphological adjustments. From the 1487 text *Gart der Gesundheit*:

(6) wie verzeren wir vnſer blůmen vnd vnſer krafft
wie verzehren wir unsere blumen und unsere kraft
how we consume our flowers and our strength

[6] https://korpling.org/ridges/
[7] Many thanks to Uwe Springmann, Bryan Jurish, and Martin Klotz for preparing the texts and dataset splits.

Here, the historical *vnfer* is normalized to the correct inflected form *unsere*, while the excerpts from Ex. (5) showed that the Anselm corpus normalizes to the orthographically closest form *unser*, reserving the correctly inflected form for a separate layer.

Lastly, tokenization is also handled slightly differently in the two datasets. When a historical word form corresponds to more than one normalized word form, as in *foltu* > *sollst du*, the historical word form is not split up in the dataset as in the Anselm corpus (cf. Ex. (4)). Moreover, when two historical words need to be joined in the normalization, they will be combined into one historical token that contains a space character, while the Anselm dataset never contains spaces within tokens.

3.1.3 Hungarian

The Hungarian dataset is a manually normalized subset from the project *Hungarian Generative Diachronic Syntax (HGDS)*[8] (Simon, 2014), containing eleven Old Hungarian codices from 1440 to 1541; this subset (including the splits) is taken from Pettersson (2016) and contains about 167,500 tokens in total.

Information about the digitization and normalization process can be found both on the project website[9] and in Simon (2014). Normalization was guided by two main principles (Simon, 2014, p. 8):

1. "Adherence to the original text", meaning that all lexemes and morphemes are preserved, even when they have no equivalent in Modern Hungarian.
2. "Consistency", stating that all variant word forms should be mapped to a single normalized form that corresponds to Modern Hungarian spelling rules.

Tokenization was also performed manually during normalization, and words could be joined or split up if the original word boundaries did not conform to modern spelling rules (Simon, 2014, p. 7).

Hungarian spelling in the 14th–16th century was highly inconsistent, particularly due to the challenges of adapting the Latin alphabet to a language with a distinct inventory of phonemes (Oravecz et al., 2010). The need to represent phonemes that did not exist in Latin makes Hungarian spelling—both historical and modern—very rich in diacritics, as the following examples from the dataset may illustrate:

(7) a. gèɔmėkimnc̣
gyermekeimnek
my children

b. iſtènǭc
istenünk
our God

[8] http://omagyarkorpusz.nytud.hu/en-intro.html
[9] http://omagyarkorpusz.nytud.hu/en-descr.html#norm

c. ǫvèlėc
ővelük
with them

d. tv̨uèlètėc
tiveletek
with you

3.1.4 Icelandic

The Icelandic dataset originates from the *Icelandic Parsed Historical Corpus (IcePaHC)*,[10] a diachronic, parsed corpus of Icelandic texts from the late 12th century to the present (Rögnvaldsson et al., 2012). The subset used for the experiments is taken from Pettersson (2016) and contains four manually normalized texts from the 15th century, "three sagas [...] and one narrative-religious text" (Pettersson, 2016, p. 80), adding up to about 63,000 tokens.

Example (8) shows an excerpt from the dataset, illustrating that the normalization can involve the addition of diacritics (such as *a* > *á*) or the introduction of the letter 'eth' (*d* > *ð*).

(8) þeir badu sina modr þeinkia upp áá sitt fyrra lif og bidia uorn herra miskunnar
þeir báðu sína móður þenkja upp á sitt fyrra líf og biðja vorn herra miskunnar
they asked their mother to think about her former life and ask for our lord's mercy

There is no detailed description of the normalization process for these texts, but Rögnvaldsson et al. (2012) suggest that deciding on a normalized word form usually does not pose many challenges:

> There was no accepted spelling standard until the 20th century[...]. However, since the morphology is the same, it is usually relatively straightforward to convert older spelling to the modern standard and get legible text.
>
> (Rögnvaldsson et al., 2012, p. 1978)

This does not mean, of course, that historical Icelandic spelling is not rich in variation; particularly for diplomatic editions of texts, spelling "is often highly irregular" (Rögnvaldsson et al., 2012, p. 1979). For example, the modern Icelandic *væri* can be found in the historical dataset as *uæri*, *uéri*, *være*, *véri*, or *uęri*.

3.1.5 Slovene

The Slovene data comes from *goo300k*,[11] a reference corpus of historical Slovene containing books from various genres (including plays, fiction, and religious books) as well as selected newspaper issues (Erjavec, 2012). This corpus has been previously used in a number of experiments on automatic normalization (Scherrer and Erjavec, 2016; Etxeberria et al., 2016; Ljubešić et al., 2016b); the data and the training, development, and test splits used here are taken from

[10] http://www.linguist.is/icelandic_treebank/Icelandic_Parsed_Historical_Corpus_(IcePaHC)
[11] http://nl.ijs.si/imp/index-en.html

Ljubešić et al. (2016a). In this resource, the corpus has been split up into two parts (cf. Ljubešić et al., 2016b):

1. *Bohorič*, containing texts published after 1750 that were written in the Bohorič alphabet.
2. *Gaj*, containing texts written before 1900 in the Gaj alphabet, which became the dominant alphabet for writing Slovene around 1843 (cf. Erjavec, 2015, p. 755 f.).

This split was not only motivated by the different alphabets, but also by further standardization processes of the Slovene language:

> The introduction of the Gaj alphabet was also closely preceded by a new grammar and subsequent standardisation of the language, therefore the change in the alphabet makes a convenient split between very non-standard and slightly non-standard historical language. (Ljubešić et al., 2016b, p. 147)

For these reasons, and for comparability with previous work, I choose to adopt the same dataset split in my experiments. Exaxmple (9) gives a sample of the Bohorič part of the corpus, while Example (10) is taken from the Gaj part:

(9) kadar je pak enkrat sraſlu je vęzhi kakòr vſe sęliſha inu poſtane enu drèvú
kadar je pa enkrat zraslo je večje kakor vsa zelišča in postane eno drevo
once grown it is bigger than all herbs and becomes a tree

(10) je še mnogo napčnih misel drevésa blizo polja so poljskim pridelkam škodljive
je še mnogo napačnih misli drevesa blizu polja so poljskim pridelkom škodljive
there are still many wrong thoughts [that] trees near fields are harmful to the field crops

Note particularly the frequent use of ⟨ſ⟩ in the Bohorič part (but not in the Gaj part) and the generally higher frequency of spelling variants that are changed in the normalization.

Erjavec (2015, p. 765 f.) describes the word-level normalization process in more detail (albeit referring to it as "modernisation"). The main principle is "giving [tokens] the inflected word form in contemporary orthography", while any archaic elements resulting from inflectional, semantic, or lexical differences are retained; an additional, separate annotation is used to provide a gloss or a modern translation in these cases.

3.1.6 Spanish and Portuguese

The datasets for Spanish and Portuguese are both derived from the corpus of the *Post Scriptum (P.S.)* project,[12] which contains a broad collection of unpublished, private letters from Spain and Portugal written between the 16th and 19th centuries (CLUL, 2014). A subset of this corpus—about 120,000 tokens for Spanish and 276,000 tokens for Portuguese—was normalized manually.[13]

[12] http://ps.clul.ul.pt/

[13] Many thanks to Rita Marquilhas for providing me with the manually normalized parts of this corpus.

To generate the dataset splits, each text is first categorized by century according to the metadata provided by the corpus files. From each century, 80% of all sentences are randomly selected for the training set, another 10% for the development set, and the final 10% for the test set of their respective language. This procedure is supposed to somewhat balance the large time period covered by the corpus.

Vaamonde (2017) provides a detailed technical description of the corpus, including a discussion of transcription and editing conventions. Detailed annotation guidelines, including those for the normalization, are laid out by Vaamonde and Magro (2017). They follow the principle of purely orthographic normalization:

> Las modificaciones realizadas sobre el texto se ciñen únicamente al nivel ortográfico, por lo que no se eliminan ni añaden palabras respecto del contenido original de la carta. Tampoco se interviene sobre el nivel léxico: se conservan los regionalismos y los arcaísmos léxicos, así como cualquier otra forma léxica no estándar, si bien estos casos son tratados en un nivel independiente para facilitar su recuperación[.][14]
>
> (Vaamonde and Magro, 2017, p. 5)

This means that archaisms are preserved whenever possible, both on a morphological and lexical level (cf. the discussions on pp. 22 and 24). Different levels of adjustments can be annotated in the data (cf. Vaamonde, 2017, p. 84):

- *fform:* expanded form of abbreviations, or free form of contractions;
- *dform:* a dialectal or non-standard form; and
- *nform:* the normalized form.

Vaamonde and Magro (2017, p. 19) give the example *hagora*, which receives the normalization (nform) *ahora* and the non-standard form (dform) *agora*. Abbreviations and contractions are not assigned a normalized form, but are resolved on the separate 'fform' layer; e.g., the historical token *q* is expanded there as *que*.

For the datasets used in my experiments, the target normalization is considered to be the 'nform' whenever it is given. When this normalized form is not available, an expanded form given by 'fform' is used if possible. If neither form is annotated, the original word form is considered to be the normalization. The 'dform' is never used here.

Example (11) is taken from the Spanish part, Example (12) from the Portuguese part of the corpus:

(11) y qndo no hallare q ebyarme pa çenar ebye huebos q aqllo çenare
y cuando no hallare qué enviarme para cenar envíe huevos que aquello cenaré
and when one cannot find what to send me for dinner, send eggs, that [is what] I will eat

[14] "The modifications made to the text are limited solely to the orthographic level, so that words are neither deleted nor added with respect to the original content of the letter. There is also no intervention on the lexical level: archaic vocabulary and regionalisms are conserved like any other non-standard lexical form, although these cases are handled on a separate level to facilitate their retrieval.' (my translation)

(12) he vos ão de fazer perguntas se são resebido cõvosquo e aveis de dizer q não
e vos hão de fazer perguntas se sou recebido convosco e haveis de dizer que não
and [they] shall ask you questions on whether I was with you[15] *and you shall say no*

Note that for Spanish, the historical text often lacks accents where the normalization has them. Also, abbreviations such as *q* > *que*, *vra* > *vuestra*, or *snres* > *senhores* (cf. p. 29) are relatively frequent throughout both datasets.

3.1.7 Swedish

The evaluation of Swedish uses data from the *Gender and Work (GaW)* corpus,[16] containing historical documents about the occupations of men and women before the 19th century (Fiebranz et al., 2011). The normalized dataset is from Pettersson (2016) and consists of 1,200 randomly sampled sentences (about 56,000 tokens) from "11 court records texts and 4 church documents from the time period 1527–1812" (p. 49).

Example (13) shows a short excerpt:

(13) när det skedt ähr will rätten lagligen i detta ährendet sluta
när det skett är vill rätten lagligen i detta ärendet sluta
when this has happened, the court will legally decide in this matter

The principles for normalization are described in Pettersson (2016, p. 49 ff.); the main criterion is that normalized word forms should be "likely to be present in a modern language dictionary" (p. 50; cf. also Sec. 2.4.1). Archaic lexemes constitute an exception and are only normalized in spelling, though morphological adjustments are sometimes performed: Pettersson (2016, p. 51) gives the example *närvarellse* 'presence', which could be orthographically normalized to *närvarelse*, but is instead changed to the appropriate modern equivalent *närvaro*.

Pettersson et al. (2012, p. 335) give some examples for regular spelling changings from 17th century Swedish; the chosen examples are all found in the Swedish dataset used here as well:

- Substitution of letters to a phonologically similar variant, e.g., *qvarn* > *kvarn* 'mill'
- Deletion of repeated vowels, e.g., *saak* > *sak* 'thing'
- Deletion of mute letters, e.g., *dömbdess* > *dömdes* 'was sentenced'
- Changing of spelling influenced by other languages such as German, e.g., *schall* > *skall* 'shall'

[15] lit. 'I was received with you'
[16] http://gaw.hist.uu.se/?languageId=1

3.2 Preprocessing

This section describes the preprocessing steps that were applied to all datasets, both historical and modern ones (when applicable). All examples, statistics, and experiments will be based on the preprocessed versions of the datasets. In particular, when statistics for datasets that have been used in previously publications differ from those presented here, this is likely to be due to the preprocessing decisions implemented here.

The input data for this step is already tokenized, and in the case of historical datasets, additionally provides one target normalization for each token. Tokenization is typically supplied by the datasets (cf. Secs. 3.1 and 3.5 for details). Preprocessing then consists of the following steps: (i) removing punctuation; (ii) lowercasing all characters; (iii) substituting digits; and (iv) performing Unicode normalization.

First, instances where either the historical token or its normalization consist *only* of punctuation characters are removed.[17] Whenever a string contains punctuation, but not exclusively, it is neither removed nor altered in any way. This affects all tokens where punctuation characters have not been split off in the tokenized data, e.g. for abbreviations. After that, all characters are converted to lowercase.[18] Rationales for both of these decisions have been discussed in Sec. 2.5.

In the historical datasets, whenever a token contains digits and the normalization is identical to the source token, all digits are replaced with zeroes. Typically, digits appear only infrequently in texts, but in a high (and potentially limitless) number of combinations. At the same time, they are usually not affected by normalization. This preprocessing step makes learning easier for the automatic normalization models in the trivial cases where digits just need to be copied over. When the normalization modifies a source token containing digits, they are not changed; e.g., this happens when a digit is used in place of a morpheme, as in the Spanish *8bre* > *octubre* 'October', or when it is used in the normalization of Roman numerals (*vii* > *7*).

Finally, Unicode normalization is performed on all characters. While most datasets already use Unicode, this does not guarantee that the same characters are always represented in an identical way. A common example are letters with diacritics: for example, the character ⟨ÿ⟩ has its own Unicode codepoint (`U+00FF`), but can also be represented as a combination of plain ⟨*y*⟩ and a combining diaeresis (`U+0079` and `U+0308`).

The Unicode standard defines normalized forms of Unicode strings, which allows conversion of strings to always use either the composed or decomposed forms of such characters (Davis and Whistler, 2017). However, it is not clear which representation is preferable in the context of normalization: using *decomposed* forms allows a model to learn rules separately for an alphabetic character and its diacritic mark or combining character, which might be desirable in some cases—e.g., learning that a nasal bar often corresponds to modern ⟨*n*⟩ or ⟨*m*⟩—but not in others where this complex source character is mapped to a single normalized character. Furthermore, always decomposing characters leads to unnatural representations in many languages (such as German or Hungarian), where umlauted or accented characters are seen as

[17] "Punctuation characters" for this purpose are all characters in `string.punctuation` in Python 3.5.
[18] This is implemented by calling `lower()` on Unicode strings in Python 3.5.

clearly distinct from their plain counterparts. For this reason, and for the sake of consistency between datasets, I choose to convert all characters to their *composed* forms instead.[19]

Note that this still does not guarantee that all datasets use consistent and valid representations for all characters. Simon (2014) reports that the Hussite ⟨ʧ⟩, a character resembling a capital letter 'L', has no Unicode codepoint and is therefore encoded as ⟨č⟩ in the Hungarian dataset (Sec. 3.1.3). The English dataset (Sec. 3.1.1) still uses digits instead of the proper Unicode codepoints to represent some characters, e.g., ⟨3⟩ in place of the Middle English letter "yogh" ⟨ȝ⟩. These issues, however, are beyond the possibilities of an automatic preprocessing step.

3.3 Character alignment

The training data for the normalization experiments consists of word pairs matching historical tokens to their gold-standard normalizations. Some normalization tools internally perform character alignment of this data: the Norma tool derives replacement rules and learns edit weights based on character-aligned word forms, while the cSMTiser tool applies statistical machine translation on a character level, which requires calculating an alignment between characters as well. (Both tools will be introduced in more detail in Secs. 4.2.1 and 4.2.2, respectively.)

Explicitly generating character-aligned data can still be useful for data analysis. In particular, I will use character alignments of the datasets for measuring spelling variation and dataset similarity (Sec. 3.4) as well as analyzing the generalization capabilities of the proposed neural network model by relating its (word-level) performance to properties of the character alignments (Sec. 7.5).

3.3.1 Iterated Levenshtein alignment

Character alignment is performed using the same method as in the Levenshtein-based normalizer of the Norma tool (cf. Bollmann, 2013a, p. 22). It is based on the observation that the algorithm for calculating Levenshtein distance of two strings can be modified to obtain character alignments for these strings (by recording the edit operations required to transform one string into the other), but these alignments are not necessarily unique. Consider the example *jre* > *ihre*, which has a Levenshtein distance of 2 that can be reached in two different ways (Bollmann, 2013a, p. 18):

(14) a. j r e
i h r e

b. j r e
i h r e

However, alignment (14-a) is much more plausible than (14-b), since writing ⟨j⟩ for modern ⟨i⟩ is very common in historical German texts. This is the motivation for using *iterated Levenshtein distance alignment*, originally intended for aligning word pronunciations (Wieling et al., 2009).

[19] This corresponds to the *Normalized Form C (NFC)* as defined by Davis and Whistler (2017).

Its main idea is to resolve ambiguities by adjusting the weights of edit operations so that more plausible alignments (such as $j > i$) are assigned lower weights. More precisely, it uses pointwise mutual information (PMI) (Church and Hanks, 1990) to derive these weights automatically from training data. A slightly modified version of the original algorithm is used here:[20]

1. Character alignments are generated using the weighted Levenshtein algorithm; initially, the weights are set to 1 for all replacement, insertion, and deletion operations.
2. PMI values are calculated for all character pairs and used to update the weights for individual edit operations (see below for details).
3. Steps 1 and 2 are repeated until the weights no longer change significantly.

If $x \in S$ and $y \in T$ are source (= historical) and target (= normalized) characters, respectively, their PMI value is calculated as:

$$\mathrm{PMI}(x, y) = \log_2 \left(\frac{p(x, y)}{p(x)p(y)} \right) \tag{3.1}$$

Here, $p(x)$ and $p(y)$ are the relative frequencies of the source and target character, respectively, among all alignment pairs, while $p(x, y)$ is the relative frequency of the alignment (x, y), i.e., the number of times this alignment was observed divided by the number of all alignment pairs. The weight $d(x, y)$ for aligning x and y—i.e., the cost of the edit operation $x > y$—is then set as follows:

$$\hat{d}(x, y) = \max_{\tilde{x} \in S, \tilde{y} \in T} \big(\mathrm{PMI}(\tilde{x}, \tilde{y})\big) - \mathrm{PMI}(x, y) \tag{3.2}$$

$$d(x, y) = \frac{\hat{d}(x, y)}{\max_{\tilde{x} \in S, \tilde{y} \in T} \hat{d}(\tilde{x}, \tilde{y})} \tag{3.3}$$

The higher the PMI of the aligned characters—i.e., the closer their association—the lower $\hat{d}(x, y)$ will be. To bring the new weights in line with the default costs of zero for identity alignments and one for all replacements, insertions, and deletions, they are normalized to be within the range $[0, 1]$ (Equation (3.3)).

This adjustment of weights cannot only serve to disambiguate two or more alignments with the same minimal Levenshtein distance, but also result in new alignments that were previously not considered. Take the following example from the Anselm dataset:

(15) a. e r b e ſ h a f t
e r b s c h a f t
b. e r b e ſ h a f t
e r b s c h a f t

[20] The implementation is publicly available at https://github.com/mbollmann/levenshtein/.

Alignment (15-a) has a plain Levenshtein distance of 2, as it can be reached by performing two substitutions ($e > s$ and $ſ > c$), while (15-b) is more plausible (as it correctly aligns $ſ > s$), but has a Levenshtein distance of 3. After iterative Levenshtein distance alignment on the Anselm dataset (cf. Sec. 3.3.2), Example (15-b) actually ends up being the preferred alignment with the lowest associated cost.

On the other hand, the algorithm does not guarantee to always produce sensible alignments. Consider the word pair *nitt* > *nicht* (from the German RIDGES dataset):

(16) a. n i t ∅ t
n i c h t
b. n i ∅ ∅ t t
n i c h t

Here, alignment (16-a) is produced, which dubiously maps ⟨*t*⟩ to ⟨*c*⟩, while the preferred alignment should arguably not align these elements at all, as in (16-b). Other instances are not downright wrong, but at least debatable—vowels with nasal bars, for example, tend to be aligned to the letter representing the nasal sound rather than the vowel:

(17) d ē
d e n

This latter example could be addressed by using decomposed forms of Unicode characters (cf. Sec. 3.2) so that the diacritic is treated as a separate entity from the vowel.

Finally, the datasets frequently contain instances that are inherently undecidable. This usually happens when one of the layers contains a doubled character while the other does not, as in this example from the Innsbruck Letter Corpus:

(18) a. w r i t e n
w r i t t e n
b. w r i ∅ t e n
w r i t t e n

There is no way to distinguish between these two alignments with weighted Levenshtein distance alone, as both will always have the same cost associated with them, no matter the weight configuration. In those cases, the first of those alignments is always chosen.

3.3.2 Generating aligned datasets

To generate the character alignments for each dataset, the algorithm for iterative Levenshtein distance alignment (described in the previous section) is trained on the combined training and development sets of each historical corpus, and then applied to training, development, and test sets. However, these alignments may contain insertions, i.e., characters in the normalization

that are not aligned to any character in the historical word form. Consider this alignment of the word pair *orubelly* > *horribly* from the English dataset:

(19)
```
  o r  u b e l l y
h o r r i b  l  y
```

In this example, the initial ⟨*h*⟩ and the second ⟨*r*⟩ of the normalization *horribly* have no equivalent in the source word. This is problematic when we want to use this character-aligned data as input for an normalization algorithm: while the alignments are known for the gold-standard training (and evaluation) datasets, during test time, the positions where insertions will occur are still unknown. More precisely, the segmentation of the historical word form can only depend on the word form itself, since its normalization is yet to be generated. Bollmann et al. (2011b) address this problem by inserting epsilon symbols between all characters to which inserted characters are aligned, but this is only a partial solution as any number of characters can be inserted at any given position.

The solution chosen here is identical to that of Bollmann and Søgaard (2016), which is to perform a leftward merging of inserted characters. This means that instead of being unaligned, inserted characters are now linked to the historical character left to the position of the insertion. Consequently, it is possible for historical characters to be aligned to more than one normalized character. This works in all cases except when an insertion occurs at the beginning of a word, since there is no historical character to the left of this position. Therefore, all historical word forms are prepended with an epsilon symbol (*ε*) to serve as the alignment target for these insertions.

Deletions, on the other hand, do not need special treatment in this scenario. On the normalized side, there are also aligned with an epsilon symbol (*ε*) to make the deletion process more explicit. With these processing steps, the final character-aligned version of Example (19) looks like this:

(20)
```
ε o r  u b e l l y
h o r r i b ε l ε y
```

3.4 Analyzing variation

The performance of automatic normalization is influenced by a variety of factors; one which has been extensively discussed (in Secs. 2.4 and 3.1) is the differences in the guidelines and decisions employed when creating the gold-standard data that we train and evaluate on. Furthermore, not all historical written language is created equal. There is the question of time period: a "historical text" might be a document from the 9th century or from the 19th, but the older a document, the more different it will be from contemporary language. However, languages and their writing traditions changed to different degrees, and languages underwent their standardization process at different times, making the age of a document alone a poor predictor for how challenging it will be to normalize. Various other factors—e.g., text genre, manuscript vs. print, education of the scribe—may also play a role in this.

Dataset		Tokens	Types		TTR		HNR	sHNR
			Hist	Norm	Hist	Norm		
DE_A	German (Anselm)	233,947	24,915	6,517	0.1065	0.0279	3.8231	1.1468
DE_R	German (RIDGES)	41,857	9,698	7,210	0.2317	0.1723	1.3451	1.0663
EN	English	147,826	17,942	9,760	0.1214	0.0660	1.8383	1.0952
ES	Spanish	97,320	12,717	9,302	0.1307	0.0956	1.3671	1.0548
HU	Hungarian	134,028	44,005	25,817	0.3283	0.1926	1.7045	1.0834
IS	Icelandic	49,633	9,451	8,040	0.1904	0.1620	1.1755	1.0468
PT	Portuguese	222,525	25,874	15,499	0.1163	0.0697	1.6694	1.0951
SL_B	Slovene (Bohorič)	50,023	14,256	10,824	0.2850	0.2164	1.3171	1.0829
SL_G	Slovene (Gaj)	161,211	34,089	30,143	0.2115	0.1870	1.1309	1.0213
SV	Swedish	24,458	7,768	5,914	0.3176	0.2418	1.3135	1.0835

Table 3.2: Ratios of types and tokens on the training sets; TTR = type/token ratio on the historical text (Hist) and its gold-standard normalization (Norm); HNR = historical/normalized type ratio; sHNR = standardized HNR, calculated as the average HNR over chunks of 1,000 tokens.

A simple way to assess the extent of variation within a dataset is to look at the relationship between types and tokens. Table 3.2 gives an overview of the amount of tokens as well as types on either the original (historical) or the normalized side of the token pairs. type/token ratios (TTRs) are given for completeness' sake; they vary considerably between datasets (e.g., ranging between 2.79% and 24.18% on the normalized types), but are also not really comparable, since they are sensitive to a variety of properties such as corpus size, lexical variation, and morphological properties of the language.

Instead, we look at the *historical/normalized type ratio (HNR)*, defined as the number of types in the source text divided by the number of types in its normalization.[21] Table 3.2 shows that the Slovene/Gaj dataset is the least variant one in this regard, with a HNR of 1.1309; this is also the most "modern" dataset in terms of time periods, containing only texts written after around 1840. The German/Anselm dataset, on the other hand, is an extreme outlier, with almost four times as many historical types than modern ones (HNR 3.8231). This can partially be explained by its limited lexical diversity, as it is based on a collection of semi-parallel texts, which are expected to overlap significantly in vocabulary. The fact that it has less types in its normalization than the RIDGES dataset, despite it being more than five times larger in terms of tokens, further supports this hypothesis. At the same time, the historical versions of the Anselm texts originate from scribes of various dialectal regions, resulting in a highly diverse set of spellings—and, therefore, a comparatively high amount of historical types.

The example above suggests that the HNR can serve to highlight certain types of bias in a dataset. It also shows, however, that this score conflates effects of spelling variation and lexical diversity. To reduce the effect of corpus size and compilation, we also calculate the *standardized historical/normalized type ratio (sHNR)*. Inspired by similar approaches for type/token ratios (TTRs), the sHNR is calculated by dividing the corpus into chunks of n tokens, calculating

[21] This is equivalent to the ratio of the historical and normalized TTRs.

Dataset		Word-based		Char-based	
		ID	MFN	ID	MFN
DE_A	German (Anselm)	29.58%	94.36%	58.66%	83.33%
DE_R	German (RIDGES)	43.80%	95.55%	69.17%	90.82%
EN	English	74.94%	98.02%	73.25%	92.58%
ES	Spanish	72.90%	97.18%	73.90%	93.85%
HU	Hungarian	17.62%	98.00%	57.44%	78.33%
IS	Icelandic	46.69%	92.39%	66.99%	87.12%
PT	Portuguese	65.33%	97.41%	70.95%	91.36%
SL_B	Slovene (Bohorič)	41.07%	98.29%	64.44%	90.70%
SL_G	Slovene (Gaj)	86.24%	99.04%	79.48%	97.56%
SV	Swedish	59.85%	99.18%	73.31%	90.60%

Table 3.3: Accuracy on the training sets if tokens were left unchanged (ID) or mapped to their most frequent normalizations (MFNs), both on the standard word-aligned and the character-aligned versions of the datasets.

the HNR for each chunk, and then taking the average of these scores. To calculate sHNR in Table 3.2, we use $n = 1000$. Here, German/Anselm still has the highest ratio by a large margin, though the tendencies change a bit for some other datasets. For example, the Hungarian dataset has a noticeably higher HNR than Slovene/Bohorič—1.70 vs. 1.32, respectively—while the standardized score is almost the same for both (sHNR 1.08). This suggests that the comparatively higher HNR of Hungarian might be a product of its larger total size, and not necessarily of greater spelling variation.

Table 3.3 provides a different way of quantifying variation, both on a word level and a character level (cf. Sec. 3.3), by giving the percentage of words/characters that do not change in the normalization (columns "ID") and the theoretical accuracy obtained by mapping each word/character to its most frequent normalization (columns "MFN").

The first, column "ID", shows the extent of spelling variation in terms of how many words and/or characters are affected by it. In the Hungarian dataset, only 17.6% of words do not need any normalization, while 86.2% are left unchanged in Slovene/Gaj. This again confirms that the spelling of the Slovene/Gaj dataset is closest to its contemporary language compared to all other datasets; at the same time, the Slovene texts written in the older Bohorič alphabet show considerably more variation. These numbers could conceivably help explain some of the different approaches to spelling normalization in previous work. For example, the VARD tool for English first detects if a word form is a spelling variant at all before trying to normalize it (Baron and Rayson, 2008), while the Norma tool (originally developed for German) makes no such distinction and runs its normalization algorithms on all input word forms equally (Bollmann, 2012). The former approach makes sense if at most one in four word forms needs to be normalized at all (as for the English dataset in Table 3.3), while a variant detection step is arguably less necessary if less than 30% of words do not need normalization (as in the German Anselm dataset).

While these numbers show how many words are affected by (spelling) variation, they provide no information on how different these word forms are from their modern counterparts. This question can be addressed by looking at the character-aligned data (from Sec. 3.3) instead. As an example, compare the Portuguese dataset with German/RIDGES: the former has a much higher percentage of unchanged words (65% vs. 44%), while the percentage of unchanged characters is roughly comparable between the two (71% vs. 69%). This suggests that while fewer word forms need to be normalized in Portuguese compared to German/RIDGES, in the words that do change, more characters need adjusting in the Portuguese dataset. In general, the numbers for the character-aligned datasets are much closer together than the word-based scores, with most datasets hovering around 70% unchanged characters.

The percentage for most frequent normalizations (MFNs) in Table 3.3 is determined by counting the number of word pairs that map a historical token to the most frequent normalized word form observed for all instances of that historical type. This can be seen as a measure of ambiguity, discussed more thoroughly in Sec. 3.4.1, as this score will be lower the more often a historical type is normalized as different contemporary word forms.[22] Consequently, it gives an impression of how context-dependent the normalization task is—if all historical word types only had a single correct normalization (i.e., the MFN accuracy is 100%), there would be no need to consider word context to disambiguate them when generating the normalization.

In practice, the MFN accuracy for most datasets is above 97%. This provides a good justification for using word forms in isolation as the input to the normalization algorithm, as almost all previous work on normalization has done (cf. Chapter 4): less than 3% of all gold-standard normalizations will be unreachable in such a scenario. Furthermore, it is not guaranteed that including word context in the normalizer's decisions will allow us to reach these remaining 3% of word forms, as they might also be the result of inconsistent annotation or idiosyncracies of the data. On the other hand, some datasets show more variation in this regard, particularly the Icelandic one with a MFN accuracy of only 92.4%. This suggests that Icelandic might profit more than other languages from an approach that considers word context.

3.4.1 Measuring ambiguity

The analysis above already touched upon the issue of ambiguity: historical source types that have more than one gold-standard target normalization. Looking at the MFN frequency (in Table 3.3) already provides some insight into the extent of ambiguity, but it is a very shallow measure, as it does not give any indication about the distribution of ambiguous word forms. A lower MFN percentage could be the result of many slightly ambiguous words or a few highly ambiguous ones (or anything in-between). For further analyses or improvements to our models, we might also be interested in knowing what the most problematic word forms in this regard are.

For this purpose, let us define the *ambiguity* α of a historical word type or token w as follows:

$$\alpha(w) = \log_2 \left(\frac{c(w)}{\text{MFN}(w)} \right) \tag{3.4}$$

[22] However, it can also be influenced by inconsistencies in the gold-standard normalizations.

Here, $c(w)$ is the number of tokens w in the dataset, and MFN(w) is the number of times they are assigned their most frequent normalization. For example, if the historical word form *ye* occurs 30 times in a dataset and is normalized as *you* in 20 of these instances, and as *the* (or anything else) in the rest, then c('ye') = 30 and MFN('ye') = 20, resulting in an ambiguity score of α('ye') ≈ 0.585.

More generally, if a historical source type is completely unambiguous, i.e., all instances of it are always assigned the same normalization, then $c(w)$ = MFN(w) and therefore $\alpha(w) = 0$. On the other hand, if $\alpha(w) = 1$, this means that the most frequent normalization only makes up 50% of all occurrences of this word type. The lower the MFN for a word type, the higher α will be, up to a theoretical maximum of $\log_2 c(w)$ (if every instance of the word had a different normalization).

Table 3.4 gives an overview of the distribution of α on the different datasets. The percentage of fully unambiguous words (i.e., $\alpha = 0$) varies between 48.2% (for German Anselm) and 93.6% (for Swedish), but is mostly in the range of 50%–75%. Icelandic again stands out as having rather high ambiguity, with almost 20% of tokens having an ambiguity score of $\alpha > 0.3$, a considerably higher ratio than any other dataset. For most datasets, the average α across all tokens tends to be below 0.05, exceptions being Icelandic and the German corpora. In combination, these numbers suggest that most historical tokens tend to have relatively unambiguous normalizations, with alternative normalized forms either occuring only rarely, or being limited to a small number of word types.

Figure 3.1 provides a more detailed visualization of the ambiguity distribution by showing the values of α at a given percentile. First of all, the plot shows that values of $\alpha > 1$ are extremely rare, affecting less than 1% of all tokens. Furthermore, the graphs for most datasets follow a very similar distribution, with the major exceptions again being German (with both the Anselm and the RIDGES dataset being relatively close together) and Icelandic. The latter already branches off noticeably around the 76$^{\text{th}}$ percentile and mostly stays on top of all other graphs, meaning that most of the top 24% of tokens in the Icelandic dataset (in terms of ambiguity) have a higher α score than those of the other datasets.

Finally, we can use our definition of ambiguity to not only measure and visualize the ambiguity distribution, but also to identify the most ambiguous word types. However, looking at α alone is not very helpful, as the tokens with the highest α scores tend to occur only a handful of times. Words have a higher impact on the normalization accuracy when they are both ambiguous *and* relatively frequent. To that effect, we calculate a weighted version $\hat{\alpha}$ by taking the geometric mean of α and the relative token frequency:

$$\hat{\alpha}(w) = \sqrt{\alpha(w) * \frac{c(w)}{\sum c(\tilde{w})}} \tag{3.5}$$

Table 3.5 shows the ten most ambiguous word types according to this measure in each of the datasets.

Dataset		Token ambiguity			Distribution of α		
		$\alpha = 0$	$0 < \alpha \leq 0.3$	$\alpha > 0.3$	AVG	95$^{\text{th}}$ PCT	MAX
DE_A	German (Anselm)	48.19%	40.15%	11.66%	0.1011	0.8078	2.0000
DE_R	German (RIDGES)	62.17%	27.19%	10.64%	0.0787	0.5850	1.5850
EN	English	49.15%	47.24%	3.61%	0.0338	0.1799	2.0000
ES	Spanish	56.03%	40.46%	3.50%	0.0479	0.2410	1.5850
HU	Hungarian	73.24%	23.35%	3.41%	0.0345	0.1855	1.5850
IS	Icelandic	62.32%	18.02%	19.66%	0.1376	0.8634	1.5850
PT	Portuguese	49.64%	47.09%	3.27%	0.0435	0.1631	2.1699
SL_B	Slovene (Bohorič)	84.67%	11.79%	3.53%	0.0301	0.1375	1.5850
SL_G	Slovene (Gaj)	80.17%	18.25%	1.57%	0.0162	0.0526	1.7004
SV	Swedish	93.61%	4.69%	1.70%	0.0148	0.0123	1.5850

Table 3.4: Token ambiguity on the training sets; left half shows the percentage of tokens falling in a given range of α; right half shows the average α, the α score at the 95$^{\text{th}}$ percentile, and the maximum α on the datasets.

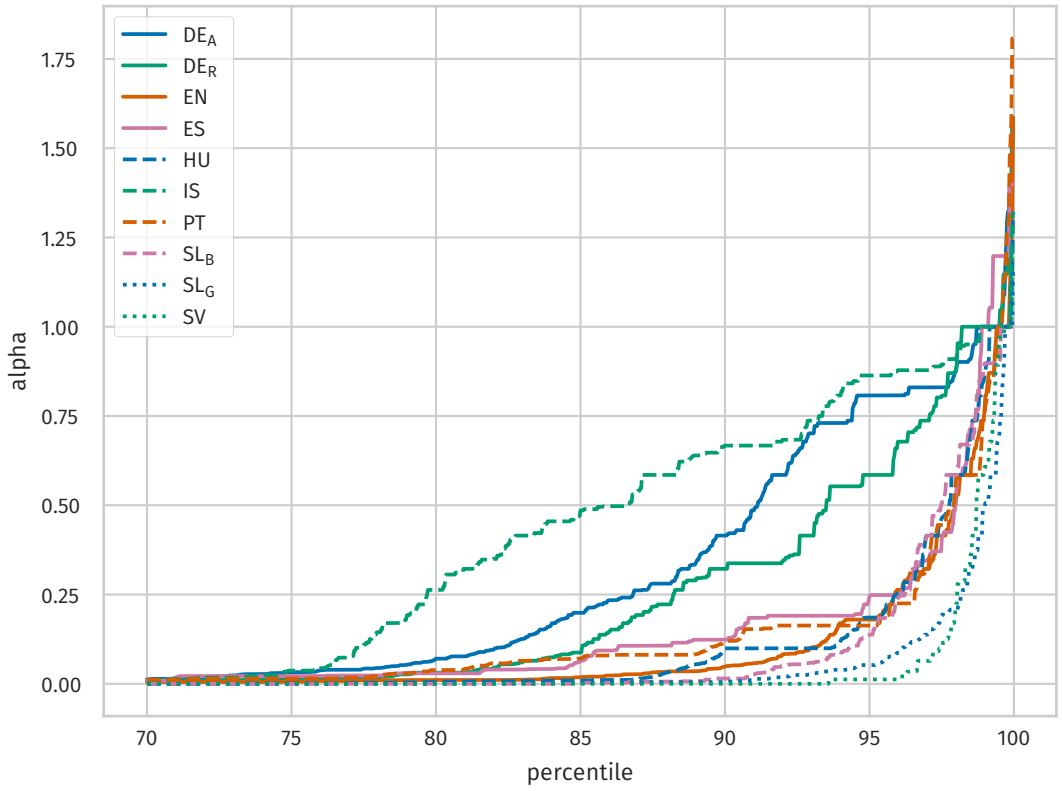


Figure 3.1: Distribution of ambiguity scores (α) as a quantile function; for any given value x_i on the x-axis, x_i percent of all tokens in the dataset have an ambiguity score $\alpha \leq f(x_i)$.

Word	Count	α	$\hat{\alpha}$	Normalizations
in	3685	0.8078	0.1128	ihn (2105), in (1573), ein (4), en (3)
das	2996	0.8303	0.1031	dass (1685), das (1308), da (3)
daz	2409	0.7304	0.0867	dass (1452), das (957)
dat	904	0.9013	0.0590	das (484), dass (419), da (1)
wer	451	0.9968	0.0438	wäre (226), wer (204), war (20), wär (1)
was	1556	0.2806	0.0432	war (1281), was (274), wasser (1)
her	205	1.3941	0.0350	her (78), er (62), herr (49), heer (12), höre (1), hier (1), hör (1), haar (1)
het	636	0.4302	0.0342	hat (472), hätte (120), hatte (42), es (1), hattte (1)
dz	374	0.7014	0.0335	dass (230), das (138), des (6)
waz	580	0.4217	0.0323	war (433), was (147)

(a) German (Anselm)

Word	Count	α	$\hat{\alpha}$	Normalizations
das	791	0.3375	0.0799	das (626), dass (165)
ein	446	0.5530	0.0768	ein (304), eine (109), einen (31), einer (1), einem (1)
dann	106	0.8699	0.0469	denn (58), dann (48)
dĕ	62	1.1468	0.0412	dem (28), den (27), der (4), denen (2), des (1)
dz	88	0.7045	0.0385	das (54), dass (33), des (1)
wider	62	0.9542	0.0376	wieder (32), wider (27), widder (2), weder (1)
diſz	68	0.8021	0.0361	dies (39), dieses (21), das (4), diese (4)
ſein	80	0.6781	0.0360	sein (50), seine (14), sind (10), seiner (3), seinen (2), seines (1)
vil	51	0.7182	0.0296	viel (31), viele (16), vielen (4)
weiſz	31	1.0473	0.0279	weiß (15), weise (8), weiße (4), weisheit (1), weisen (1), weißem (1), weißer (1)

(b) German (RIDGES)

Table 3.5: Top 10 ambiguous words (by weighted $\hat{\alpha}$) in the training sets

Word	Count	α	$\hat{\alpha}$	Normalizations
be	1638	0.1799	0.0446	be (1446), by (104), been (35), are (27), is (23), am (2), the (1)
then	267	0.8709	0.0397	then (146), than (121)
ye	374	0.3133	0.0282	you (301), the (63), ye (5), we (2), yes (2), they (1)
yt	113	0.6304	0.0220	it (73), that (40)
ther	151	0.4310	0.0210	there (112), their (38), since (1)
here	207	0.3098	0.0208	here (167), hear (30), her (5), their (4), year (1)
the	5504	0.0105	0.0198	the (5464), you (28), they (9), that (2), by (1)
a	1492	0.0352	0.0189	a (1456), have (14), an (8), he (3), on (2), ado (1), has (1), i (1), and (1), a((1), as (1), ah (1), o’ (1), à (1)
mr	91	0.5305	0.0181	mister (63), mr. (14), mr (12), master (2)
ben	67	0.7085	0.0179	been (41), be (7), are (7), ben (7), benjamin (3), benedicta (1), have (1)

(c) English

Word	Count	α	$\hat{\alpha}$	Normalizations
es	4816	0.0992	0.0597	és (4496), is (312), ez (4), ás (2), es (2)
monda	447	0.2845	0.0308	mondá (367), monda (65), monda´ (12), mondja (3)
meg	489	0.1855	0.0260	meg (430), még (51), míg (4), megy (2), mégy (2)
kezde	69	0.8991	0.0215	kezde (37), kezdé (32)
mikoron	121	0.4759	0.0207	mikoron (87), mikor (34)
el	190	0.2479	0.0187	el (160), él (17), ily (9), élj (3), ez (1)
mēt	42	1.0704	0.0183	mint (20), mert (16), ment (6)
ada	52	0.8425	0.0181	adá (29), ada (23)
v̇tet	49	0.7567	0.0166	őt (29), őtet (20)
felele	54	0.6674	0.0164	felele (34), felele´ (11), felelé (8), felül (1)

(d) Hungarian

Word	Count	α	$\hat{\alpha}$	Normalizations
enn	840	0.6671	0.1063	en (529), enn (310), og (1)
nu	579	0.8782	0.1012	nú (315), nu (258), un (6)
j	564	0.8634	0.0991	i (310), í (252), j (2)
suo	439	0.4973	0.0663	svo (311), sou (128)
uar	318	0.6835	0.0662	var (198), aur (120)
med	403	0.4550	0.0608	með (294), med (109)
þier	172	0.8413	0.0540	þér (96), þeir (76)
kongr	154	0.9092	0.0531	kóngur (82), göngur (72)
þu	213	0.6472	0.0527	þú (136), ðu (76), þu (1)
sier	143	0.9504	0.0523	sér (74), sjer (69)

(e) Icelandic

Table 3.5: Top 10 ambiguous words (by weighted $\hat{\alpha}$) in the training sets (cont.)

Word	Count	α	$\hat{\alpha}$	Normalizations
a	7979	0.1631	0.0765	a (7126), à (604), há (188), anos (23), ana (15), arroba (3), amigo (3), o (3), ao (2), ah (2), antónio (2), alma (1), alteza (1), ano (1), almeida (1), e (1), antunes (1), antónia (1), alpoim (1)
esta	874	0.5841	0.0479	esta (583), está (291)
o	5710	0.0813	0.0457	o (5397), ao (235), ou (65), ó (8), os (3), aos (1), oh (1)
he	1585	0.2251	0.0400	é (1356), e (192), hei (30), he (3), em (2), aí (2)
ma	480	0.5760	0.0352	minha (322), maria (99), ma (35), má (10), mesma (7), mau (2), meia (1), manuel (1), há (1), me (1), mãe (1)
m	140	1.8074	0.0337	mercê (40), maria (40), me (13), muito (10), muitos (10), majestade (6), mano (4), mesmo (3), miranda (3), manuel (3), minha (2), mestre (1), m (1), meu (1), moreira (1), martiniano (1), matias (1)
d	288	0.8480	0.0331	de (160), dona (49), dom (47), deus (8), da (7), do (6), d. (4), di (2), diogo (2), don (1), doutor (1), adeus (1)
nos	524	0.4447	0.0324	nos (385), nós (138), nosso (1)
s	191	1.1345	0.0312	são (87), sua (22), santo (20), senhor (19), santa (10), se (8), seu (6), senhora (4), scilicet (3), senhoria (3), soror (2), as (1), santidade (1), santos (1), servidor (1), sacramento (1), serva (1), seja (1)
as	1320	0.1532	0.0301	as (1187), às (118), anos (13), das (1), hás (1)

(f) Portuguese

Word	Count	α	$\hat{\alpha}$	Normalizations
esta	546	1.1979	0.0820	esta (238), está (170), ésta (138)
a	2917	0.1906	0.0756	a (2556), ha (327), años (32), aquí (1), arroba (1)
mi	1049	0.2485	0.0518	mi (883), mí (165), mía (1)
m	145	1.3985	0.0456	mano (55), muchos (32), maría (16), me (12), majestad (9), manos (4), merced (3), mil (3), mi (2), mis (2), memorias (2), medina (1), muchas (1), misericordioso (1), madre (1), no (1)
el	1787	0.1064	0.0442	el (1660), él (127)
se	1110	0.1234	0.0375	se (1019), sé (85), si (5), septiembre (1)
como	632	0.1848	0.0346	como (556), cómo (75), comercio (1)
s	110	1.0534	0.0345	su (53), señor (36), servidor (4), seguro (4), san (2), sus (2), seda (1), s (1), siempre (1), señora (1), señoría (1), se (1), suyo (1), suya (1), si (1)
tu	255	0.4245	0.0334	tu (190), tú (64), tus (1)
este	291	0.3711	0.0333	este (225), éste (38), esté (27), estén (1)

(g) Spanish

Table 3.5: Top 10 ambiguous words (by weighted $\hat{\alpha}$) in the training sets (cont.)

Word	Count	α	$\hat{\alpha}$	Normalizations
s	272	0.8976	0.0699	z (146), s (126)
s’	183	0.6702	0.0495	z (115), s (65), iz (3)
ko	115	0.4705	0.0329	ko (83), kot (32)
is	225	0.1844	0.0288	iz (198), z (15), s (10), zlahkoma (1), izmed (1)
se	93	0.3894	0.0269	se (71), si (22)
she	89	0.2469	0.0210	že (75), še (14)
more	32	0.6781	0.0208	more (20), mora (12)
ſvojim	32	0.6077	0.0197	svojim (21), svojem (11)
leto	37	0.5090	0.0194	leto (26), le-to (11)
leta	26	0.7004	0.0191	leta (16), le-ta (10)

(h) Slovene (Bohorič)

Word	Count	α	$\hat{\alpha}$	Normalizations
ko	469	0.2081	0.0246	ko (406), kot (63)
ste	169	0.5061	0.0230	ste (119), sta (50)
o	293	0.2004	0.0191	o (255), ob (37), zilogoro (1)
z	978	0.0526	0.0179	z (943), s (35)
svojim	91	0.4854	0.0166	svojim (65), svojem (25), svoje (1)
vsaki	61	0.6828	0.0161	vsak (38), vsaki (23)
ti	344	0.1179	0.0159	ti (317), tej (25), ta (1), tem (1)
tak	49	0.8074	0.0157	tak (28), tako (21)
saj	92	0.3738	0.0146	saj (71), vsaj (21)
veči	36	0.9220	0.0143	večji (19), večjo (9), večje (7), večja (1)

(i) Slovene (Gaj)

Word	Count	α	$\hat{\alpha}$	Normalizations
bemälte	47	0.9696	0.0432	bemälda (24), bemälde (23)
haffua	26	0.8931	0.0308	har (14), ha (12)
hafwa	32	0.6077	0.0282	ha (21), har (11)
j	50	0.2863	0.0242	i (41), ni (7), j (2)
här	34	0.3326	0.0215	här (27), herr (7)
alt	24	0.4150	0.0202	allt (18), att (6)
kunne	13	0.7004	0.0193	kunna (8), kunde (4), kunnat (1)
hafva	14	0.6374	0.0191	ha (9), har (5)
ware	7	1.2224	0.0187	vara (3), vare (3), var (1)
wåre	7	1.2224	0.0187	våra (3), vore (2), vår (1), var (1)

(j) Swedish

Table 3.5: Top 10 ambiguous words (by weighted $\hat{\alpha}$) in the training sets (cont.)

Many of the ambiguities in Table 3.5 are the result of homophones or near-homophones that are clearly distinguished in contemporary orthography, but not (necessarily) in historical writing. Prominent examples include: *das/dat/daz/dz* in the German datasets, which can be normalized as either the definite article *das* or the conjunction *dass*; the German *in* from the Anselm dataset which can be either the preposition *in* or the pronoun *ihn*; the English *then* > *then/than* or *ther* > *there/their*; the Spanish and Portuguese *esta* that is normalized as either the demonstrative pronoun *esta* or the inflected verb form *está*; or the Slovene preposition *z/s*, which is often just written as *s/s'* in the historical texts. In Swedish, which has very few ambiguous word forms, a common source of ambiguity appears to be the obsolete forms *haffua/hafwa/hafva* of the modern *ha/har* '(to) have'.

Other cases are exemplary of the nature of a dataset or its normalizations. In German/RIDGES, many common ambiguities arise from historical word forms missing inflectional suffixes, such as *difz* > *dies/dies(e|es)*, *ſein* > *sein/sein(e|er|en|es)*, or *weifz* > *weiß/weiß(e|er|em)*, while examples like these are not prominent in the Anselm dataset. This is a consequence of their different normalization guidelines, as the Anselm corpus moves the adjustment of inflectional morphemes to a separate layer (cf. the discussion on p. 32), while the RIDGES corpus treats it as a normal part of normalization. The Post Scriptum corpus stands out for containing a high number of abbreviations, with many one-letter or two-letter tokens—such as *a*, *d*, *m*, *ma*, or *s*—mapping to an unusually high number of normalizations. For example, in the Portuguese dataset, the historical token *s* is normalized as *são/se/seu/sua*, but also as *senhor/senhora* 'mister/lady', *santo/santos/santa* 'holy/sacred', *santidade* 'sanctity', *sacramento* 'sacrament', and a few others. English also has a few of these cases, with *ben* including the normalizations *benjamin* and *benedicta*.

Icelandic is of particular interest, since it has the highest curve in the ambiguity distribution of Figure 3.1 and is quite different from all other datasets in this regard. Comparing the normalization targets in Table 3.5e to the word forms in the modern Icelandic datasets introduced in Sec. 3.5, we find that many of the gold-standard normalizations are unattested there: e.g., the contemporary resources have *nú*, but not *nu*; *svo* but not *sou*; *þú* but neither *ðu* nor *þu*; etc. This leads me to believe that the increased frequency of ambiguous word forms might—at least in parts—be the result of erratic or inconsistent normalizations in the gold-standard data.

3.4.2 Measuring similarity

A question that will be analyzed in later chapters (Ch. 8 and 9) is whether training on a pair of datasets in parallel can yield better results than training on a single dataset in isolation, and if so, whether the effectiveness of a dataset pairing can be traced to common (or diverging) properties of these datasets. For the latter aspect, I propose an approach to measure the similarity between two datasets.

Dataset similarity is estimated here by transforming the training sets into feature vectors, applying term frequency–inverse document frequency (tf–idf) weighting, then measuring

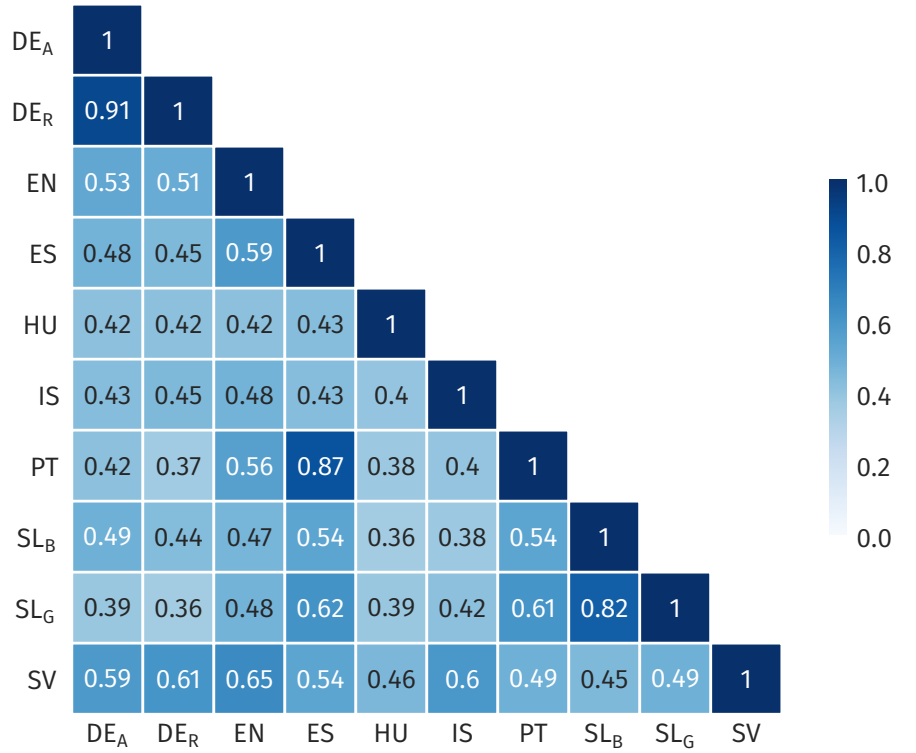


Figure 3.2: Cosine similarity of datasets based on tf–idf of historical character bi- and trigrams

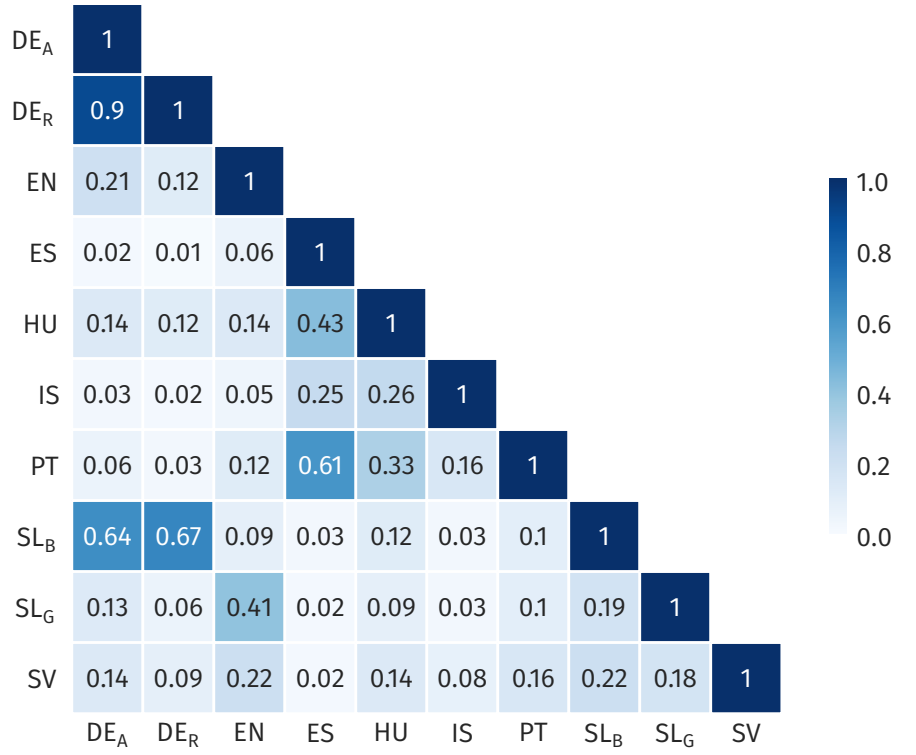


Figure 3.3: Cosine similarity of datasets based on tf–idf of non-identical character alignments

pairwise cosine similarity between the datasets. This is also called the *cosine coefficient*. For a dataset $d \in D$ and a feature φ, we define the tf–idf score to be:

$$\text{tf–idf}(\varphi, d) = \text{tf}(\varphi, d) * \text{idf}(\varphi) \tag{3.6}$$

$$= \text{tf}(\varphi, d) * \left(\ln \frac{|D|}{\text{df}(\varphi)} + 1 \right) \tag{3.7}$$

Here, $\text{tf}(\varphi, d)$ is the number of times feature φ appears in dataset d, while $\text{df}(\varphi)$ is the number of datasets that contain feature φ. The addition of 1 in Eq. (3.7) is done to prevent features that are present in all datasets from vanishing completely, which emphasizes similarity over individual differences.[23]

Figure 3.2 shows the pairwise similarity of datasets when using character bi- and trigrams from the historical text as features. This serves to give an impression of how similar the historical datasets are in terms of spelling. Not surprisingly, datasets of the same or similar languages score highest in this comparison: the German Anselm and RIDGES datasets show the highest similarity (0.91), with the pairs Portuguese–Spanish (0.87) and Slovene Bohorič–Gaj (0.82) following close behind. Some of the other scores also vaguely follow language families: take Swedish as an example, for which the highest similarity scores occur with the English, Icelandic, and German datasets, i.e., those from the other Germanic languages. Similarly, Hungarian, which is the only dataset from a Finno-Ugric language, has the lowest overall scores (< 0.46).

Figure 3.3 shows dataset similarity when using non-identical character alignments (from Sec. 3.3) as features; identical alignments—i.e., whenever a character is normalized as itself—are filtered out because of their high frequency across all datasets. This comparison highlights the similarity in terms of character-based normalizations, i.e., it takes the mapping of historical to contemporary spelling into account. Again, the two German datasets are the most similar ones (0.9), while Portuguese–Spanish has a considerably lower score (0.61). The Slovene datasets, however, receive a very low score of 0.19 in this evaluation, which makes sense since they were written in different alphabets. Interestingly, Slovene/Bohorič also shows a high similarity (0.64–0.67) to both of the German datasets. The Bohorič alphabet was "modelled on the German one" (Erjavec, 2015, p. 755) and frequently uses ⟨ſ⟩ for modern ⟨s⟩, a substitution that almost never appears in the other datasets except for German and, to a lesser extent, Hungarian. Other scores seem to be more coincidental: e.g., the similarity of 0.43 between Spanish and Hungarian appears to be caused by both datasets frequently mapping vowel characters to their accented counterparts, such as *a* > *á* or *e* > *é*, although the usage of acute accents is of course different between the two languages.

[23] I calculated dataset similarities with many more configurations than shown here. When using the more traditional definition of inverse document frequency without the **+1** term, many dataset pairs receive a similarity score of zero because $\text{tf–idf}(\varphi, d) = 0$ for many features φ. Also, different features (e.g., character bi-/trigrams vs. character alignments) tend to produce nearly identical similarity scores. The variant of tf–idf in Eq. (3.7) was ultimately chosen because it allows for more interesting and diverse perspectives on the data.

3.5 Contemporary datasets

In addition to the historical datasets, some experimental setups also make use of a contemporary language resource. Firstly, the rule-based and distance-based algorithms of Norma (cf. Sec. 4.2.1) require a list of valid target word forms to function properly. Secondly, adding a lexical filter that only allows "valid" word forms to be generated can also be a helpful strategy for neural network models.

The main criteria for a resource to be usable in this way are: (i) it should consist of inflected word forms; (ii) it should have a comprehensive coverage of the modern language, and particularly all parts of speech; and (iii) it should be relatively noise-free, i.e., not contain any non-words. Ideally, we would like to use a carefully curated *full form lexicon* for this purpose; however, such a lexicon is not readily available for many languages. A simpler alternative is to derive a list of word forms from a corpus.

For my experiments, I mainly consider two different resources of contemporary language: the Europarl corpus (Koehn, 2005) and modern Bible translations. The only exception is Icelandic, which is the only language in my experiments that is not represented in Europarl, so other modern Icelandic resources are used instead. The following sections describe all the resources, why they were chosen, and some of their properties.

3.5.1 Europarl

The Europarl corpus[24] is a parallel corpus of European parliament proceedings from 1996 to 2011 (Koehn, 2005). Release v7 of the corpus, which is the one used here, contains between 7 and 55 million words per language from a total of 21 European languages. The corpus is freely available online, including parallel corpora for all language pairs that include English.

There are several reasons for choosing Europarl as one of the contemporary language resources:

1. It is freely available and covers all languages used in my experiments except one (Icelandic).
2. Since it is based on carefully transcribed and translated parliament speeches, it is comparatively free of spelling errors, non-words, or other features of informal writing that are likely to be found in, e.g., a corpus derived from the web.
3. Its parallel nature allows us to control for size, domain, and vocabulary of the contemporary dataset for all covered languages.

An obvious drawback is that the domain of these texts—political speeches—is quite distant from that of the historical texts, which for the most part contain religious treatises, scientific texts, and personal letters (cf. the overview in Table 3.1). The main alternative here is to use more varied corpora that are available for each given language; I choose the parallel corpus here for better comparability.

[24] http://www.statmt.org/europarl/

Not all languages in the Europarl corpus are represented in equal quantities. In order to obtain comparable datasets for each language, I choose to only use sentences which are represented in all of the seven languages investigated here (English, German, Hungarian, Portuguese, Slovene, Spanish, and Swedish; Icelandic is not available in the Europarl corpus). This is done via a simple heuristic: Based on the parallel corpus files from the Europarl website, I first extract all non-English sentences and index them by their aligned English counterparts. Then, all English sentences that do not have equivalents in all of the six other languages are filtered out (along with their translations). The remaining sentences are tokenized using the tokenization script provided by the Europarl corpus, lowercased, and used to obtain a wordlist for each language by extracting all word types.

This process results in about 450,000 sentences per language, with each language consisting of a total of 10–13 million tokens and between 55,000 and 268,000 types (cf. the overview in Table 3.6).

3.5.2 BÍN and MÍM

Since the Europarl corpus does not cover Icelandic, a different resource needs to be used for that language. I follow Pettersson (2016) in using a combination of two Icelandic resources: the *Beygingarlýsing íslensks nútímamáls (BÍN)*,[25] a database of Modern Icelandic inflected word forms (Bjarnadóttir, 2012); and all tokens from the *Tagged Icelandic Corpus (Mörkuð íslensk málheild, MÍM)*[26] (Helgadóttir et al., 2012) occurring at least 100 times. This frequency threshold "is chosen due to a considerable amount of noisy corpus data" (Pettersson, 2016, p. 80).

The BÍN database is a very extensive resource, contributing about 2.9 million word types, while the high-frequency words from the MÍM corpus constitute only about 9,300 types. However, the latter also includes function words while BÍN does not.

3.5.3 Bible

Modern Bible translations share many of the traits of the Europarl corpus: they are readily available for many languages, consist of carefully edited text, and constitute highly parallel texts. Moreover, they are more likely to overlap in domain with historical texts, which are often religious in nature. This makes it another ideal resource for contemporary word forms.

The data used here comes from the parallel Bible corpus by Christodouloupoulos and Steedman (2015).[27] It contains Bible translations in more than 100 languages, among them complete Bible texts (both Old and New Testament) for all eight languages investigated here. Plain text files were extracted from the corpus files using specialized, open-source processing tools,[28] while tokenization was done with the Europarl tokenizer, before again lowercasing all words and collecting word types.

[25] http://bin.arnastofnun.is/data/
[26] http://mim.hi.is/
[27] http://christos-c.com/bible/
[28] https://github.com/christos-c/bible-corpus-tools

Language	Bible	Modern		Historical (Train)	
	Types	Resource	Types	Resource	Types
English	12,572	Europarl	54,752	ICAMET	9,760
German	20,492	Europarl	164,005	Anselm	6,517
				RIDGES	7,210
Hungarian	64,332	Europarl	268,391	HGDS	25,817
Icelandic	35,389	BÍN+MÍM	2,875,081	IcePaHC	8,040
Portuguese	29,374	Europarl	85,304	Post Scriptum	15,499
Slovene	39,528	Europarl	124,609	goo300k (Bohorič)	10,824
				goo300k (Gaj)	30,143
Spanish	27,089	Europarl	84,255	Post Scriptum	9,302
Swedish	23,776	Europarl	140,066	GaW	5,914

Table 3.6: Overview of contemporary word types in the modern Bible translation, the Europarl corpus (or BÍN+MÍM for Icelandic), and of the gold-standard normalizations in the training set of the historical corpora.

The resulting wordlists contain about 600,000 to 800,000 tokens per language. This is considerably less than for the Europarl corpus (cf. Sec. 3.5.1), and suggests that the Bible alone might be a little too short for extracting a comprehensive full form wordlist. Therefore, I use it only as a supplement to the other contemporary language resources, and not as a standalone replacement.

3.5.4 Coverage

For each historical dataset, there are now two external sources of modern inflected target word forms: (i) a modern Bible translation (Sec. 3.5.3); and (ii) another modern corpus or dictionary resource—BÍN+MÍM for Icelandic (Sec. 3.5.2), and Europarl (Sec. 3.5.1) for the other languages. In addition to those, we can also include the gold-standard normalizations of the respective training sets in our list.

Table 3.6 gives an overview of the number of word types obtained from these three sources. The training sets always provide the lowest amount of types, suggesting that the addition of external resources can be helpful to increase coverage. Of these, the modern resources (Europarl/BÍN+MÍM) provide significantly more word types than the Bible translations.

However, a larger amount of types is not necessarily always better. Noise in the corpus, i.e., words that are not actually valid forms of the target language, does not provide any value, but can increase the number of types significantly. This should not be much of an issue here, though, as the resources were chosen specifically to be relatively free of noise. Another factor is how well the chosen resources cover the gold-standard normalizations on the datasets we evaluate on—since the contemporary resources are intended to guide the normalization process by providing valid target word forms, the effectiveness of a resource is restricted by how much it overlaps with the set of correct target normalizations.

Dataset		Bible	Modern	Bible+Modern	Train	All
DE_A	German (Anselm)	9.17%	8.37%	5.46%	3.63%	1.52%
DE_R	German (RIDGES)	19.37%	15.93%	12.11%	10.15%	5.32%
EN	English	11.58%	5.38%	4.35%	3.76%	1.81%
ES	Spanish	12.02%	6.44%	5.25%	4.62%	1.64%
HU	Hungarian	27.52%	33.55%	19.02%	11.64%	7.17%
IS	Icelandic	22.44%	13.50%	13.10%	9.22%	3.55%
PT	Portuguese	11.74%	7.01%	5.35%	3.44%	1.45%
SL_B	Slovene (Bohorič)	16.88%	16.44%	11.61%	14.64%	6.15%
SL_G	Slovene (Gaj)	21.52%	15.92%	12.19%	11.65%	6.29%
SV	Swedish	28.51%	20.54%	16.93%	16.08%	8.29%

Table 3.7: Tokens in the development sets of the historical corpora *not* covered by the contemporary language resources; Modern = wordlist extracted from BÍN+MÌM for Icelandic (Sec. 3.5.2) or Europarl for all other languages (Sec. 3.5.1); Train = wordlist extracted from gold-standard normalizations of the respective training sets.

Table 3.7 shows how many of the gold-standard normalizations from the development sets of the historical datasets are *not* covered by the contemporary wordlists. Essentially, this is the percentage of target normalizations that *cannot* be reached if the given resource is used to strictly filter the normalization candidates. In this sense, a higher percentage is worse because it reduces the maximum normalization accuracy we can obtain. Importantly though, this does *not* mean that a lower score is always better: a list containing all possible strings—with all combinations of all characters—would naturally include all possible normalizations, resulting in 0% of normalizations not covered, but such a list is obviously not very useful. Therefore, these numbers only provide an upper bound on the usefulness of the resources for filtering normalizations.

The numbers show that wordlists extracted from the training sets of the corpora almost always provide a better coverage of the development sets than the Bible and the other modern resource combined, the only exception being Slovene (Bohorič). This demonstrates the large effect of domain bias, since training and development sets are always taken from the same corpus, while the external resources are taken from different corpora and domains. This is true even for the datasets of religious texts when compared to the Bible: the Bible does not cover 9.17% of word forms of the German Anselm development set, while its training set is only missing 3.63% of these word forms; the differences are even higher for the other religious corpora (e.g., 22.44% vs. 9.22% for Icelandic, or 27.52% vs. 11.64% for Hungarian). Combining the training sets with the external resources always results in a large decrease of this figure, though, again suggesting that such a combination can be beneficial. With all three resources combined, the percentage of tokens not covered is between 1.45% and 8.29%.

The lowest scores are obtained on the English, German, Portuguese, and Spanish datasets. The two German datasets, however, show a large discrepancy, with the RIDGES dataset having significantly worse coverage than the Anselm dataset in all configurations (e.g., 5.32% vs. 1.52% on all resources combined). This effect could result from a bias in the Anselm dataset due to its semi-parallel nature (cf. Sec. 3.1.2), causing it to be less diverse in terms of vocabulary. The

difference to the RIDGES dataset could be further influenced by the latter containing a higher number of proper nouns or Latin terms (e.g., herbal terms) or using different normalization guidelines.

The Hungarian dataset shows some of the highest numbers of Table 3.7, with a missing coverage of 33.55% when using the Hungarian part of the extracted subset from the Europarl corpus, and 7.17% when using all three resources combined. At the same time, this Europarl subset as well as the Hungarian Bible translation contain much more types than their respective counterparts in the other languages (cf. Table 3.6), e.g., about 268,000 types in the Hungarian Europarl subset compared to 164,000 types in the second-largest subset (German). This is almost certainly an effect of the morphological properties of Hungarian: being an agglutinative language, words will typically consist of a higher number of morphemes, resulting in a larger variety of word types. Compare this to English, which has lost most of its inflection, and consequently has by far the lowest amount of types within the external resources (e.g., 55,000 types in Europarl) as well as the lowest percentage of tokens not covered by them (4.35%). These figures suggest that an approach based on full form wordlists might generally be less suitable for morphologically rich languages.

Another notable data point comes from the Icelandic corpus, which uses BÍN+MÍM as its modern resource instead of Europarl. The number of types contained within them is almost 2.9 million, exceeding those in the Europarl subsets by more than a factor of 10 (cf. Table 3.6). Still, this huge amount of types is not reflected in the coverage on the historical dataset, which is only average among all datasets, with 13.5% of tokens missing in the wordlist from BÍN+MÍM. The numbers for Swedish are equally surprising, as the coverage here profits the least from its own training set, resulting in the highest number of missing tokens there (16.08%) as well as for all resources combined (8.29%). This is unexpected since Swedish morphology is much closer to that of English than, e.g., Hungarian, meaning that this effect cannot be explained by morphological properties. Instead, it is possible that these numbers highlight another shortcoming of full form wordlists: proper nouns. These form an open-ended class that can only ever be partially covered by a finite list of words. The Swedish dataset contains official records about the occupation of citizens (cf. Sec. 3.1.7), which are likely to include a high number of personal names and place names.

3.6 Summary

This chapter introduced all corpora that will be used in the following chapters, both historical ones (providing datasets for automatic normalization) and contemporary ones (serving as auxiliary data for some of the normalization approaches). In total, the corpora cover eight languages: English, German, Hungarian, Icelandic, Slovene, Spanish, Swedish, and Portuguese. Whenever possible, dataset splits from previous work on normalization have been reused.

The guidelines used to create the gold-standard normalizations can and do differ between corpora (Sec. 3.1). Detailed normalization guidelines are not always published in the first place—information was particularly sparse on the English and Icelandic datasets. In most corpora, word forms are adjusted to the correct (contemporary) inflected forms in their respective context. Notable exceptions are the German/Anselm dataset, where inflection is not modified at all in

the "normalization" layer, and German/RIDGES, which does not adjust inflection for gender and/or grammatical case when these do not conform to modern conventions. Treatment of archaic elements is handled very differently: some corpora always replace them with equivalent contemporary lexemes or morphemes (English, German/Anselm), some always preserve them and only normalize them graphematically (Hungarian, Slovene, Spanish, Portuguese), while others preserve extinct lexemes but not bound morphemes (German/RIDGES, Swedish).

For the contemporary datasets, mainly two corpora are used: Europarl and a dataset consisting of bible translations (Sec. 3.5). They have been chosen primarily for their coverage of languages, as the only language that is not covered is Icelandic in the Europarl corpus, for which two Icelandic resources (BÍN and MÍM) have been selected instead. Preprocessing for all datasets—historical and contemporary—includes the removal of punctuation, lowercasing of all characters, substitution of digits, and performing Unicode normalization (Sec. 3.2).

Finally, a large part of this chapter was concerned with quantifying properties of the datasets, particularly (*intra*-dataset) ambiguity and (*inter*-dataset) similarity (Secs. 3.3 and 3.4). I proposed an ambiguity measure α that is based on the frequency of a historical token compared to its most frequent normalization, and showed that the historical datasets differ significantly in this regard (Fig. 3.1), with Icelandic and the German corpora containing relatively many tokens with ambiguous normalizations, while Slovene/Gaj and Swedish contain relatively few. For dataset similarity, I proposed a measure based on cosine similarity of tf–idf scores, and showed that it reflects, e.g., the close relationship between Spanish and Portuguese, or the close relation of Slovene/Bohorič spelling to German orthography (Fig. 3.3).

Chapter 4

Methods for automatic normalization

(Semi-)automatic normalization of historical texts has a long history. As soon as computers were utilized for philological analyses, scholars noted the troubles caused by excessive spelling variation:

> [S]i deux mots diffèrent si peu que ce soit, ce sont pour [la machine] deux mots totalement différents. La collation automatique fournit alors une masse de variantes inutiles et insignifiantes, qu'il faudra ensuite éliminer.[1] (Froger, 1970, p. 212)

Earlier attempts to perform normalization automatically often rely on word substitution lists or hand-crafted rules that encode common spelling variants; they are typically tailored to one specific language and require expert knowledge of that language (and its historical variant) to build. Other attempts are inspired by automatic spelling correction, essentially treating historical word forms as "misspellings" of the modern words. These methods typically use some form of phonetic coding algorithm or a string distance measure to find the correct modern cognate. Later approaches try to infer spelling characteristics automatically from a training set of manually normalized word forms, e.g. to learn a set of replacement rules. More recent work is based on the application of statistical machine translation or neural network models. These supervised algorithms are not strictly language-specific and do not require manually encoding domain knowledge, but they are reliant on having a manually normalized training resource. Nonetheless, since corpora with manual normalization layers are becoming increasingly available, supervised models have become the de facto method of choice in recent years.

In the context of spelling correction, Pollock (1982) classifies automatic methods into two groups: *absolute* and *relative*. Absolute methods infer the correct spelling directly from the deviant word form, e.g. by means of replacement rules, while relative methods make use of a reference list of correct target word forms, e.g. by calculating distance scores. Piotrowski (2012) applied this classification to historical spelling normalization methods. For the following overview, I will not adopt this classification, but rather group normalization methods by the main technique(s) they employ. To this end, I will distinguish the following techniques:

[1] 'If two words differ even slightly, they are two completely different words to the machine. Automatic collation then supplies a large amount of unnecessary and unimportant variants which consequently have to be eliminated.' (Froger, 1970, p. 212, my translation)

- *wordlist mapping,* which simply lists variants together with their modern form;
- *rule-based methods,* applying some form of transformation rules which can be either manually designed or learned from training data;
- *distance-based methods,* employing a string distance measure such as Levenshtein distance (Levenshtein, 1966);
- *statistical models,* typically based on established techniques from statistical machine translation; and
- *neural models,* implementing some form of neural network.

Not every work on normalization can be strictly classified into one of these categories; e.g., some methods use distance measures to derive replacement rules, while others use a combination of approaches. Nevertheless, I believe that each of these categories represents a conceptually or methodically different approach to this task, to the extent that this classification results in a useful aggregation of previous work. In the terminology of Pollock (1982), distance-based methods can mostly be described as a *relative* strategy, while most of the other categories fall under *absolute* strategies.

In the context of this work, I am mainly concerned with spelling normalization as the mapping of different historical variants to a single modern word form, e.g., for improved accuracy of downstream applications, or for use as an additional layer of information in an annotated corpus. However, not all work that deals with normalization shares this perspective. In particular, work in the context of information retrieval (IR) is often concerned with finding equivalent variant word forms given a modern word form as a search query. This is, in a way, going in the opposite direction: it does not require connecting each historical token to a single modern form, but rather generating plausible historical variants based on a modern token (cf. Pilz et al., 2007). Furthermore, while I am dealing with normalizing to *inflected forms,* a substantial amount of work approaches the task from the perspective of lemmatization, which is not concerned with morphology. While results from these works may not be directly comparable to normalization as it is presented here, the overlap is substantial enough to warrant inclusion in the following overview.

Historical text is not the only domain that concerns itself with (spelling) normalization. Contemporary dialectal texts, for example, face very similar challenges (cf. Samardžić et al., 2015; Scherrer and Ljubešić, 2016). Social media data, e.g. from Facebook or Twitter, is typically also full of variation, and similar considerations apply that make it desirable to map this data to a standard language form (cf. Eisenstein, 2013; Baldwin and Li, 2015). Indeed, many approaches for historical spelling normalization have also been applied to social media language, and vice versa. There are a few notable properties of social media language, though, that distinguish it from the historical domain: reduplication of letters (*whaaaaat*), typing errors resulting from the ideosyncracies of keyboard layouts, frequent use of acronyms (*afk*, *brb*), emoticons (*:-)*), URLs, hashtags, etc. These features result in specific choices during the preprocessing step or require additions/alterations to the normalization step—e.g., inserting placeholders for URLs, reducing duplicated letters, detecting and resolving acronyms, and so on. For these reasons, I feel that normalizing social media data is different enough from the historical domain to be treated as a separate problem, although it can be worthwhile to consider which methods from this area could be adapted to the historical domain (and vice versa).

Finally, techniques used for other string-to-string transduction tasks can also be applicable to normalization, particularly when they employ generic sequence-to-sequence algorithms. Examples for such tasks include transliteration (e.g., Knight and Graehl, 1998; Li et al., 2009) or morphological inflection generation (e.g., Durrett and DeNero, 2013; Faruqui et al., 2016).

In the remainder of this chapter, I will first give an overview of previous work on automatic normalization (Sec. 4.1), grouped by the categories presented above, before describing two specific normalization tools that I will use for a comparative evaluation (Sec. 4.2).

4.1 Previous work

This section will give an overview of previous work on historical text normalization, following the classification laid out previously on page 63.

4.1.1 Wordlist mapping

The conceptually simplest form of automatic normalization is to look up each historical word form in a pre-compiled list that maps it to its supposed "best" normalization. This approach can go by many names, such as *dictionary lookup* or *lexical substitution*; I will mainly call it *wordlist mapping* here.

Compiling a list of "known variants" is the main principle of the first VARD tool (Rayson et al., 2005), a semi-automatic normalization tool for Early Modern English. Its main goal is to assist the user in finding and normalizing variant historical forms; the list of known variants (along with their normalizations) is used to highlight these potentially variant forms and suggest modern substitutions. VARD 2 keeps this substitution list as one of several normalization components (Baron and Rayson, 2008).

Wordlist mapping is also a critical component of the Norma tool (Bollmann, 2012). During training, a list of all historical tokens along with their gold-standard normalizations is compiled. For normalization, this list is applied in a fully automatic fashion as the first step in a "chain" of normalizers. Each historical token is first looked up in the wordlist: if it is found there, it is replaced with the most frequent normalization that was seen for this word during training; if it is not, other algorithms are used to suggest a normalization candidate (cf. Sec. 4.2.1 for more details).

Despite its simplicity, the wordlist mapping approach can often provide correct normalizations with a high degree of accuracy, as the later evaluations (e.g., in Sec. 7.2) will also show. Its main drawback, of course, is that it does not generalize in any way to previously unseen word forms, as it does nothing more than memorize the training data. Still, the high effectiveness of this memorization means it should not be easily dismissed as a potential component in a normalization system.

4.1.2 Rule-based approaches

While spelling variants in historical texts can be numerous and inconsistent, it is usually possible to identify patterns that occur frequently and in texts by more than one writer. A common example from German (as well as some other languages) is the letter ⟨*v*⟩, which can be used in the same way as modern ⟨*v*⟩—i.e., to represent a labiodental fricative—or in place of modern ⟨*u*⟩. Rule-based systems try to encode these regularities in the form of replacement rules, typically including some form of context information to discriminate between the different usages of a character.

Some of the earliest approaches to historical text normalization are rule-based, with rules being manually created for one particular target language. Fix (1980) describes such an approach for normalization of Old Icelandic lemmata. It is comprised of several processing steps, some of which implement normalization rules of the form "replace ⟨*i*⟩ with ⟨*j*⟩ in front of a vowel." Koller (1983) describes a system for Old German which segments word forms into morphemes before applying similar replacement rules, e.g.:

(1) u f * *V

This rule defines a substitution *u* > *f* whenever ⟨*u*⟩ appears morpheme-initially (*) and is followed by a vowel grapheme (*V). The system is designed for a grammar consisting of up to 150 replacement rules.

Bollmann et al. (2011b) describe a system that uses similar normalization rules, but instead of defining them manually, the rules are derived automatically from a training set of gold-standard normalizations.[2] In contrast to Ex. (1), these rules may not refer to grapheme classes (such as vowels or consonants) but only to individual graphemes, though they may also refer to word boundaries and sequences of graphemes, e.g.:

(2) j ⟶ ih / # _ n

This rule describes the substitution *j* > *ih* when ⟨*j*⟩ is preceded by a word boundary (#) and followed by ⟨*n*⟩. Rules are not only learned for actual modifications, but also for "identities" of characters between the historical and modern word, and from all applicable rules, only the most probable one (based on frequencies in the training corpus) is applied during normalization.

The context restrictions in replacement rules are necessary to prevent overgeneralization and, consequently, producing the wrong modern word forms. However, many systems are concerned with an information retrieval (IR) perspective of finding historical variant spellings given a modern word form (e.g. Hauser et al., 2007). In this scenario, rule-based systems can produce a list of potential candidate spellings which are then matched against the historical corpus. While these systems still need to balance precision and recall of the generated word forms, they can afford to use more general and fuzzy rules, since their output is a list of candidates instead of a single, accurate word form.

[2] Sec. 4.2.1 describes this system in more detail.

Barnbrook (1996, Chapter 8) describes a simple system for finding spelling variants in Chaucer's *Canterbury Tales*: word pairs that differ by only one letter are extracted, and a frequency analysis is performed to rank the types of differences found between the pairs. Most likely differences included "⟨*i*⟩ substituted for ⟨*y*⟩", inserted ⟨*u*⟩, word-final ⟨*e*⟩, and doubled characters (⟨*a*⟩, ⟨*c*⟩, ⟨*e*⟩, ⟨*o*⟩, and ⟨*t*⟩). Applying these characteristics to modern word forms results in candidates for spelling variants which are then looked up in the historical text; e.g., *brought* was mapped to the historical variants *broughte*, *broghte*, and *broght*. Essentially, this approach can be described as applying very broad replacement rules that are not restricted by context.

Many approaches in IR also rely on rules defined by domain experts. Pilz et al. (2006) describe a fuzzy search engine for historical German texts that uses rules "derived from statistical analyses, historical publications, linguistic principles, and expert knowledge." Giusti et al. (2007) use 43 manually developed transformation rules for Brazilian Portuguese, some of which are context-dependent while others are not. However, their goal is to cluster spelling variants, which means that their rules do not necessarily have to describe a mapping to a modern form. Porta et al. (2013) implement context-aware phonological sound change rules for Old Spanish, using edit transducers to convert input strings to phonemic representations and (after applying the sound changes) back to sequences of graphemes.

Etxeberria et al. (2016) utilize a finite-state transducer for modeling phonological changes, which is trained on a set of gold-standard normalizations. Besides evaluating on historical Basque, they found that their model outperforms Porta et al. (2013) on Spanish and Scherrer and Erjavec (2016), who use a machine-translation approach, on Slovene.

Koolen et al. (2006) describe another method to construct rules automatically from a training sample. Their rules are derived based on phonetic sequence similarity as well as relative frequencies of consonant/vowel sequences and character n-grams, and evaluated on a text collection of historical Dutch. Similarly, Ernst-Gerlach and Fuhr (2006) describe an algorithm to automatically learn a set of probabilistic transformation rules, which is then used to transform a modern word form into a set of historical spellings.

In general, most of the work with rule-based normalization systems has been carried out with the IR task in mind. A possible explanation is that transformation rules tend to overgeneralize, and it is not obvious how to apply them in a scenario where a single target normalization is desired. An exception is the VARD 2 tool (Baron and Rayson, 2008), which uses context-free "letter replacement rules" to find variant spellings. However, these rules only make up one of several components used by the tool; and at least for its interactive mode, the generated normalizations only constitute a set of suggestions for the user, making overgeneralization less of a problem.

Nonetheless, the approach by Bollmann et al. (2011b) shows that fully automatic rule-based methods can be used for producing a single modern target normalization, while the evaluation by Etxeberria et al. (2016) suggests that such models can be competitive with other approaches.

4.1.3 Distance-based approaches

Edit distance, also called *Levenshtein distance* after Levenshtein (1966), is a measure of the difference between two strings.[3] In its most commonly used form, the edit distance between two strings is defined to be the minimum number of edits required to transform one string into the other, where "edits" can be either: (i) the *insertion* of a character; (ii) the *deletion* of a character; or (iii) the *substitution* of one character with another.

Weighted Levenshtein distance is a variant of this measure that allows to assign weights to individual edit operations. For a set of edit operations that transform string a into string b, consider the sum of its weights; the distance between a and b is then defined as the minimum sum from all possible sets of edit operations that transform a into b. Under this definition, the "plain" Levenshtein distance is simply a special case where the weight for all insertions, deletions, and substitutions is set to 1. Note that while plain Levenshtein distance is always symmetric (i.e., $\mathrm{LD}(a, b) = \mathrm{LD}(b, a)$ for all possible strings a, b), its weighted variant does not need to be; e.g., the transformation $j > i$ does not need to have the same weight as the transformation $i > j$.

Normalization approaches that use distance metrics are most commonly found when normalization is performed in the context of *information retrieval (IR)*. This is because, by definition, a distance metric requires two strings to compare, which is a natural fit for an IR scenario that aims to match up a search term with relevant word forms in a (historical) document.

Robertson and Willett (1993) investigate this exact scenario for queries on 16th–18th century English texts and find that edit distance is one of the methods giving the best results (the other being the longest common subsequence metric). Kempken et al. (2006) perform a similar evaluation on historical German and also find edit distance to be effective, additionally proposing *FlexMetric*, a form of weighted Levenshtein distance with a training algorithm to derive weights automatically from manually defined training word pairs. Hauser and Schulz (2007) similarly learn edit distance weights for IR on historical English and German, but additionally introduce an unsupervised algorithm that tries to match up spelling variants with lexicon entries.

Mapping historical tokens to their standardized forms in a modern lexicon or corpus is another common use case for distance metrics. Kestemont et al. (2010) show that plain Levenshtein distance is quite effective for lemmatization by taking the lemma from the closest word form in a training corpus. Jurish (2010a) includes a distance-based transducer in a normalization pipeline. The Norma tool (Bollmann, 2012) includes a distance-based normalization component that is inspired by FlexMetric and works by finding the lexicon entry with the lowest distance to the historical source string. Pettersson, Megyesi, and Nivre (2013) find a similar approach to be more effective than hand-crafted rules on Swedish.

String similarity measures can also be used to compile a dictionary of historical spelling variants in an unsupervised way (Amoia and Martínez, 2013; Barteld et al., 2015); this can be seen as a "clustering" of the historical variant forms, whereas mapping them to a single normalized form is not necessarily desired. Adesam et al. (2012) use the Levenshtein algorithm to automatically

[3] Technically, the term "edit distance" can be applied to other metrics that measure similarity (or dissimilarity) between strings as well; Levenshtein distance is probably the most common edit distance metric, though, and the two terms will be used interchangeably throughout this thesis.

derive "substitution rules" from training data, which are then used to link up historical Swedish word forms with lexicon entries; Halteren and Rem (2013) describe a comparable approach for Dutch. Arguably, these latter approaches could also be classified as "rule-based" since they are described in terms of replacement rules, although they use distance metrics to derive these rules and apply them in a fashion very similar to an edit distance algorithm.

Generally, besides the IR scenario, distance-based approaches lend themselves to situations where a sizeable corpus or lexicon of the target language is available. Since they can work in an unsupervised way, they are a good fit when no or only little training data for the normalization task is available, although they can usually be improved by supervised training.

4.1.4 Statistical models

It is possible to take a probabilistic view of the normalization task, in which the goal is to optimize the probability $p(t|s)$ that a contemporary word form t is the normalization of a historical word form s. This can be interpreted as a *noisy channel model*, traditionally used for spelling correction (e.g., Brill and Moore, 2000) or machine translation, where the historical token is seen as a "distorted" version of the contemporary word, which is to be "restored". Some approaches apply the noisy channel model directly (e.g., Oravecz et al., 2010; Etxeberria et al., 2016); most, however, rely on existing toolkits from statistical machine translation (SMT), such as Moses (Koehn et al., 2007), and apply them to the normalization task.

Traditionally, the input for SMT systems has been a sequence of tokens; historical normalization applies these systems to a sequence of characters instead. This variant, also called character-based statistical machine translation (CSMT), was introduced by Vilar et al. (2007) and successfully applied to transliteration and the translation of closely related languages (Tiedemann and Nabende, 2009; Nakov and Tiedemann, 2012), two tasks that can be seen as very similar to historical normalization. Character-based approaches have become common for all forms of machine translation since then, though they are mostly used in a neural network framework (Ling et al., 2015).

For normalization, CSMT has been applied to Spanish (Sánchez-Martínez et al., 2013), Icelandic and Swedish (Pettersson, Megyesi, and Tiedemann, 2013), Slovene (Scherrer and Erjavec, 2013, 2016; Ljubešić et al., 2016b), as well as Hungarian, German, and English (Pettersson, 2016), where it was usually found to outperform previous approaches. Pettersson et al. (2014a) compare a filtering method, a distance-based approach, and a CSMT system on five languages and find that the CSMT system often (though not always) performs best. Schneider et al. (2017) compare the VARD 2 tool to CSMT on an English dataset and find that VARD 2 performs slightly better, although they note that VARD 2 was specifically developed and tuned for historical English, while the CSMT-based approach is not constrained to a specific language. Besides historical texts, CSMT has also successfully been applied to the normalization of dialectal texts, e.g., dialectal Swiss (Samardžić et al., 2015; Scherrer and Ljubešić, 2016).

Over the last few years, the CSMT-based approach has been both the most popular and the most successful overall for historical normalization, and can be considered the current state-of-the-art approach for this task.

4.1.5 Neural network models

Artificial neural networks constitute a family of machine learning algorithms that has become enormously popular in recent years, often branded under the term "deep learning", referring to the architecture of these models as a "deep" stack of individual layers. They have been successfully applied to a wide variety of NLP tasks, including sentiment classification, question answering, part-of-speech tagging, discourse parsing, and many others; see Goldberg (2017, Sec. 1.3) for a broad overview.

Previous work has applied neural networks to the normalization of historical German, either using a sequence labeling approach (Al Azawi et al., 2013; Bollmann and Søgaard, 2016) or an encoder–decoder architecture (Bollmann et al., 2017; Korchagina, 2017).[4] In all cases, the neural network has been found to outperform the Norma tool, while Korchagina (2017) also finds it to outperform a CSMT system. However, these evaluations were only focused on German and often performed using very small datasets.

Neural network models have also been highly successful in the field of machine translation, where they are often applied on a character level analogous to CSMT (e.g., Ling et al., 2015; Chung, Cho, et al., 2016; Wu et al., 2016; Lee et al., 2017). Considering this and the fact that CSMT-based systems are often used for historical normalization, it is surprising that neural networks have not been evaluated more thoroughly for this task. Some examples exist for noisy text normalization in the social media domain (e.g., Chrupała, 2014; Lusetti et al., 2018) or for lemmatization of historical texts (Kestemont et al., 2016), although the application to historical normalization is still rare.[5]

For this reason, the evaluation of (one particular kind of) neural network models on a large and varied set of historical corpora and their comparison to previously established methods is the central goal of this thesis.

4.2 Methods for comparison

In this thesis, I will focus on the description and analysis of a neural network model for normalization that will be introduced in later chapters (Ch. 5 & 6). To compare this model to previously established normalization methods, I will utilize two existing software tools:

1. Norma, which combines wordlist mapping, rule-based, and distance-based approaches.
2. cSMTiser, which uses established statistical machine translation software.

[4] Chapters 5 and 6 will introduce the neural encoder–decoder architecture in more detail.

[5] Between the time of my original thesis submission and preparing this revised version (09/2018), several works have been published on historical normalization with neural network models. Robertson and Goldwater (2018) compare soft and hard attention models, and also provide learning curves for training with different training set sizes. Domingo and Casacuberta (2018) evaluate both word-based and character-based models—with character-level models being superior for this task—and find that SMT outperforms the NMT approach. Tang et al. (2018), however, report the opposite result when evaluating with character error rate (CER) on five different datasets; they also compare many different neural architectures. Finally, Hämäläinen et al. (2018) evaluate SMT, NMT, an edit-distance approach, and a rule-based finite state transducer, and advocate for a combination of these approaches to make use of their individual strengths.

4.2.1 Norma

Norma (Bollmann, 2012) is a normalization tool originally developed for the Anselm Corpus (cf. Sec. 3.1.2). It is freely available[6] and has so far only been evaluated on varieties of historical German (Bollmann et al., 2011a, 2012; Bollmann, 2013a; Korchagina, 2017). Norma combines three different approaches:

1. A simple *wordlist substitution*.
2. A *rule-based* normalizer that applies rewrite rules on a character level, taking the immediate context into account.
3. A method using *weighted Levenshtein distance* to find the likeliest normalization candidate from a list of target word forms.

These components can either be used on their own or composed into a "normalizer chain", in which case they are applied in the order shown above, with methods further down the chain only being called if the previous one was not able to find a result (e.g., due to the historical word not being included in the word substitution list). Furthermore, all components learn their parametrization automatically from training data; none of them is specifically tailored towards German.

The *wordlist substitution* method simply learns the substitutions found in the training data. Whenever there is ambiguity, i.e., a historical token has been mapped to different normalized tokens, the most frequent substitution is always chosen.

The *rule-based* method derives rewrite rules from the training pairs by aligning them using the Levenshtein algorithm and merging neighboring alignments. Example rules are (2) or (3):

(3) a. u → üh / f _ h *(substitution rule)*
b. ε → m / m _ e *(insertion rule)*
c. n → ε / e _ # *(deletion rule)*
d. v → v / # _ o *(identity rule)*

Rule (3-a) describes the substitution *u* > *üh* between the two characters ⟨*f*⟩ and ⟨*h*⟩. Since word forms are always processed left to right, the left context refers to the already-normalized portion of the word, while the right context always refers to the remaining portion of the historical token. Therefore, substitution (3-a) could have been learned from a training pair like *vuren* > *führen*, for example. Context can also refer to word boundaries (#), while the source or target of a rule can also be an empty element (ε), indicating an insertion (rule (3-b)) or deletion (rule (3-c)) of a character. In addition to those modifications, "identities"—i.e., positions where characters are *not* changed—are also learned. During normalization, the normalizer attempts to find the most probable sequence of rules (where probability is defined by the rule frequency in the training corpus) that leads to a valid target word form.

A contemporary corpus or full form lexicon is used to define what counts as a "valid" target word form for the rule-based approach. The *weighted Levenshtein distance* method also requires

[6] https://github.com/comphist/norma

such a resource, as it simply picks the word form from that resource with the lowest distance to the historical word. Distance is calculated using a modified version of Levenshtein distance that assigns weights to individual edit operations, which in turn are derived from the Levenshtein alignments of the training data.

4.2.2 cSMTiser

cSMTiser is a freely available tool for normalization using character-based statistical machine translation (CSMT).[7] It builds upon the SMT toolkit Moses (Koehn et al., 2007), essentially implementing the system described by Ljubešić et al. (2016b). The basic idea is to apply Moses on a sequence of characters (as opposed to a sequence of words), normalizing a historical word form by "translating" it into a modern one on a character level.

To perform word-level normalization, each word is first split up into its characters, and a "word separator" symbol is added to the beginning and end of each word; e.g., if '÷' is the word seperator symbol, the word pair *ayeins* > *against* would be represented as:

(4) a. ÷ a y e i n s ÷
b. ÷ a g a i n s t ÷

The resulting word pairs are then fed into the Moses pipeline, i.e., GIZA++ (Och and Ney, 2003) is used to perform character alignment between the training pairs, and the statistical model is trained on them. Afterwards, a separate tuning set is used to perform parameter tuning using minimum error rate training (MERT).

The tool includes options for automatic tokenization, lowercasing, and truecasing of the data. I do not make use of any of these options, since the datasets are already tokenized and preprocessed (cf. Sec. 3.2) before feeding them into the tool. By default, cSMTiser trains a (character-based) language model of order 6 on the target-side training data, which I also did not change. Optionally, cSMTiser can use additional contemporary language data to improve this language model.

Essentially, cSMTiser provides a front-end for utilizing Moses for the normalization task, by taking care of the necessary data pre- and post-processing and providing sensible defaults for Moses' configuration files. It constitutes an easy, "off-the-shelf" solution for normalizing texts using CSMT.

[7] https://github.com/clarinsi/csmtiser

Chapter 5

Neural network basics

> *Although deep learning is a fairly old subfield of machine learning, it only rose to prominence in the early 2010s. In the few years since, it has achieved nothing short of a revolution in the field[.]*
>
> — Chollet (2017)

> *The neural network revolution has happened. We are living in the aftermath.*
>
> — Hanson and Olson (1991)

The main focus of this work is on using neural network models for automatic historical text normalization. On the one hand, neural networks have become enormously popular for many NLP tasks, and many publications are available describing the same or very similar types of networks as used here. On the other hand, neural networks are often very complex—involving many parameters and decisions that often cannot be exhaustively described due to page limits—and there is no single agreed-upon standard for many of their components.[1] Consequently, many variations of their core concepts are used in practice, but not always documented in detail.

The purpose of this chapter is to introduce the basic components of the neural networks used in this work, and document any relevant details of their implementation. While some of the core concepts and the motivation behind them will also be explained, this chapter cannot possibly replace a thorough, general introduction to neural networks—for that, see Goodfellow et al. (2016), Goldberg (2017) or Chollet (2017).

All neural networks models described and evaluated in this thesis are implemented using Keras (Chollet et al., 2015).[2] Any parameter or implementation detail that is not explicitly described is identical to the default behavior of Keras 1.2.2.

Sec. 5.1 briefly introduces neural networks in general, while Sec. 5.2 describes the individual neural network layers that will be used in my models. Sec. 5.3 discusses aspects of the training procedure.

[1] "Unfortunately, it is often the case that inferring the exact model form from reading its description in a research paper can be quite challenging. Many aspects of the models are not yet standardized, and different researchers use the same terms to refer to slightly different things." (Goldberg, 2017, p. 174)

[2] https://keras.io/

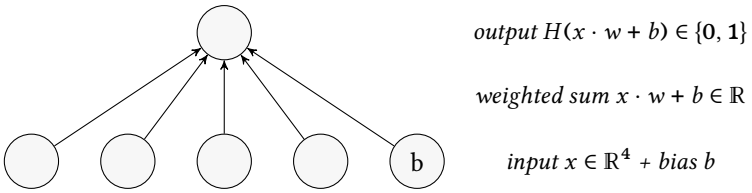


Figure 5.1: Perceptron for binary classification

5.1 Basic concepts

Neural networks belong to the huge family of *machine learning algorithms*. This means that instead of being specifically programmed for a given task, they are trained to perform that task, typically in a supervised manner by observing training pairs of input data and the desired gold-standard output(s). The *neural* part refers to these models being composed of *artificial neurons*, a type of mathematical function that calculates its output as a weighted sum of its inputs followed by some non-linear activation function. The *network* part refers to the fact that several of these functions are used in succession, with the output of one artificial neuron being connected to the input of another one, forming a directed graph or network. This "stacking" of individual components can lead to very "deep" networks where an input is passed through many different functions before an output is produced, giving rise to the term *deep learning* for this type of models.

The simplest type of neural network is the *perceptron* (Rosenblatt, 1958; Freund and Schapire, 1999). It can be defined as follows:[3]

$$f(x) = H(x \cdot w + b) \tag{5.1}$$

Here, the input $x \in \mathbb{R}^n$ is an n-dimensional vector, while the weight vector $w \in \mathbb{R}^n$ and the bias term $b \in \mathbb{R}$ are parameters of the model that are adjusted during training. H is the *Heaviside step function*, defined as:

$$H(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{5.2}$$

Figure 5.1 shows a visualization of this basic perceptron, which can be used for binary classification problems. It can also be extended to multi-class classification by replacing the weight vector with a weight matrix $W \in \mathbb{R}^{n \times m}$ (where m is the number of output classes), the bias term with a bias vector $b \in \mathbb{R}^m$, and the step function with the argmax function:

$$f(x) = \text{argmax}(x \cdot W + b) \tag{5.3}$$

This function essentially runs m perceptrons in parallel—one for each possible output class—and instead of applying the step function, the magnitude of the calculated value is used in determining which class is the most likely prediction. Importantly, the parameters W and b

[3] Some definitions of the perceptron omit the bias and assume that it is part of the input vector x (e.g., by having one dimension of the input vector always be 1); others use the sign function instead of the step function H, which returns $\{-1, +1\}$ instead of $\{0, 1\}$, but is functionally equivalent.

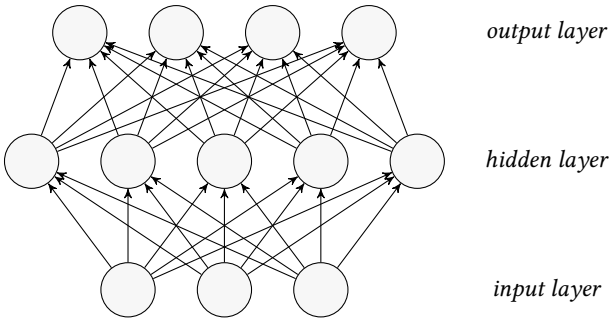


Figure 5.2: A multi-layer perceptron with one hidden layer

can be learned in a supervised manner from training data (for details of the training algorithm, see, e.g., Freund and Schapire, 1999).

The perceptron, either single-class or multi-class, is an example of a *linear classifier*. Consequently, it can only work well on problems where the input vector space $\mathbb{R}^n$ is *linearly separable* with regard to the output classes. In practice, this means that the composition of the input vector is often carefully tuned to make linear separability more likely, a process also known as *feature engineering*.

The *multi-layer perceptron (MLP)* circumvents this problem by introducing a *non-linear* function g:

$$f(x) = \text{argmax}(g(x \cdot W_1 + b_1) \cdot W_2 + b_2) \tag{5.4}$$

In practice, common choices for g are the hyperbolic tangent (tanh) or the sigmoid function. In the context of neural networks, g is also referred to as the *activation function*. With this definition, $f(x)$ is now a composition of two artificial neurons: the non-linear inner function $g(x \cdot W_1 + b_1)$, and the outer perceptron $(x \cdot W_2 + b_2)$ which takes the output from the inner function as its input. Since the output of the inner function is never directly observed, it is also called a *hidden layer*. Figure 5.2 shows one possible visualization of this model.

This multi-layer perceptron effectively consists of a non-linear transformation followed by a linear classifier; the purpose of the non-linear transformation—i.e., the hidden layer—is to map the input data into a vector space where it is linearly separable (e.g., Goldberg, 2017, p. 43). It is possible to add an arbitrary number of additional hidden layers before the linear transformation: simply replace the input vector x in Eq. (5.4) with another non-linear transformation $g_i(x \cdot W_i + b_i)$. However, even an MLP with a single hidden layer is a universal function approximator (Cybenko, 1989): i.e., with the right set of weight matrices W_1, W_2 and bias vectors b_1, b_2, the MLP function $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ as defined in Eq. (5.4) can approximate any other function $F : X \mapsto X$ as long as that function is continuous and defined on a compact subset X of $\mathbb{R}^n$.

In theory, MLPs are capable of expressing almost any function that we might wish to learn, such as mapping historical word forms to normalized ones. The problem, indeed, is in the learning, as the theoretical guarantee of universal approximation makes no claims about the *learnability* of such a function. Essentially, all neural network layers more complex than the MLP are designed to make learning easier, and different types of layers guide the learning process in different ways.

5.2 Layers

5.2.1 Embedding layer

Since we are working with character-level models, the input to a model consists of historical word forms represented as sequences of characters. In mathematical terms, the simplest way to represent a character $c \in \Sigma$ is as a *one-hot vector*, i.e., a vector $v \in \mathbb{R}^{|\Sigma|}$ that has a value of 1 on one dimension assigned to that character, and 0 everywhere else.

An *embedding layer* maps the one-hot representation of a character to a vector space with dimensionality d_m. It is essentially a lookup table assigning each input value a dense vector representation. Mathematically, it performs a simple dot product:

$$f(v) = v \cdot W_m \tag{5.5}$$

The matrix $W_m \in \mathbb{R}^{|\Sigma| \times d_m}$ stores the embedded vectors for each input character. Eq. (5.6) shows an example input and output of the embedding layer:

$$v = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \quad \mapsto \quad f(v) = \begin{pmatrix} 0.57 \\ -0.71 \\ \vdots \\ 0.01 \\ -1.72 \end{pmatrix} \tag{5.6}$$

The main purpose of this operation is to capture generalizations: in a one-hot representation, each possible input value is distinct from any other value, while in an $\mathbb{R}^{d_m}$ vector space, distances and similarities between input values can vary. In the normalization setting, input characters that behave similarly in terms of their normalization—e.g., $\langle s \rangle$ and $\langle f \rangle$ both often mapping to $\langle s \rangle$—should ideally be mapped to dense vectors that are close together in the embedding space.

The size d_m of the embedding space is a hyperparameter that can be freely chosen. Normally, we would expect it to be considerably smaller than the size of the alphabet Σ to encourage the model to learn generalizations. Sec. 6.2 will discuss this in more detail.

In practice, the embedding layer can either be initialized randomly and trained jointly with the rest of the model, or initialized with pre-trained embeddings instead. For the latter approach, it is common to train a language model that uses embeddings (e.g., Kim et al., 2016), then re-using these learned embeddings for a different NLP task. For many languages, such pre-trained embeddings are also available for download.[4] However, since we are dealing with historical languages, which are unlikely to have pre-trained embeddings,[5] and embeddings of characters are considerably less complex to learn than word embeddings (due to the size of the alphabet

[4] E.g., https://github.com/minimaxir/char-embeddings

[5] Although word embeddings for some historical languages do exist: https://nlp.stanford.edu/projects/histwords/

being considerably smaller than the pool of all possible word forms), no pre-trained embeddings are used here.

5.2.2 Dense layer

A *densely-connected layer*, or just *dense layer* for short, is a common type of neural network layer that simply performs a weighted multiplication of its inputs followed by an activation function g:

$$f(x) = g(x \cdot W + b) \tag{5.7}$$

If $x \in \mathbb{R}^{d_x}$ is the input vector, then $W \in \mathbb{R}^{d_x \times d_y}$ is the weight matrix and $b \in \mathbb{R}^{d_y}$ is a bias vector, where d_x is the dimension of the input and d_y the dimension of the output space. The layer is called "densely connected" because each component of the input vector contributes to each component of the output vector. This is identical to the hidden layer of an MLP (cf. Eq. (5.4) and Fig. 5.2). The dimension d_y is also sometimes referred to as the *number of units* in a layer, as the layer essentially performs one simple perceptron calculation per dimension.

In the normalization models, dense layers are most often used as the last layer in a model to predict the most likely output character. In this function, their purpose is to transform their input vector (which can be of arbitrary size) to a probability distribution over the set of all possible target characters. To achieve this, the output dimension is set to the size of the target alphabet (i.e., $d_y = |\Sigma_t|$), and the *softmax* function is used as the activation function:

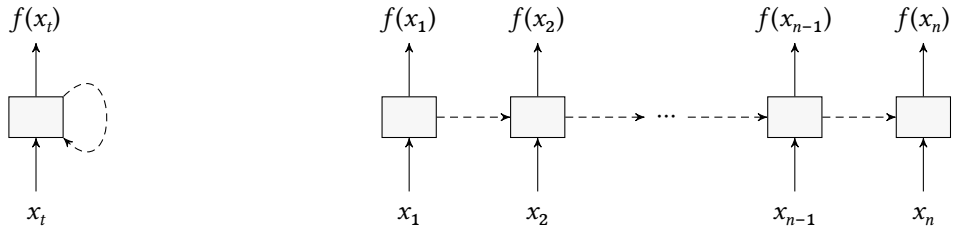
$$g(x) = \text{softmax}(x) = \frac{\exp x}{\sum_{i=1}^{m} \exp x_i} \tag{5.8}$$

Applying the exponential function ensures that all values are positive, and normalizing them by the sum of all values ensures that they sum to 1, thereby constituting a valid probability distribution.

5.2.3 Recurrent layers

In many NLP tasks, including our normalization task, the input to a model is not a single data point, but a sequence of data points, such as a sequence of words forming a sentence, or (as in our case) a sequence of characters forming a word. These data points are interdependent, and the order they appear in provides crucial information. *Recurrent layers*, or *recurrent neural networks (RNNs)*, aim to capture these sequential relationships of the input. For a comprehensive overview, see Goldberg (2017, chapters 14 and 15).

Many variants of RNNs exist. Commonly, they operate by using a *state vector* (or *hidden state*) that is carried over between the elements of the input sequence. The sequential dimension of the input is often referred to as the *time dimension*, with individual positions in the sequence being referred to as *timesteps* of the data.



(a) Compact (b) Unrolled

Figure 5.3: Two representations of the same recurrent neural network (RNN)

In its simplest instantiation, an RNN is identical to the dense layer in Eq. (5.7) with the addition of an internal state vector that carries over between timesteps (Elman, 1990):

$$h_0 = \vec{0} \tag{5.9}$$

$$h_t = g(x_t \cdot W + h_{t-1} \cdot U + b) \tag{5.10}$$

$$f(x_t) = h_t \tag{5.11}$$

Here, the input is assumed to be a sequence of vectors $x = \{x_1, \dots, x_n\}$, where each $x_t \in \mathbb{R}^{d_x}$. Consequently, the learnable parameters of the layer are a weight matrix $W \in \mathbb{R}^{d_x \times d_h}$ that transforms the input vector, a weight matrix $U \in \mathbb{R}^{d_h \times d_h}$ that transforms the hidden state of the previous timestep (h_{t-1}), and a bias vector $b \in \mathbb{R}^{d_h}$. The output of this simple RNN is identical to its hidden state; depending on the task, either the full sequence of hidden states $h_1, \dots, h_n$ can be used (e.g., for a sequence labeling task) or only the last timestep h_n (to obtain a single vector for the entire sequence).

Figure 5.3a shows a common visualization of an RNN layer; the looping arrow is intended to represent the recurrent connection, i.e., the hidden state vector that is carried over to the next timestep. When the number of timesteps is finite, the loop can also be *unrolled* to show the model over the full sequence, as in Figure 5.3b. Here, the unit at each timestep is a copy of the same RNN layer with identical parameters. The latter visualization will be preferred from now on.

Long short-term memory layers

In principle, the RNN architecture presented above allows to capture any kind of input–output dependencies as long as the output at timestep t only depends on information from input timesteps $1, \dots, t$. This is because the model is able to transfer information from any timestep to any following timestep via its hidden state vector. In practice, actually getting the model to *learn* these dependencies becomes exceedingly difficult as the distances grow—this is a fundamental problem of the common backpropagation algorithm (cf. Sec. 5.3) for training neural networks (Bengio et al., 1994).

The long short-term memory (LSTM) is a type of RNN intended to address this issue (Hochreiter and Schmidhuber, 1997). While it is not the only type of layer that has been proposed to solve the problem of long-term dependencies, it is certainly the most widely used and has been

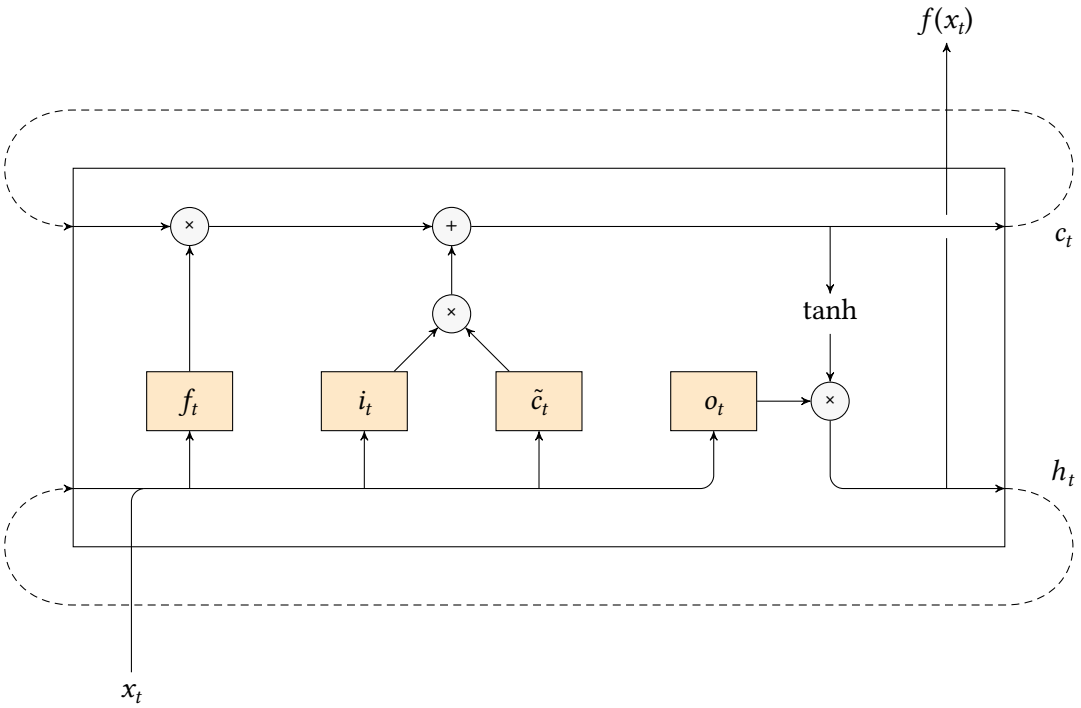


Figure 5.4: A long short-term memory (LSTM) network; rectangular cells represent internal dense layers, while circular cells represent pointwise operations; illustration adapted from Olah (2015).

proven to be successful for many NLP tasks (Goldberg, 2017, p. 181; or cf. Sutskever et al., 2014; Lample et al., 2016; Plank et al., 2016, and many more).

The LSTM is different from the simple RNN introduced above by keeping two internal states:

1. the *hidden state* h_t, which is combined with the input to serve as the basis for all further calculations; and
2. the *cell state* c_t, which is designed to carry information over long stretches of time and therefore functions as a form of "memory" for the LSTM.

The calculations performed by the LSTM are in essence just an elaborate combination of densely-connected neural network layers. In total, an LSTM unit is made up of four different of these layers, which all receive as input both the previous hidden state h_{t-1} and the LSTM's current input x_t:

- a *forget gate* f_t that determines how much and which parts of the memory cell state to keep (or to forget);
- an *update layer* that produces a new candidate cell state $\tilde{c}_t$;
- an *input gate* i_t that determines how much and which parts of the candidate cell state to integrate into the current cell state; and
- an *output gate* o_t that determines how much and which parts of the new cell state to copy into the new hidden state and, therefore, the cell's output.

Figure 5.4 illustrates how these layers are combined to form an LSTM unit. In mathematical terms, they are defined as follows:

$$f_t = \sigma(x_t \cdot W_f + h_{t-1} \cdot U_f + b_f) \tag{5.12}$$
$$\tilde{c}_t = \tanh(x_t \cdot W_c + h_{t-1} \cdot U_c + b_c) \tag{5.13}$$
$$i_t = \sigma(x_t \cdot W_i + h_{t-1} \cdot U_i + b_i) \tag{5.14}$$
$$o_t = \sigma(x_t \cdot W_o + h_{t-1} \cdot U_o + b_o) \tag{5.15}$$

As before, $W_* \in \mathbb{R}^{d_x \times d_h}$ and $U_* \in \mathbb{R}^{d_h \times d_h}$ denote weight matrices for the inputs and hidden states, respectively, while the $b_* \in \mathbb{R}^{d_h}$ denote bias vectors. Notice that the equations (5.12)–(5.15) each describe a densely-connected layer, identical to that defined in Eq. (5.7) if the input is defined as the concatenation of x_t and h_{t-1}. Furthermore, the input, output, and forget gates use the sigmoid (σ) activation function, while the update layer that calculates $\tilde{c}_t$ uses the hyperbolic tangent (tanh).

With these layers, the internal states of the LSTM are then calculated as follows:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{5.16}$$
$$h_t = o_t \cdot \tanh(c_t) \tag{5.17}$$

As before, the internal states are initialized to zero ($c_t = h_t = \vec{0}$), and the output of the LSTM is identical to its hidden state ($f(x_t) = h_t$).

For simplicity's sake, LSTM units will be visualized the same way as the simple RNN units in Figure 5.3b, except that each rectangular unit now represents an LSTM cell as depicted in Figure 5.4.

Bi-directional RNNs

So far, all recurrent layers processed their input timesteps from left to right. As a consequence, information derived from an input timestep can only be propagated to future timesteps, but never to previous ones. However, it is conceivable that the normalization of a character depends on information from both its left and right context—the same is true for many other NLP tasks.

A common solution in this situation is to use a *bi-directional RNN* (Schuster and Paliwal, 1997; Graves and Schmidhuber, 2005). This is simply a combination of two independent recurrent layers: a *forward layer* which reads the input sequence from left to right, and a *backward layer* which reads the input sequence from right to left.[6] The output of the bi-directional layer is constructed as the concatenation of the individual RNN outputs:

$$f(x_t) = (\text{RNN}_{\text{forward}}(x_t), \text{RNN}_{\text{backward}}(x_t)) \tag{5.18}$$

[6] Alternatively, the same recurrent layer can be used for both forward and backward passes, but this is not done here.

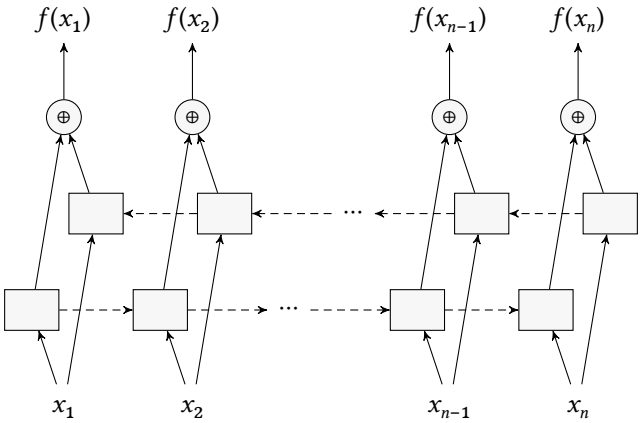


Figure 5.5: A bi-directional recurrent neural network (RNN)

Figure 5.5 illustrates this approach. If only a single output vector is desired, the RNN's outputs after reading the whole input sequence are used—this will be the last element of x for the forward layer, but the first element of x for the backward layer:

$$f(x) = (\text{RNN}_{\text{forward}}(x_n), \text{RNN}_{\text{backward}}(x_0)) \tag{5.19}$$

For uni-directional recurrent layers, the dimensionality d_h of their hidden state is also the dimensionality of the output vector. For bi-directional layers, the output vector will have the size of the combined forward and backward layers's hidden states. As a convention, a "bi-directional RNN with hidden dimensionality d_h" will refer to a composition of forward and backward RNNs with $\frac{1}{2}d_h$ dimensions for their hidden states, and only even numbers for d_h will be used in this case.

Bi-directional RNNs are commonly used for many NLP tasks, including dependency parsing (Kiperwasser and Goldberg, 2016) or sequence tagging tasks such as POS tagging or named entity recognition (Huang et al., 2015; Lample et al., 2016).

5.3 Training

The previous sections discussed the components of the neural network models and what their trainable parameters are, but have not touched upon the question how to actually train them, i.e., how to find appropriate values for their weight matrices and bias vectors to produce the desired outputs. This section summarizes all relevant aspects of the training process.

5.3.1 Objective function

In Sec. 5.1, we established that the models are trained in a supervised manner using pairs of historical tokens and their gold-standard normalization. Additionally, we need an *objective function* (or *loss function*) that quantifies how well the model matches the training data. Since the goal is to predict normalized characters, a natural choice for the objective function is the

categorical cross-entropy loss, also called *negative log-likelihood loss* (Goldberg, 2017, p. 27). Let y_i be the gold-standard probability of outputting the i-th character from our alphabet (of size m), and $\hat{y}_i$ the probability predicted by the model, then the cross-entropy $L(\hat{y}, y)$ is defined as:

$$L(\hat{y}, y) = -\sum_{i=1}^{m} y_i \log \hat{y}_i \tag{5.20}$$

The goal of the training process is then to find a set of weights for the neural network layers that minimizes the average cross-entropy loss over all training samples. A prerequisite for using this objective function is that the neural network outputs a probability distribution over all possible characters, which is achieved by using the softmax function on the final layer (cf. Sec. 5.2.2).

5.3.2 Optimizer

Training the layer weights is done using gradient-based optimization with backpropagation through time, the most common type of algorithm to train neural networks with recurrent layers. For details of the training procedure and the concepts behind it, see Goodfellow et al. (2016, chapters 8 and 10.2.2) or Goldberg (2017, chapters 2.8 and 5). The general idea is to compute the gradients of the network parameters—i.e., the weight matrices and bias vectors—with respect to the loss function L over the training set, and stochastically update the parameters based on these gradients (Goldberg, 2017, p. 30 f.).

An *optimization algorithm* specifies how exactly this stochastic parameter update is performed; the most basic algorithm is *stochastic gradient descent (SGD)*. Here, I will only be using the *Adam* optimization algorithm instead (Kingma and Ba, 2014). While there is evidence that stochastic gradient descent can achieve better generalization (Wilson et al., 2017), Adam is used here because I found it to be considerably less sensitive to its hyperparameters settings, making it easier to obtain good results across different configurations.[7]

5.3.3 Batch size

Neural networks can often take a long time to train. A common optimization is to feed training data into the network in *batches*; i.e., instead of calculating the training error and updating the model's parameters after each training sample, these updates are only performed after seeing m training samples, where m is also called the *batch size*.

This method speeds up the training process since gradient computations are performed less often and the hardware capabilities for parallelization can be utilized more effectively. However, since model updates happen less frequently, more passes over the training data can be needed

[7] I found that stochastic gradient descent often showed erratic behavior, with training accuracy first improving, then degrading again or even dropping to zero. Of course, this could be mitigated with careful hyperparameter tuning. The Adam optimizer also performed differently based on its hyperparameters, but almost never showed this pathological behavior. Since I am evaluating on a variety of different configurations and datasets, an extensive tuning of learning parameters for all configurations did not seem feasible, and Adam appeared to be the better choice overall due to its more stable performance.

for the model parameters to converge. Essentially, choosing a batch size is a trade-off between these two factors and mainly affects the total training time of a model (Bengio, 2012).

All experiments in this work are run with a batch size of 50, which I found to perform well with the normalization models.

5.3.4 Randomization of samples and initial weights

When training neural networks, the order in which samples from the training data are processed can have an effect on the learning process and the final state of the model. For this reason, a common technique is to shuffle the training samples before each epoch. This is done automatically by the Keras library.

Another important factor in training neural networks is how the parameters—i.e., the weight matrices—are initialized before training. Due to the way the networks are trained, they cannot simply be initialized to zero or any other constant value, as that would cause all gradients—and, therefore, the weight updates—to be either zero or be identical across all dimensions of the weight matrix. Random or probabilistic initialization of weights is key to successful training of neural networks (Goodfellow et al., 2016, Sec. 8.4). In this work, I always use the default initialization strategies as defined by Keras 1.2.2.[8]

The random shuffling of training samples and the probabilistic nature of the weight initialization means that different restarts of the same training procedure will yield different results. To make results deterministic, the random number generator is always initialized with a fixed seed—also determined randomly—before each training run, and in each group of experiments, the same fixed seed is used for all runs unless explicitly stated otherwise.

5.3.5 Dropout

Dropout is a common technique to prevent neural networks from overfitting on the training data (Srivastava et al., 2014). Its main idea is to randomly drop a fraction of the inputs during training—i.e., set some elements of the input vectors to zero—in order to make the model less reliant on having the full information from the training data. A *dropout rate* (or simply *dropout*) of 0.2, for example, means that 20% of each input vector is randomly set to zero during training.

The Keras implementation of dropout on recurrent layers follows Gal and Ghahramani (2016). Dropout is used both on the inputs and the recurrent connections of each RNN layer, although different dropout masks are used for each. However, the same masks are used for every timestep. In other words, the same randomly generated dropout mask is used for all inputs to all recurrent layers, while another randomly generated mask is used for all recurrent connections of all recurrent layers.

[8] cf. https://keras.io/initializers/

5.3.6 Early stopping

Another technique to prevent overfitting is *early stopping*. Typically, a model is trained for a certain, pre-defined number of epochs, where an *epoch* refers to a full pass over the training set. Training for too few epochs can result in suboptimal performance because the model has not converged to a good state yet; training for too many epochs can result in overfitting, where the model performance continues to improve on the training set, but starts to decline on a separate test or validation set.

Early stopping monitors model performance on a separate validation set after every epoch, and stops the training process if the validation error no longer significantly improves. More precisely, if $\bar{L}_i$ is the average cross-entropy loss (cf. Sec. 5.3.1) on the validation set after epoch i, then the criterion for early stopping is $\bar{L}_{i-1} - \bar{L}_i < 0.001$. The validation error does not always improve in a strictly monotonic fashion; therefore, training is only stopped when the criterion is met for the second time[9] during the training process. Additionally, a snapshot of the model's state is saved after every epoch, and only the state with the best validation accuracy is kept after training ends.

[9] For experiments with smaller datasets of less than 10,000 tokens, a larger number of epochs (up to a maximum of 10) without improvement is allowed before training stops.

Chapter 6

Encoder–decoder model

We have seen that the normalization task can be modeled using machine translation techniques applied on a character level: Sec. 4.1.4, for example, discussed previous work on normalization using statistical machine translation. For machine translation using neural networks, a popular framework is the *encoder–decoder architecture* (Cho, Merrienboer, Gülçehre, et al., 2014; Sutskever et al., 2014). In this chapter, I will analyze the suitability of encoder–decoder models for historical text normalization.

Figure 6.1 shows the basic concept of any encoder–decoder model: the input—in our case, a historical word form—is fed into the *encoder*, which produces a *vector representation* of this input; i.e., it *encodes* that input into a single vector of fixed (but potentially very high) dimensionality. This vector is fed into the *decoder*, which then produces the normalized word form; i.e., it *decodes* the supplied vector into the desired output. For NLP tasks, encoders and decoders typically take the form of RNNs (as in Cho, Merrienboer, Gülçehre, et al., 2014), but in principle, any type of network can be used, such as convolutional networks (e.g., Kalchbrenner and Blunsom, 2013; Gehring et al., 2016) or hybrid networks consisting of both convolutional and recurrent components (e.g., Vosoughi et al., 2016).

Sec. 6.1 describes the models used in this work in more detail. Sec. 6.2 discusses the hyperparameter settings for these models in the context of normalization, and Sec. 6.3 analyses and compares the different models.

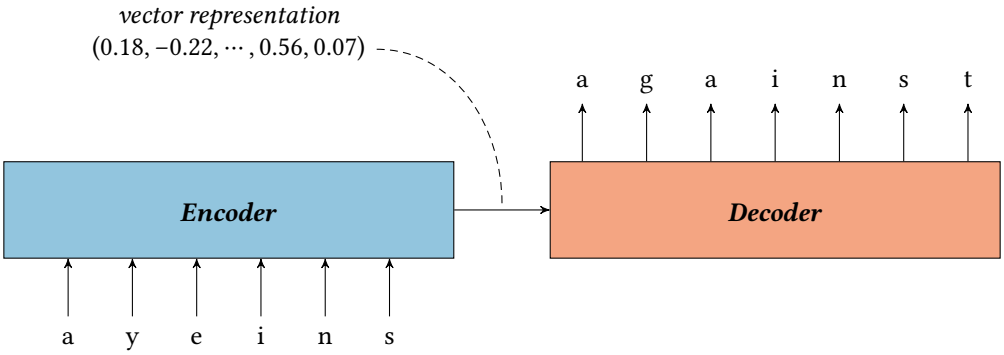


Figure 6.1: Basic encoder–decoder architecture for normalization

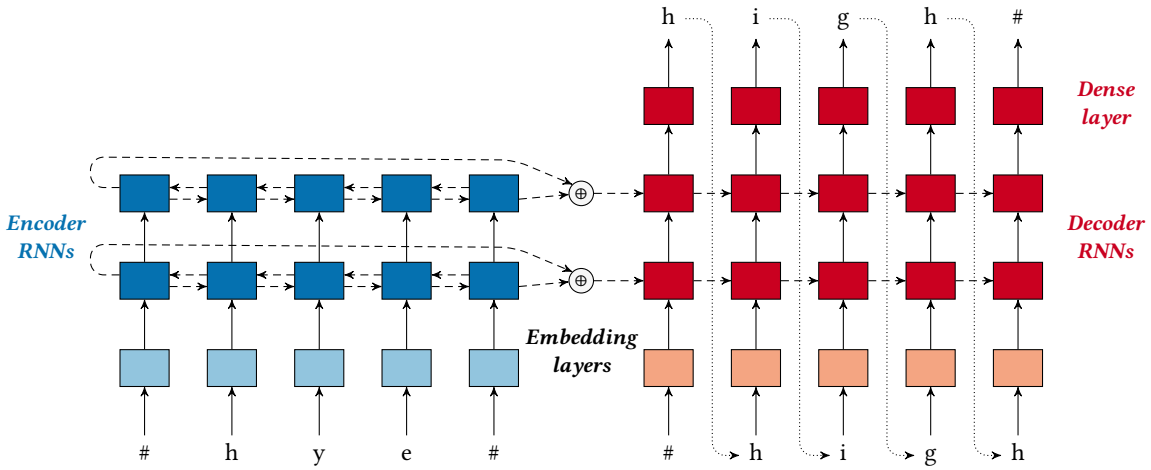


Figure 6.2: Encoder–decoder model with a stack of two bi-directional RNNs for the encoder (left) and a stack of two uni-directional RNNs for the decoder (right)

6.1 Model description

The base model investigated here uses LSTMs for its encoder and decoder (Sec. 6.1.1). Additionally, a variant of this model using an attention mechanism is tested (Sec. 6.1.2).

6.1.1 Base model

Figure 6.2 shows the base model used in the experiments. It is made up of these main components: (i) an encoder consisting of a stack of bi-directional LSTMs (bi-LSTM; cf. Sec. 5.2.3); (ii) a decoder consisting of a stack of uni-directional LSTMs followed by a dense prediction layer (cf. Sec. 5.2.2); and (iii) embedding layers for both the encoder's and the decoder's inputs (cf. Sec. 5.2.1).

This model is almost identical to that proposed by Sutskever et al. (2014) for neural machine translation and used by Bollmann et al. (2017) for historical text normalization, with the main exception of always using bi-directional layers in the encoder.

The encoder can consist of any number of LSTM units; if more than one LSTM layer is used, the outputs of one layer are fed as inputs into the next layer. Bi-directional LSTMs are chosen to allow dependencies to flow in both directions. Sutskever et al. (2014) note that in their setup (using uni-directional LSTMs), reversing the input sequences was beneficial, which they believe to be the result of shorter dependencies within the model—the first elements of the input sequence being closer to the first elements of the target sequence in the decoder. With bi-directional layers, both ends of the input sequence are connected directly to the decoder (cf. Fig. 6.2).

The internal states of the n-th bi-directional LSTM after reading the full input sequence—i.e., the last character for the forward layer and the first character for the backward layer—are concatenated and used to initialize the internal states (h_0, c_0) of the n-th decoder LSTM. As a consequence, the encoder and decoder will always use the same number of recurrent layers.

The input to the encoder is a historical word form as a sequence of characters, padded on each end by a special "word separator" symbol. The output of the decoder is the normalized word form as a sequence of characters, with an added "end of word" symbol. During decoding, this symbol is used to determine when to stop generating more characters, which allows the decoder to produce output sequences of arbitrary lengths.

Additionally, the decoder uses *input feeding*: the input to the decoder at timestep t is the predicted character at timestep $t - 1$. For the first decoder timestep, a special "start symbol" is used. This method explicitly conditions the decoder's output not only on its current internal states, but also on the previously generated character. Besides making the model aware of its predictions at previous timesteps, this also allows us to influence the decoding process via beam search or filtering (cf. Sec. 6.1.3). At the same time, this is the main reason why the decoder uses uni-directional LSTMs only, since the input feeding introduces left-to-right dependencies that are hard to reconcile with bi-directional layers.

Training is done using categorical cross-entropy on the individual decoder outputs (cf. Sec. 5.3.1) and *teacher forcing*, which means that the gold-standard predictions are always fed into the decoder inputs, regardless of whether the decoder has already learned to correctly predict them. Only at test time are the actual model predictions fed into the next decoder timestep. While this is a common approach for this type of model, it is not without drawbacks, as the loss function does not take global accuracy into account, and the teacher forcing potentially introduces a discrepancy of the decoder inputs between training and test time. These issues can be addressed with more sophisticated training methods (e.g., Bengio et al., 2015; Wiseman and Rush, 2016), but these will not be considered here.

Finally, while the input and output sequences can theoretically be of arbitrary lengths, in my experiments I introduce a *length cutoff* to reduce the runtime for training and decoding. The maximum input length for the encoder is set to 22 characters, while the maximum output length for the decoder is 20 characters. These limits were set so that they affect less than 0.02% of samples from any of the datasets.[1] When training the model, word pairs above these limits are simply discarded; for evaluation, they are included so as to not distort the results.[2]

6.1.2 Attentional model

The encoder–decoder models presented above all have a common bottleneck, which is the vector representation passed from the encoder to the decoder (cf. Fig.6.1): the input sequence can be of arbitrary length, while its encoded vector representation will always be of the same fixed size. For machine translation, Cho, Merrienboer, Bahdanau, et al. (2014) show that the performance of such an encoder–decoder model degrades as the input sequences get longer.

The *attention mechanism* was introduced to address this issue (Bahdanau et al., 2014). Instead of generating a single vector to represent the whole input sequence, the encoder generates a vector for each timestep, and a connection is introduced at each decoder timestep that passes on

[1] For datasets other than Swedish, it is actually less than 0.01% of samples.

[2] The historical input word forms will be truncated in these cases, with the most likely consequence that the model will produce an incorrect normalization.

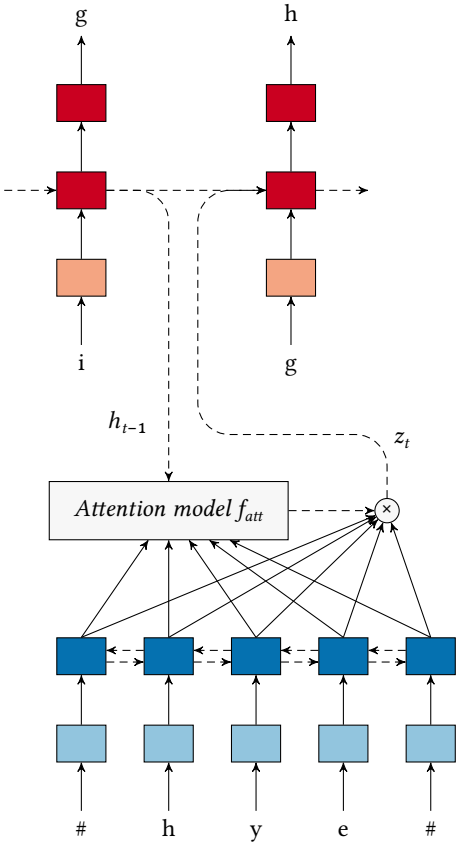


Figure 6.3: Encoder–decoder model with attention mechanism

a weighted combination of these encoded vectors. Intuitively, this allows the decoder to focus on different parts of the encoded input sequence at different times of the decoding process.

Figure 6.3 shows an illustration of the attention mechanism. At its core is the *attention model*, which takes the decoder's current hidden state and the encoder outputs to determine which parts of the sequence to focus on. Here, the attention model will simply be a multi-layer perceptron (cf. Sec. 5.1). The weighted combination of encoder outputs, also called the *context vector* (z_t), is then integrated into the hidden state of the decoder's next timestep.

Many different variants of attention have been proposed, e.g., using different functions for the attention model, conditioning only on parts of the encoder outputs or enforcing monotonicity constraints, or using the context vector as the decoder's input (see, e.g., Luong, Pham, et al., 2015; Cohn et al., 2016). The model used here closely follows those of Bahdanau et al. (2014) and Xu et al. (2015).[3] In particular, feeding the weighted encoder vectors into the decoder's hidden state—instead of simply using them as the decoder's input—allows us to still use the input feeding approach described above (cf. p. 87).

[3] I experimented with many other variations of the attention mechanism, including the use of different scoring functions such as the simple dot product or weighted multiplication (Luong, Pham, et al., 2015, Sec. 3.1) and local attention models with monotonic or predictive alignment (Luong, Pham, et al., 2015, Sec. 3.2). I could not find a clear advantage of any of these alternatives on a small development dataset, so I did not explore them further.

Technical description

Let f_{att} be the attention model and $r = r_0, \dots, r_n$ the outputs of the encoder. The context vector z_t is then calculated as a weighted combination of the encoder outputs r, where the weights α_i are determined by the softmax function applied to the output of the attention model f_{att}:

$$e_{t,i} = f_{\text{att}}(h_{t-1}, r_i) \tag{6.1}$$

$$z_t = \sum_{i=0}^{n} \alpha_{t,i} r_i \tag{6.2}$$

$$= \sum_{i=0}^{n} \text{softmax}(e_{t,i}) r_i \tag{6.3}$$

$$= \sum_{i=0}^{n} \frac{\exp e_{t,i}}{\sum_{j=1}^{n} \exp e_{t,j}} \; r_i \tag{6.4}$$

The attention model is a multi-layer perceptron (MLP) with trainable parameters $W_{\text{att}}, U_{\text{att}} \in \mathbb{R}^{d_h \times d_h}$, $b_{\text{att}}, v_{\text{att}} \in \mathbb{R}^{d_h}$:[4]

$$f_{\text{att}}(h_{t-1}, r_i) = \tanh(r_i \cdot W_{\text{att}} + h_{t-1} \cdot U_{\text{att}} + b_{\text{att}}) \cdot v_{\text{att}}) \tag{6.5}$$

The context vector z_t is fed back into the decoder's hidden state by essentially concatenating it to the previous timestep's hidden state—the LSTM equations from Eq. (5.12)–(5.15) are augmented by the dot product of z_t and its weight matrix $Z_* \in \mathbb{R}^{d_h \times d_h}$:

$$f_t = \sigma(x_t \cdot W_f + h_{t-1} \cdot U_f + z_t \cdot Z_f + b_f) \tag{6.6}$$

$$\tilde{c}_t = \tanh(x_t \cdot W_c + h_{t-1} \cdot U_c + z_t \cdot Z_c + b_c) \tag{6.7}$$

$$i_t = \sigma(x_t \cdot W_i + h_{t-1} \cdot U_i + z_t \cdot Z_i + b_i) \tag{6.8}$$

$$o_t = \sigma(x_t \cdot W_o + h_{t-1} \cdot U_o + z_t \cdot Z_o + b_o) \tag{6.9}$$

Similar to Xu et al. (2015), the initial hidden and cell states of the decoder LSTM are set as the output of a dense layer on the averages of the encoder outputs:

$$h_0 = \tanh\left(\left(\frac{1}{n}\sum_{i=0}^{n} r_i\right) \cdot W_{\text{init,h}} + b_{\text{init,h}}\right) \tag{6.10}$$

$$c_0 = \tanh\left(\left(\frac{1}{n}\sum_{i=0}^{n} r_i\right) \cdot W_{\text{init,c}} + b_{\text{init,c}}\right) \tag{6.11}$$

6.1.3 Decoding

In all models described above, each output character is fed back into the decoder as input for its next timestep. During training, the gold-standard output characters are known and can simply be set as the decoder's input (cf. p. 87). At test time, a decoding strategy must be used.

[4] Note that I assume that the dimensionality d_h of the hidden state is identical between the encoder and the decoder, which will be the case in my experiments, but is not a strict requirement.

The simplest strategy is to dynamically iterate over all timesteps and output the character with the highest probability as predicted by the model. This process continues until the special "end of word" symbol is predicted, at which point the decoding stops. This strategy is also known as *greedy decoding*.

One drawback of this approach is that it only considers the predicted probabilities locally for each timestep, but for evaluation, we are mostly interested in optimizing word accuracy, i.e., performance over the full output sequence. *Beam search decoding* can help to produce better global predictions by keeping a list of the n most likely candidate sequences, where n is also called the *beam size*. At each timestep, beam search considers the concatenation of each sequence already in the candidate list with all possible output characters, adds the probabilities of the output characters to the probability of the sequence, then filters the list to keep only the best n candidates. When the "end of word" symbol is predicted for a sequence, that sequence is no longer updated further. Decoding continues until all n best candidates have reached the "end of word" symbol, at which point the sequence with the highest probability is chosen as the output. For the experiments in this work, I always use a beam size of $n = 5$, as this was found to work reasonably well in preliminary experiments; this is also the same beam size chosen in Bollmann et al. (2017).

In addition to these decoding strategies, we can use *dictionary filtering* (or *lexical filtering*) to restrict the list of possible candidate sequences. This is inspired by the Norma tool (cf. Sec. 4.2.1), which constrains the outputs of its rule-based and distance-based normalizers to those contained within a list of valid contemporary word forms. This behavior can easily be replicated in the encoder–decoder model: during decoding, sequences which cannot lead to a valid target word form—i.e., sequences that do not occur as a word-initial substring in the contemporary dictionary resource—are filtered out of the list of candidates. Similarly, the "end of word" symbol can only be predicted if the generated word is contained within the dictionary.

6.2 Hyperparameter tuning

One of the more challenging aspects of training a neural network successfully is the selection of good hyperparameters. While the model descriptions above specify the general architecture, some details still need to be determined, such as the number of LSTMs to use for the encoder and decoder (cf. Fig. 6.2), the dimensionality d_h of the network layers (cf. Sec. 5.2), or the dropout rate to be used during training (cf. Sec. 5.3.5). These settings are usually determined before the training procedure starts, and are referred to as *hyperparameters* to distinguish them from the model's parameters (such as the weight matrices) that are fit to the training data.

In many studies, the selection of hyperparameters is done manually, for example by trying a few chosen combinations—often determined by previous work and/or the researcher's intuition—and picking the best one. This is partly due to the lack of definitive guidelines for hyperparameter selection, and partly due to the often prohibitively long training times of these networks. A good neural machine translation model, for example, can easily take days or

weeks to train even on comparatively powerful hardware.[5] Testing dozens or even hundreds of different hyperparameter combinations is not feasible in these situations.

The normalization models considered here, on the other hand, are comparatively fast to train. For the tuning experiments reported below, the average time of a training run was around thirty minutes; the exact times vary, of course, depending on the dataset and the hyperparameter configuration.[6] This enables us to perform a more extensive hyperparameter search than is typically done in the literature. An exhaustive search, however, is still not feasible: e.g., if five hyperparameters are to be tested with ten different values each, this would result in 10^5 different hyperparameter combinations—testing all of them would take almost six years in total! For this reason, I conduct these tuning experiments on a reduced selection of the historical datasets, and split up the tuning into two phases, considering hyperparameters of the model—i.e., number of layers and their dimensionality—and hyperparameters of the learning procedure—i.e., learning rate and dropout rate—separately.

Sec. 6.2.1 describes which datasets are used for hyperparameter tuning. Sec. 6.2.2 describes the tuning procedure. The tuning of the model (hyper)parameters is discussed in Sec. 6.2.3, while the learning parameters are discussed in Sec. 6.2.4.

6.2.1 Tuning datasets

One challenge in hyperparameter tuning is that properties of the dataset can conceivably affect the optimal parameter settings, and thus, using different datasets for tuning might result in different "optimal" configurations being found. To reduce the dataset bias and find a configuration that works reasonably well across many different datasets, it seems desirable to perform hyperparameter tuning on more than just a single dataset. On the other hand, tuning hyperparameters individually for *each* dataset is both computationally expensive and not representative for most application scenarios, where training data is scarce and separate tuning data not necessarily obtainable.

As a compromise, I perform hyperparameter tuning on only five of the ten historical datasets (cf. Sec. 3.1) and only use a subset of each dataset. More precisely, the following five datasets are chosen:

- English (Sec. 3.1.1), as its texts are among the oldest represented in this study (cf. Tab. 3.1);
- the German Anselm dataset (Sec. 3.1.2), as it has the highest HNR (cf. Tab. 3.2) and is also an outlier in the ambiguity score distribution in Fig. 3.1;
- Hungarian (Sec. 3.1.3), as it is the least similar to the other datasets based on the comparison in Fig. 3.2;
- Icelandic (Sec. 3.1.4), as it stands out the most in the ambiguity score distribution in Fig. 3.1; and

[5] For example, Google trains an English-to-French neural machine translation model in around 6 days, using 96 (sic!) NVIDIA Tesla K80 GPUs (Wu et al., 2016).

[6] This is on a single NVIDIA GeForce GTX 980 GPU. The shortest training run took 6 minutes, while the longest took 5 hours and 43 minutes.

- the Slovene Gaj dataset (Sec. 3.1.5), as it is the most "modern" dataset in terms of the time periods represented (cf. Tab. 3.1) and has a low quantity of spelling variation (cf. Tab. 3.3).

For each of these datasets, 50,000 tokens are randomly selected from the training set to serve as the reduced training set for hyperparameter tuning. The only exceptions are the German Anselm dataset, where a balanced selection is taken from the beginning of each of the texts it consists of (due to the biased nature of the dataset, discussed in Sec. 3.1.2), and Icelandic, where the full training set consisting of 49,633 tokens is chosen (cf. Tab. 3.1). For the subset used to evaluate the trained model, 5,000 tokens are selected from the development sets of the respective corpora in the same fashion.

This construction of tuning datasets has the advantage of keeping the runtime of the tuning process more manageable, as the computational cost of training the models is similar across the datasets, but lower than training on the full sets, which are up to 234,000 tokens in size. Additionally, as the ideal hyperparameter configuration might be sensitive to the size of the training set, keeping this size constant eliminates one potentially confounding factor from the comparison.

6.2.2 Tuning procedure

There are several strategies for finding a good configuration of hyperparameters, such as grid search or random search;[7] in this work, I am using the tree-structured Parzen estimator (TPE) algorithm (Bergstra et al., 2011) as implemented by the hyperopt library (Bergstra et al., 2013).[8] The general idea is to use knowledge of previous hyperparameter trials to try to predict which new set of hyperparameters is most likely to improve results further. At the start of the optimization process, hyperparameters are drawn randomly from a pre-defined probability distribution, and the model is trained and evaluated using this set of parameters. After a certain number of trials,[9] the TPE algorithm is used to select new parameters instead, which is likely to choose parameters close to those that produced good results in the previous runs.

Training the models is done using early stopping (cf. Sec. 5.3.6) by validating on the 5,000-token evaluation set (as described above) after each training epoch. The highest word accuracy among all epochs on this evaluation set is considered to be the final accuracy score for the hyperparameter trial (and reported back to the hyperopt library).

6.2.3 Model parameters

For the hyperparameters of the model itself, I consider these three different variables:

- the *size of the embedding layers*, i.e., the dimensionality d_m of the vector space $\mathbb{R}^{d_m}$ into which the input characters are projected (cf. Sec. 5.2.1);

[7] See http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html for an informal, but illustrative overview.

[8] http://hyperopt.github.io/hyperopt/

[9] Judging from the source code, it appears that hyperopt uses 20 random trials by default, but this is not clearly documented.

- the *depth* of the model, i.e., how many LSTM layers to stack in the encoder and decoder (cf. Fig. 6.2); and
- the *size of the hidden layers*, i.e., the dimensionality d_h of each LSTM layer in the encoder and decoder stack (cf. Sec. 5.2.3).

Increasing the dimensionality of layers increases the number of trainable parameters a model has. Too few parameters might make it hard or impossible for the model to fit the data in a good way. More parameters mean a higher model capacity, but also a longer training time and the potential for lower generalization—after all, in a hypothetical scenario with limitless capacity, a model could simply learn to memorize all training samples, while we would like it to be able to generalize beyond these previously seen examples.

Model parameters, such as the size of the embedding layer, are often selected empirically, and their values can vary greatly between different experimental setups:

> Unfortunately, there are no theoretical bounds or even established best-practices in this space. [...] In current research, the dimensionality of word-embedding vectors range between about 50 to a few hundreds, and, in some extreme cases, thousands. (Goldberg, 2017, p. 99)

For neural machine translation using words as input, Cho, Merrienboer, Gülçehre, et al. (2014) use a dimensionality of 500 for the embedding layer and a dimensionality of 1000 for the hidden layers, while Sutskever et al. (2014) and Luong, Pham, et al. (2015) use a size of 1000 for both. However, this is not necessarily a good guideline for our normalization models, as word-based models have a much larger vocabulary size than character-based ones—Sutskever et al. (2014) report using an input vocabulary of 160,000 words—and processing full sentences requires a higher capacity than processing single words.

For character-based machine translation, Ling et al. (2015) use an embedding size of 50 and a hidden layer size of 150, considerably lower than those of the word-based counterparts. In previous work on normalization, Bollmann et al. (2017) used a dimensionality of 128 for both embedding and hidden layers.

Some encoder–decoder approaches use only a single RNN/LSTM for their encoder and decoder (e.g., Bahdanau et al., 2014; Cho, Merrienboer, Gülçehre, et al., 2014; Bollmann et al., 2017). However, Sutskever et al. (2014) found that stacking several layers of LSTMs performed significantly better; they use a stack of four layers in their experiments.

For tuning the normalization models, I experiment with using either a single LSTM layer or stacks of two or three layers (i.e., depth $\in \{1, 2, 3\}$). The size of the embedding layer is varied between 10 and 80 (with a step size of 5), while the size of the hidden layers are tuned within the range of 20 to 500 (with a step size of 20). The initial probability distribution is uniform over these value ranges. For each dataset, a total of 100 trials with different parameter configurations (as suggested by the TPE algorithm) are run.

This approach to tuning the model hyperparameters is, again, not exhaustive. For example, there is no fundamental reason why the number of layers in the encoder should have to match the number of layers in the decoder; similarly, each of the different LSTM layers could have a

different dimensionality.[10] Bahdanau et al. (2014), for example, use 1000 units in their decoder and 1000 units for each of the forward and backward RNNs in their bi-directional encoder, corresponding to a dimensionality of 2000 according to the definition used here (cf. p. 81). However, tuning the layers separately adds more hyperparameters to the tuning procedure, exponentially increasing its computational cost, while it is unclear whether this is actually beneficial. Therefore, I choose the simpler approach here.

Base model

Figure 6.4 shows the results of the tuning trials on the base model as a function of each *individual* model hyperparameter; each point in the graph represents the result from one trial, and each column only analyzes a single hyperparameter, encompassing all possible values of the other two hyperparameters.

The general trends appear to be similar across all five datasets. For the size of the hidden layers, there is a sharp increase in accuracy as the layer size increases, up until a dimensionality of around 100, where the gain from increasing the layer size further seems to taper off. The size of the embedding layers shows very similar tendencies, although using very small values here (≤ 20) appears to hurt the overall accuracy less compared to the hidden layers. Interestingly, there is no clear tendency that increasing the dimensionality beyond a certain point hurts the model's accuracy; in all cases, higher values seem to result in comparable or slightly better accuracy scores. For some datasets, such as German and Icelandic, a slight downward trend can be observed for particularly high-dimensional hidden layers (> 400), but since this decrease is much smaller than the overall variance in the accuracy scores, it is unclear whether this is an actual effect or just noise. Testing with even higher layer sizes might shed some more light on this, but since the results do not suggest that this could result in a significant accuracy gain, I do not investigate this further.

The depth of the model appears to have only a minor impact on the best obtainable accuracy. In particular, models with only one LSTM layer are able to perform better or comparably well than those with two or three layers. This is somewhat surprising, since adding more layers increases the model's complexity considerably, both by significantly increasing its total number of parameters and by adding more nonlinear activation functions. These results suggest that the normalization task can already be modeled sufficiently well with single-layer encoders and decoders, and the increased capacities by stacking more layers are not really needed for this task.

So far, we only looked at the effects of each hyperparameter individually. This does not tell us which *combination* of hyperparameter values produced each given result, and neglects to consider the interaction between the hyperparameters. For example, it is conceivable that two-layer models perform best with very different hidden layer sizes than single-layer models, but these results are conflated when looking at the effect of the hidden layer size in Figure 6.4. Therefore, Figure 6.5 provides a different view on the data, plotting the exact combinations of hyperparameters that were tested and visualizing the model accuracy in three discrete

[10] For the non-attentional model, the decoder's hidden states are initialized with the output of the encoder, which requires the dimensionalities to match, but this could theoretically be overcome by adding another transformation between them. For the attentional model, no such restriction exists.

categories: "best" trials—defined as being within 0.1 percentage points (pp) of the best result—, "good" trials—within 1 percentage point of the best result—and all "other" trials.

Regarding the model depth, most results in the "good" and "best" categories are found in the single-layer models. A notable exception is the Hungarian dataset, which shows an equal preference for depths one and three;[11] i.e., stacking more layers is not detrimental here, but also not clearly advantageous to a single layer. For Slovene, results in the "good" category are spread out more across different model depths, but most "best" results are again found in single-layer configurations. In all cases, there does not seem to be a clear advantage of using more than one LSTM layer, reaffirming the impression that using a single layer is sufficient for the normalization task.

Within the group of trials using a depth of one, the best results are achieved with a hidden layer size $d_h \geq 300$. Some datasets—like English and Hungarian—seem to work better with even higher dimensionalities of up to 500, the maximum that was tested, while others—such as Icelandic and Slovene—also work well with slightly smaller values. In general, choosing $d_h = 300$ appears to be a good compromise between model size and validation accuracy, which is the value I will use for further experiments.

The embedding size shows considerably more variation. On English and Icelandic, any dimensionality $d_m \geq 20$ seems to work equally well, and a similar tendency can be observed for Slovene and $d_m \geq 35$. For German and Hungarian, on the other hand, the "good" and "best" results are concentrated on values of $d_m \geq 60$. Since those datasets show worse performance with embeddings of lower dimensionality, but no datasets perform generally worse with high-dimensional embeddings, the size of the embedding layers will be set to $d_m = 60$ for further experiments.

Attentional model

The encoder–decoder model with attention (cf. Sec. 6.1.2) builds on the base model by changing the way the encoder's hidden states are integrated into the decoder. In particular, the encoder now produces one output vector per timestep (i.e., per character in the historical word form). This means that the information encoded in these hidden vectors is now potentially very different, as they do not have to capture the full word form, but can focus on more local information. Due to this, it is conceivable that much smaller sized vectors are sufficient or maybe advantageous in attentional models.

Since running the hyperparameter trials is computationally expensive and the models are otherwise very similar, I choose not to repeat the full trial runs with the attentional model, but rather focus on varying the hidden layer size only. Since concentrating on a single hyperparameter reduces the search space significantly, I decide to perform these trials in a deterministic manner

[11] The best results are achieved for high embedding and hidden layer sizes, i.e., parameter combinations in the upper right area of the plot. Very few such configurations have been tried in the case of Hungarian and depth = 2. It is possible that if more configurations with high layer sizes had been tried with two-layer models (e.g., by running more trials or having a different random selection of initial trials), these models with depth = 2 would have had a similar share of "good" results compared to depths one and three.

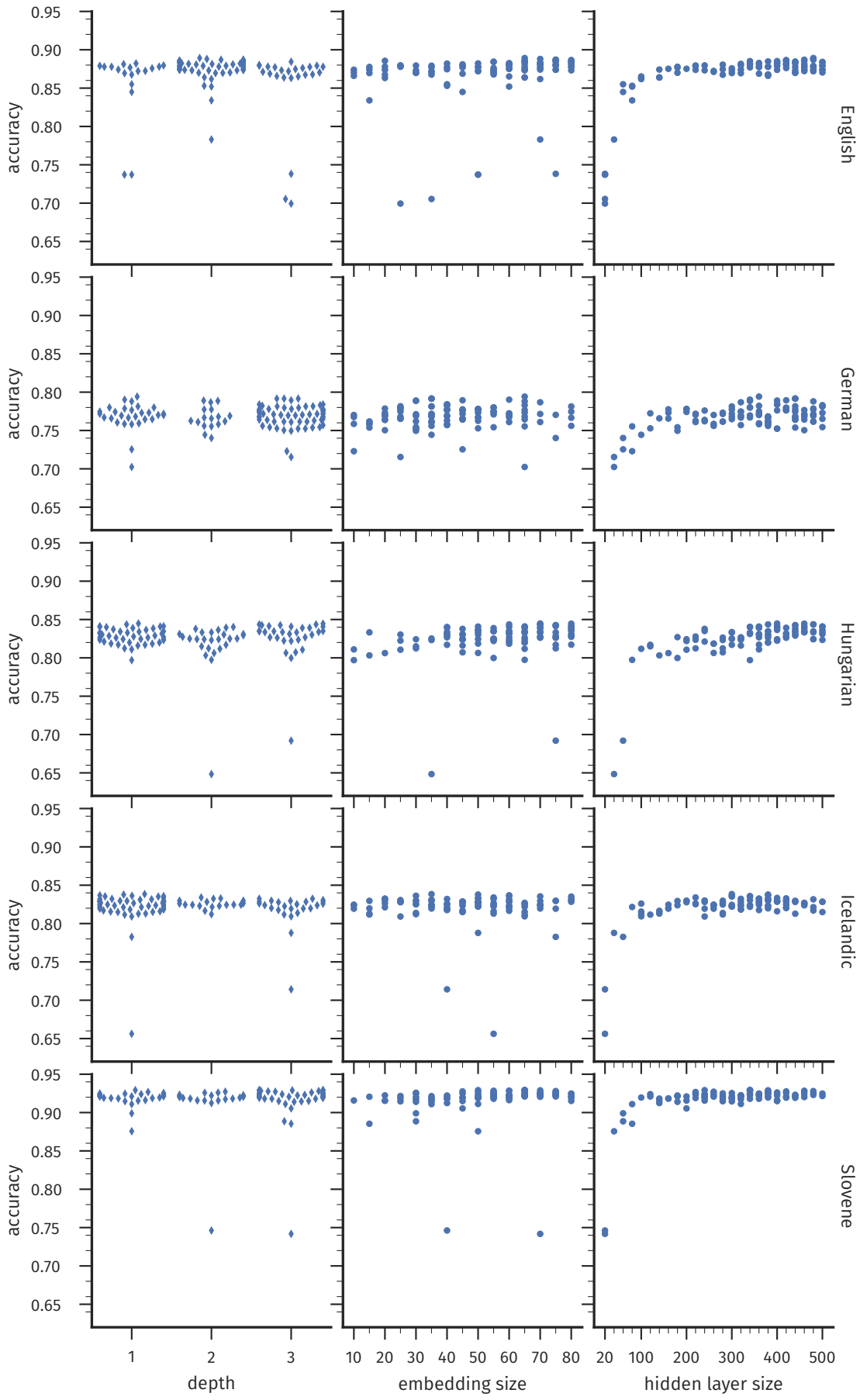


Figure 6.4: Accuracy of the base encoder–decoder model on the development sets as a function of each individual model hyperparameter

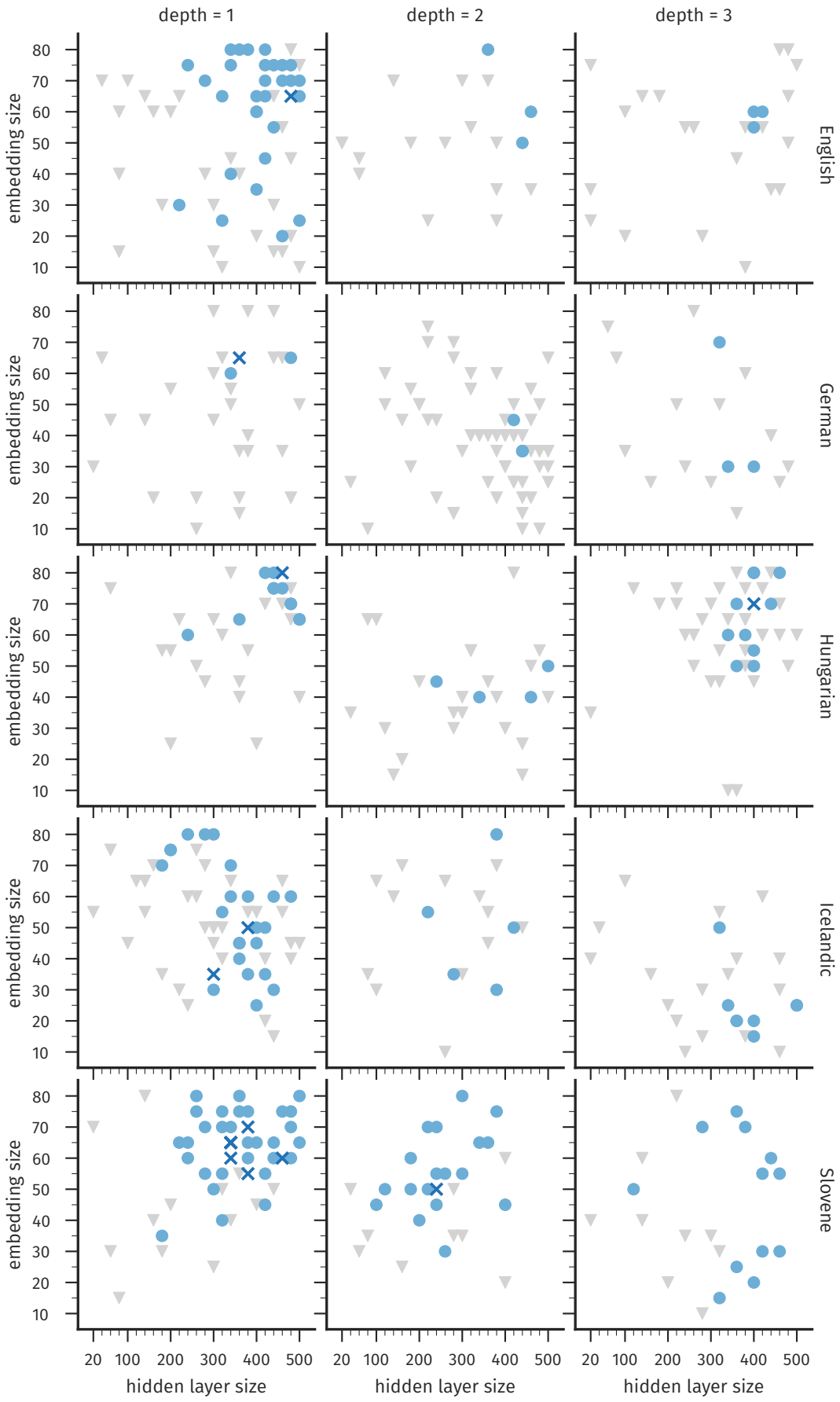


Figure 6.5: Combinations of model hyperparameter values categorized by the distance of their accuracy a_i to the dataset's overall best accuracy $a_{\max}$; "best" ×: $|a_{\max} - a_i| \leq 0.001$, "good" •: $0.001 < |a_{\max} - a_i| \leq 0.01$, "other" : $|a_{\max} - a_i| > 0.01$.

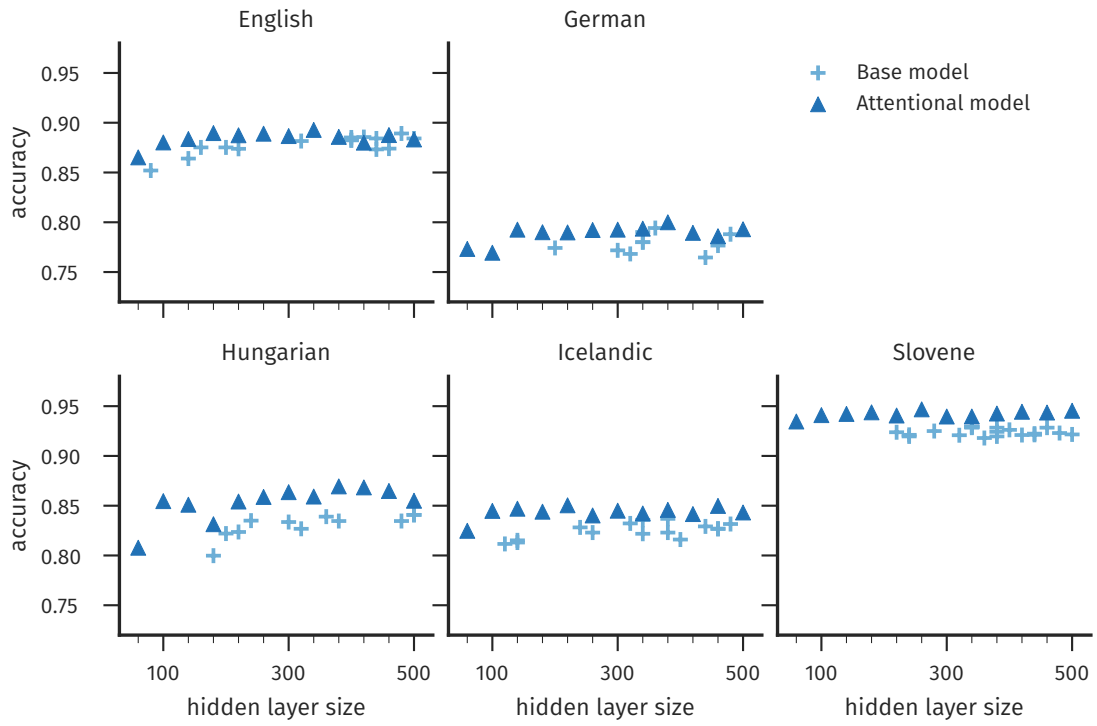


Figure 6.6: Accuracy of the attentional encoder–decoder model on the development sets, compared to that of the base model trials with similar hyperparameters (depth = 1, embedding size $d_m \in \{55, 60, 65\}$).

by testing all hidden layer sizes in the range $[60, 500]$ with a step size of 40.[12] The model depth is fixed to one, while the size of the embedding layers is fixed to 60, as determined by the hyperopt trials for the base model above.

Figure 6.6 shows the results of those trial runs in comparison to base model trials with similar hyperparameters; since not many trial runs of the base model used an embedding size of exactly 60, those with $d_m = 55$ and $d_m = 65$ are also included in the plot. The results suggest that very small values of the hidden layer size d_h can perform worse, but otherwise the accuracy scores are relatively steady across different values of d_h. While it appears that slightly smaller values (around 200) might be sufficient here, there is also no strong argument against reusing the value $d_h = 300$ that was chosen for the base model, which I adopt here for simplicity's sake. Additionally, the results show that the attentional model outperforms the base model on average—with the possible exception of the English dataset—justifying the use of the attention mechanism.

However, there is one drawback of the experimental setup that needs to be considered when interpreting these results: I only vary the size of all hidden layers simultaneously, i.e., changing d_h means changing the hidden layer sizes in both the encoder and decoder at the same time. While the encoder is used differently in the attentional model (as explained above), there is no clear reason why the decoder should also profit from a reduced dimensionality in this

[12] 60 is chosen as the lower bound (compared to 20 in the previous runs) as that is the size of the embedding vectors as determined by the previous hyperopt trials, and it did not seem sensible to make the hidden layers smaller than the embedding layers that feed into them.

model. In other words, a more thorough investigation of the effect of hidden layer sizes on the performance of attentional models should probably consider d_h independently for the encoder and the decoder. I have not done this here, but rather only focussed on the case of identical values for all layers since it is conceptually simpler.

6.2.4 Learning parameters

For the hyperparameters of the learning algorithm, I consider two different variables:

- the *dropout rate*, i.e., the fraction of the layer inputs that is randomly set to zero during training (cf. Sec. 5.3.5); and
- the *learning rate* of the Adam optimization algorithm, called "stepsize" in Kingma and Ba (2014), i.e., an approximate bound for how large the parameter updates are (cf. Sec. 5.3.2).

Dropout is a common regularization technique supposed to prevent overfitting, with many deep learning models using some amount of dropout in their training; e.g., Zaremba et al. (2014) experiment with dropout rates of 0.5 and 0.65, and Bollmann et al. (2017) use a rate of 0.3. In the tuning experiments here, I consider dropout rates in a uniform distribution over the range $[0, 0.9]$. As with the model hyperparameters, this is not an exhaustive approach, as the dropout rate is set globally for all parts of the model—it is conceivable, for example, to use different dropout rates for the embedding layers and the recurrent layers (e.g. Gal and Ghahramani, 2016).

The Adam algorithm is an adaptive optimization algorithm, i.e., the actual learning rate is calculated individually for each parameter update, and the learning rate hyperparameter only gives an approximate bound for these updates. Kingma and Ba (2014) suggest a value of $\alpha = 0.001$ as a good default setting. For tuning, I consider values in the range $[0.0001, 0.1]$, but since minor changes to the learning rate should be less important than its general order of magnitude, the values are drawn from a logarithmically uniform distribution in this case.

These learning hyperparameters are only tuned on the base encoder–decoder model, with the resulting configurations being used for the attentional model as well. As only two hyperparameters are tuned (compared to the previous three), only 30 trial runs are performed for each dataset.

Figure 6.7 shows the results of the hyperparameter tuning for each of the two learning parameters individually. High dropout rates > 0.5 always lead to a noticeable decline in accuracy; the same is true for high learning rates that, after a certain point (approx. $\alpha > 0.005$), cause the model accuracy to steadily decrease the higher α becomes. For dropout, the best values are mostly found in the range $[0, 0.4]$, while the ideal learning rate is typically concentrated around the suggested default setting $\alpha = 0.001$.

This impression is mostly confirmed by the plot in Figure 6.8, which again shows the model accuracy categorized into "best", "good", and "other" results with regard to the combination of the two hyperparameters (cf. p. 94). For English and Hungarian, the better results are centered around a dropout rate of about 0.2; German and Icelandic prefer the same or slightly lower dropout rates, while Slovene works better with slightly larger ones. In general, a dropout rate

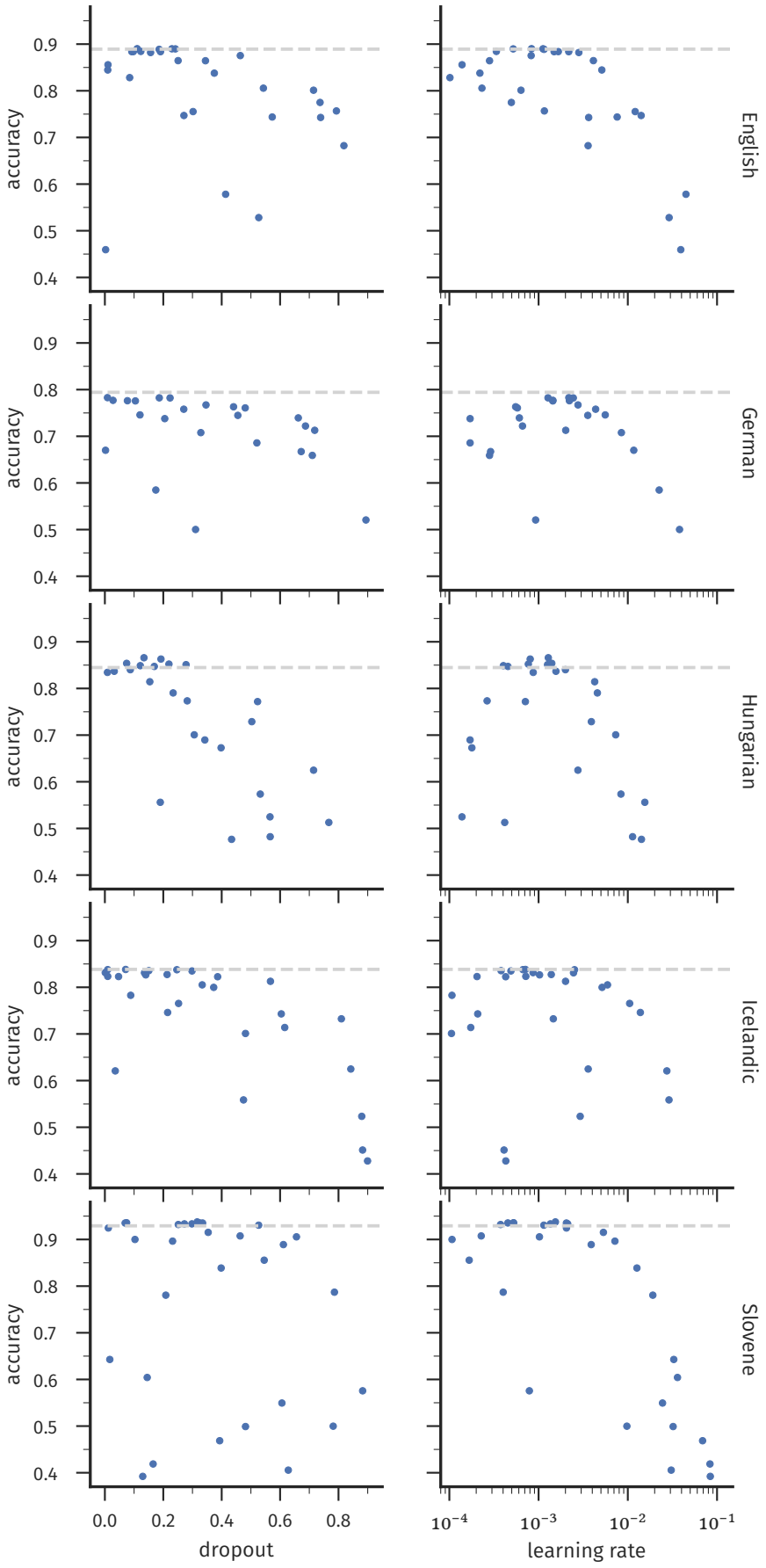


Figure 6.7: Accuracy of the base encoder–decoder model on the development sets as a function of each individual learning hyperparameter; dashed lines represent the highest accuracy from the model hyperparameter trials.

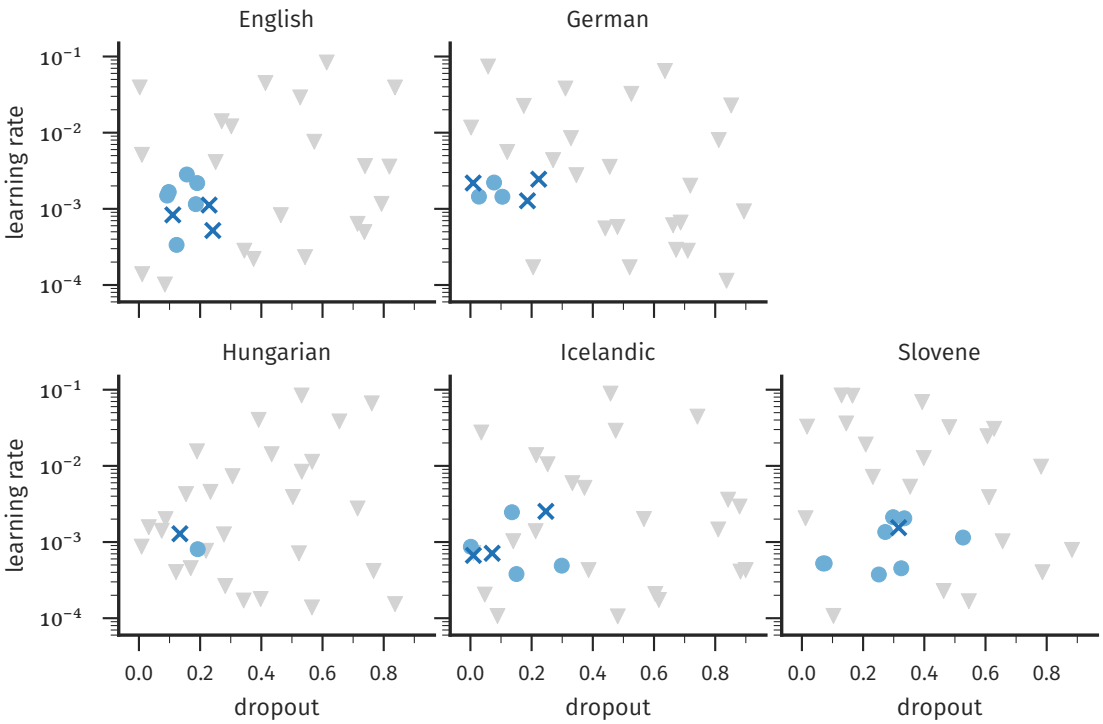


Figure 6.8: Combinations of learning hyperparameter values categorized by the distance of their accuracy a_i to the dataset's overall best accuracy $a_{\max}$; "best" ×: $|a_{\max} - a_i| \leq 0.001$, "good" •: $0.001 < |a_{\max} - a_i| \leq 0.01$, "other" : $|a_{\max} - a_i| > 0.01$.

of 0.2 appears to be a good compromise. No scenario clearly prefers learning rates that deviate from the default, except maybe the German dataset which suggests that a slightly increased value might be beneficial. In conclusion, though, the results do not offer a good reason to deviate from the default of $\alpha = 0.001$.

6.2.5 Final hyperparameter settings

Tuning of the model hyperparameters showed that, except for some particularly low values, the encoder–decoder model for the normalization task is not particularly sensitive to the exact dimensions of the embedding and hidden layers. Also, stacking more than a single LSTM layer does not seem to be beneficial in most scenarios. Learning parameters are more sensitive to extreme values: dropout noticeably decreases accuracy beyond a certain rate (around 0.4), while the learning rate of the Adam optimizer is also best left unchanged from its default.

While some datasets behave slightly differently from others—Hungarian, for example, seems to slightly favor deeper and higher-dimensional models—the overall results are similar enough to select a single configuration for all datasets. For all further experiments, the encoder–decoder models will use the following hyperparameters:

- a *depth* of one, i.e., a single LSTM layer in the encoder and decoder;
- *embedding layers* with dimensionality $d_m = 60$;

- *hidden layers* with dimensionality $d_h = 300$;
- a *dropout rate* of 0.2; and
- a *learning rate* for Adam of $\alpha = 0.001$.

These settings apply to both the base model and the attentional model.

6.3 Analysis

In the previous section, I determined which configuration of hyperparameters to use. The tuning process also provided first clues that the attentional model has a slight advantage over the base model. In this section, I will analyze the properties of the encoder–decoder model further, to answer questions such as:

- How stable is the training process, i.e., how do the results vary for different restarts of the training process?
- Is the attentional model always preferable to the base model on all datasets?
- What is the impact of the different decoding techniques introduced in Sec. 6.1.3?

To address those questions, I will train encoder–decoder models—with the hyperparameter settings provided in Sec. 6.2.5—on all datasets and evaluate them using the development portion of those datasets. Importantly, since my goal is to analyze the model's properties and possibly improve its performance further, I do *not* yet evaluate on the test sets here and do *not* compare the results to other normalization methods, which will be done in Chapter 10.

6.3.1 Stability of the training process

Training a neural network is not a fully deterministic process; different training runs can lead to different parameter configurations being learned and, consequently, different accuracy scores of the resulting model. This is mainly due to two factors (cf. Sec. 5.3.4): (i) the random initialization of weights before the start of the training process; and (ii) the shuffling of the training data before each training epoch. While this process can technically be made deterministic by fixing the seed of the random number generator, this only shifts the source of the randomness to the choice of the seed value.

Another factor potentially leading to varying accuracy scores of multiple training runs is the question of when to stop training. In this work, I always use early stopping (cf. Sec. 5.3.6), which stops training when the accuracy on the validation set has stopped improving. While this is a deterministic criterion, it does not guarantee that training stops at an ideal point—it cannot be ruled out that validation accuracy would improve further at some later point in time if training were continued. Different random restarts could exhibit different behaviors in this regard, and it is possible that some training runs result in lower accuracy only because they were stopped too early.

Dataset	Base model				Attentional model			
	min	max	avg.	s	min	max	avg.	s
DE_A	87.05%	87.98%	87.53%	0.4467	87.49%	88.14%	87.78%	0.2378
DE_R	85.60%	86.34%	86.02%	0.3213	86.42%	87.25%	86.91%	0.3042
EN	93.36%	93.80%	93.52%	0.2037	93.52%	94.02%	93.78%	0.2301
ES	93.41%	93.85%	93.60%	0.1612	93.53%	94.32%	93.99%	0.3170
HU	87.09%	87.56%	87.39%	0.1961	88.03%	89.36%	88.75%	0.4905
IS	84.11%	84.99%	84.57%	0.3479	83.84%	85.73%	84.55%	0.7286
PT	93.57%	93.87%	93.71%	0.1361	93.61%	94.01%	93.88%	0.1557
SL_B	89.18%	89.90%	89.40%	0.2909	89.44%	89.85%	89.61%	0.1564
SL_G	94.39%	94.88%	94.66%	0.1912	94.96%	95.39%	95.17%	0.1639
SV	85.92%	87.71%	87.17%	0.7244	88.33%	89.31%	88.71%	0.4078

Table 6.1: Statistics over five independent training runs per dataset and model type; min/max/avg correspond to minimum, maximum, and average validation accuracies for all training runs, while s denotes the sample standard deviation.

For estimating the true performance of a neural network model, it is desirable to quantify the variance from starting the training process with different random seeds. Otherwise, it is difficult to judge the significance of different accuracy scores from two experimental settings—i.e., are we observing an actual effect, or just random variation? Statistical significance testing can be misleading here, as it might be able to tell us that experimental setup A performs significantly better than setup B, but cannot make any guarantees that a different random restart of A will yield the same result, or in fact whether the result holds true when averaging the results of running A and B multiple times with different seeds. In fact, a recent study by Reimers and Gurevych (2017) showed that simply changing "the seed value for the random number generator can result in *statistically significant* ($p < 10^{-4}$) differences for state-of-the-art systems"[13] for sequence tagging.

Therefore, if we do not factor in the stability of the training process under different random initializations, any claims about generalization to other datasets (or even reproducibility on different hardware or software versions!) are, at best, to be treated with caution.

Variance

The simplest, although somewhat computationally expensive, way to estimate the variance of the training process—including the effect of early stopping—is to train the same model several times with different random initializations and compare the results. Therefore, for each dataset, I choose to train and evaluate both the base encoder–decoder model and its attentional variant five times each. The same five randomly determined seeds were used in each configuration to initialize the random number generator.

[13] Emphasis by the authors.

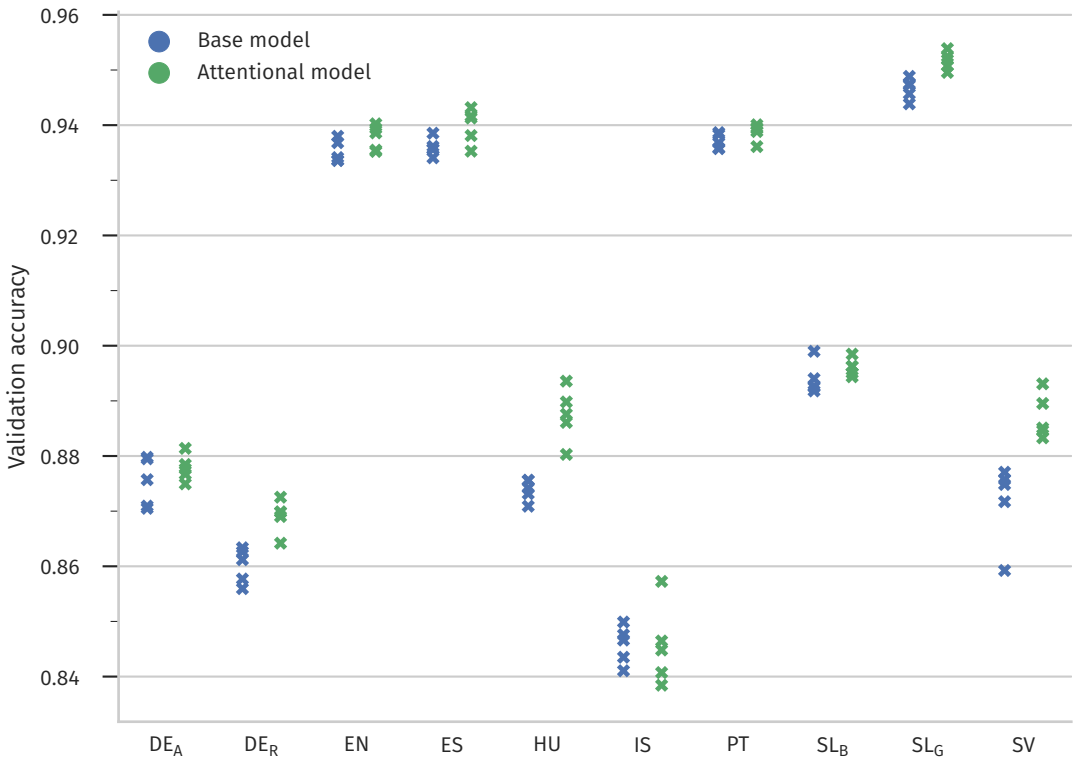


Figure 6.9: Validation accuracy of five different initializations per dataset and model type

The full results of these experiments are shown in Figure 6.9; in addition to that, Table 6.1 gives a summary of minimum/maximum/average scores and their sample standard deviation.[14] The spread of the observed accuracy scores ranges from 0.3 pp (Portuguese, base model) to almost 1.9 pp (Icelandic, attentional model). Generally, there is a tendency for this spread to be lower the higher the absolute accuracy scores become; e.g., with the base model, German (Anselm & RIDGES), Icelandic, and Swedish all have an average accuracy below 88% with $s > 0.3$, while English, Spanish, Portuguese, and Slovene/Gaj average an accuracy above 93% with $s < 0.21$, but some exceptions to this exist (such as Hungarian).[15] A similar, but weaker, tendency can be found correlating the standard deviation to the size of the validation dataset (cf. Tab. 3.1).[16]

Compared to the effect sizes often reported in historical normalization, which are not seldom below 2 pp (e.g., Reynaert et al., 2012; Pettersson et al., 2014a; Bollmann et al., 2017), the standard deviations of the encoder–decoder models appear to be relatively high. Indeed, the results highlight the perils of relying on a single training run to draw conclusions about model performance. Consider the case of Icelandic: if we wanted to evaluate the accuracy of the base model vs. the attentional variant and, by chance, selected the random initializations of the results on the extreme ends of the spectrum, we could either report an improvement for the attentional model of 1.62 pp (cf. Tab. 6.1, "min" for base, "max" for attentional) or a *decline* in

[14] The sample standard deviation is defined as $s = \sqrt{\frac{1}{n-1}\sum_i (x_i - \bar{x})^2}$, where x_i is the accuracy of the i-th training run, $\bar{x}$ is the average accuracy of all runs, and n is the total number of runs (here, $n = 5$).

[15] Calculating Spearman's ρ for the average accuracy and the standard deviation s, there is an inverse correlation of $\rho \approx -0.77$ for the base model and $\rho \approx -0.58$ for the attentional model.

[16] Spearman's $\rho \approx -0.45$ (base model), $\rho \approx -0.38$ (attentional model).

accuracy of 1.15 pp (“max” for base, “min” for attentional). This corresponds to a change to the error rate by -10.2% or +7.7%, respectively. Considering the size of the validation set (around 6,100 tokens), the first result would turn out to be statistically significant with $p < 0.05$ using a chi-squared test,[17] albeit the second one would not.

6.3.2 Base vs. attentional model

To help answer the question if the attention mechanism is effective, i.e., if the attentional model performs better on average than the base model, we can look at the entirety of the training runs in Fig. 6.9. For some datasets, such as Hungarian or Swedish, the results are pretty clear, as each training run with attention performs strictly better than each of the runs without attention. For other datasets, the results of the two setups are much closer together, and the advantage of the attentional model is uncertain.

Looking at the separate training runs per dataset, we can estimate if the average accuracy scores of the base and attentional model are equal by using Welch’s t-test.[18] Using a significance level of 5%, we find that:

- the attentional model is significantly better than the base model on German/RIDGES, Hungarian, Slovene/Gaj, and Swedish ($p < 0.05$);
- the attentional model only barely misses the significance threshold on the Spanish dataset ($p \approx 0.051$); and
- there is no significant difference on English & Portuguese ($p \approx 0.1$), Slovene/Bohorič ($p \approx 0.2$), German/Anselm ($p \approx 0.3$), and Icelandic ($p \approx 0.96$).

A different approach of comparing the models is to look at the averages for each dataset (cf. Tab. 6.1) and perform a paired t-test comparing the difference between base and attentional models. The improvement of the attentional model turns out to be significant with $p < 0.01$.

In summary, while it is questionable whether the attention mechanism improves accuracy in *all* datasets, it significantly improves it in at least some of them, and is an unnecessary addition at worst—in particular, it never degrades the average performance. Therefore, unless computational complexity is an issue (as the attentional model is slightly more complex than the non-attentional one), the results suggest that there is no downside to always using the attentional encoder–decoder variant for historical normalization.

[17] To clarify: the result of the chi-squared test is not “wrong”, as the worst base model (among the five training runs evaluated here) probably *does* perform significantly worse than the best attentional model. It is just misleading to apply it here, in the sense that these two models are not representative of the average results that you would obtain from repeating these experiments several times with different initializations. The question we ask of the chi-squared test—i.e., is the difference between *these particular initializations* of the neural networks significant?—is just not the question we are ultimately interested in.

[18] This assumes the scores of the different training runs follow a normal distribution, which is not strictly known, but not an unreasonable assumption.

Dataset	Base model		Attentional model	
	Best single	Ensemble	Best single	Ensemble
DE_A	87.98%	88.64%	88.14%	**89.11%**
DE_R	86.34%	87.68%	87.25%	**88.40%**
EN	93.80%	94.48%	94.02%	**94.82%**
ES	93.85%	94.33%	94.32%	**94.46%**
HU	87.56%	89.57%	89.36%	**90.64%**
IS	84.99%	**85.79%**	85.73%	85.73%
PT	93.87%	94.50%	94.01%	**94.67%**
SL_B	89.90%	**90.77%**	89.85%	90.60%
SL_G	94.88%	95.40%	95.39%	**95.62%**
SV	87.71%	89.00%	89.31%	**89.62%**
Avg. Δ	–	*+0.93*	–	*+0.63*
Avg. ER	–	*-9.34%*	–	*-6.80%*

Table 6.2: Validation accuracy of model ensembles compared to the best individual model; "Avg. Δ" shows the average difference to the best individual models (in percentage points), while "Avg. ER" gives the average error reduction (in percent); best result per dataset highlighted in bold.

6.3.3 Ensembles

As we have seen above, one way to deal with a model's variance during evaluation is to train it repeatedly with different initializations and, instead of relying on a single training run, to report average accuracy scores. Another option is to consider a *model ensemble*, i.e., using a combination of the individually trained models to generate predictions (e.g., Hashem, 1997; Goldberg, 2017, p. 60). The general idea here is that each individual model has learned slightly different things, and by combining the knowledge of all of them, we can obtain a model that generalizes better. Another view is that an ensemble "smoothes out" the individual errors made by each model.

To test if ensembling can help in our case, I combine the models trained in Sec. 6.3.1 using an *ensemble averaging* approach. Each model outputs a probability distribution over the set of possible (normalized) characters; in this approach, each model is run independently on the input data and the output probabilities from all models are averaged to form the output of the ensemble. This does not require any internal modifications of the involved models, which means it can be used with any type of model and decoding technique.

Table 6.2 shows the accuracies of the model ensembles. In almost all cases, the ensemble performs better than the *best* of the five individual models it is composed of. The only exception here is the ensemble of the attentional models for Icelandic, which "only" performs equally well as its best component (but note that the latter also is an extreme outlier in terms of accuracy, cf. Fig. 6.9). Overall, this is a strong result in favor of the ensembling approach.

In most cases, the attentional ensemble performs better than the base ensemble for the same dataset. However, this does not hold true for Icelandic and Slovene/Bohorič, where the base

ensembles slightly outperform the attentional ones. Still, the general tendency that using the attention mechanism is advantageous holds for ensembles as well.

Finally, it should be noted that ensembling could be performed in many other ways than the one evaluated here. The combination of models could be done using a weighted average, possibly giving more weight to better-performing models. The individual models need not be the result of fully independent training runs, but could also be multiple snapshots of the same model saved at different epochs during the training process—this would reduce the overall training time considerably, as only a single full training run is needed. The models do not even have to be of the same type; an ensemble could combine models trained with different hyperparameter settings, or even models of completely different architectures, as long as all of them output a probability distribution over identical output classes. It is left up to future research to determine if more sophisticated ensembles could improve performance even further.

6.3.4 Effect of decoding technique

All evaluations performed so far have used greedy decoding for their predictions, i.e., they simply choose the best normalized character (as predicted by the model) at each timestep until the full normalized word form is generated. Sec. 6.1.3 discussed this in more detail and introduced *beam search* and *lexical filtering* as possible enhancements to the decoding process. Here, I will analyze if these techniques actually improve the model's performance.

First, I use all of the models trained in Sec. 6.3.1 and evaluate them again, but this time using beam search decoding (with a beam size of five). The distributions of the resulting accuracy scores vary from those presented in Fig. 6.9, but the general conclusions do not change: the standard deviations cannot be shown to be significantly lower or higher compared to those using greedy search (as shown in Tab. 6.1), and looking at the average scores, the attentional model still significantly outperforms the base model. Comparing the two decoding techniques, the average scores for beam search are indeed higher than those for greedy decoding. However, as the evaluation above has shown that ensembles are preferable to any model in isolation, I will focus my evaluation on the model ensembles here.

Table 6.3 shows the accuracies of the model ensembles for various decoding techniques. In addition to greedy and beam search decoding, I also combine beam search with lexical filtering, using the modern resources described in Sec. 3.5 as the basis for the lexical filters.

The results show that beam search performs better than greedy search in almost every instance; in cases where greedy search is better or equally good, this is only by a tiny margin (of ≤ 0.2 pp). On the other hand, the improvements are relatively small: for the base ensemble, beam search gets an error reduction of only 3.3% when averaged over all datasets, while the attentional ensemble gets only 0.6%. In comparison, the introduction of the ensembling technique achieved an average error reduction of 9.3% and 6.8%, respectively (cf. Tab. 6.2), making it potentially more rewarding than the choice of decoding technique.

Lexical filtering does not generally seem to help, and even *decreases* accuracy for most datasets. Indeed, whether the filtering is helpful seems to depend most on the dataset: it improves accuracy for German/Anselm, English, Icelandic, and Portuguese, though in general, the gains are pretty small, with the Anselm dataset in the base ensemble showing the most improvement

Dataset	Base ensemble			Attentional ensemble		
	Greedy	Beam	Beam+Filter	Greedy	Beam	Beam+Filter
DE_A	88.64%	88.92%	**89.43%**	89.11%	89.31%	**89.76%**
DE_R	87.68%	**88.33%**	87.46%	88.40%	**88.51%**	87.78%
EN	94.48%	94.73%	**94.97%**	94.82%	94.81%	**94.91%**
ES	94.33%	**94.42%**	94.38%	94.46%	**94.61%**	94.34%
HU	89.57%	**90.08%**	87.58%	90.64%	**90.81%**	88.10%
IS	85.79%	85.97%	**86.09%**	85.73%	85.63%	**85.84%**
PT	94.50%	94.67%	**94.80%**	94.67%	94.71%	**94.87%**
SL_B	90.77%	**90.91%**	89.71%	**90.60%**	90.58%	89.44%
SL_G	95.40%	**95.55%**	91.89%	**95.62%**	**95.62%**	91.68%
SV	89.00%	**89.62%**	87.39%	**89.62%**	**89.62%**	88.02%
Avg. Δ	–	*+0.30*	*-0.95*	–	*+0.05*	*-0.89*
Avg. ER	–	*-3.34%*	*+10.58%*	–	*-0.58%*	*+10.34%*

Table 6.3: Validation accuracy of model ensembles for different decoding techniques; greedy decoding, beam search decoding, and beam search decoding combined with lexical filtering (cf. Sec. 6.1.3), using the corpora from Sec. 3.5 as its basis; "Avg. Δ" shows the average difference to greedy decoding (in percentage points), while "Avg. ER" gives the average error reduction/increase (in percent); best result per dataset and model type highlighted in bold.

(+0.65 pp compared to beam search without the filtering). On the other hand, when lexical filtering is not beneficial, the loss in accuracy can be pretty severe: for the attentional models, Hungarian loses 2.71 pp, while Slovene/Gaj loses 3.94 pp.

It makes sense that the performance of the filtering step depends heavily on the evaluated dataset, as its effectiveness is dependent on the overlap between the dataset's vocabulary and the content of the full-form lexicon. Whenever the filtering *decreases* the accuracy, the lexicon is missing more word forms from the target dataset than it helps to correct nonsensical word forms. The percentage of these "missing" word forms was shown in Tab. 3.7; indeed, both Hungarian and Slovene/Gaj have a high percentage of word forms not covered by the lexical resources (7.2% and 6.3%, respectively), while the percentages are comparatively low for those datasets that benefit from filtering. This is not a strict correlation, though: the highest rate of missing tokens is found in the case of Swedish (8.3%), which does lose accuracy from lexical filtering, but less so than other datasets. A possible explanation is that the majority of tokens that cannot be normalized correctly with the filter was not correctly normalized without the filter either.

In general, comparing the effects of the lexical filtering to the coverage of the modern resources reported in Tab. 3.7, a good rule of thumb appears to be that we should aim for a coverage greater than 96%—preferably even greater than 98%—in order to expect some improvements or, at least, not risk a significant loss of accuracy from the filtering step.

6.4 Summary

In this chapter, I introduced an encoder–decoder model for historical normalization, tuned its hyperparameters, and analyzed its variance as well as the effect of different decoding strategies. The model uses character-level inputs, standard LSTM units for the encoder and the decoder, and optionally uses an attention mechanism over the encoded representations (Sec. 6.1).

Hyperparameter tuning was performed using the TPE algorithm on a reduced subset of five of the historical datasets (Sec. 6.2). First, three model parameters were tuned: the depth (i.e., the number of LSTM layers) of the encoder and decoder, the size of the hidden layers, and the size of the embedding layers. Afterwards, two learning parameters are tuned on the best-performing model configuration: dropout rate and learning rate. Overall, the results were stable enough to justify selecting a single hyperparameter configuration for all further experiments and datasets, which is summarized in Sec. 6.2.5.

In the analysis part (Sec. 6.3), the training of the encoder–decoder model was shown to have a high variance, sometimes leading to significantly different accuracy scores between two training runs of the same model configuration. Model ensembling is a possible solution to this problem, as it consistently outperformed the accuracy of the best single training run. The attentional model sometimes outperformed the base model without attention, and was roughly equal in performance in other cases. Finally, when it comes to decoding techniques, beam search almost always leads to minor accuracy improvements over greedy search. The same cannot be said for lexical filtering, which should be used with caution, as it increased normalization accuracy on some datasets but noticeably decreased it on others.

Chapter 7

Comparative analysis

> *Unfortunately, errors committed by most MT systems span the gamut from innocuous to severe, and current systems seldom realize when they commit severe errors. Thus, a 95% [fully automatic high-quality translation] system in the worst case produces a translated text that is analogous to a jar of cookies, only 5% of which are poisoned. Such a cookie jar is useless without a complete professional analysis to localize the poisoned ones.*
>
> — Carbonell and Tomita (1985)

The previous chapters have introduced the normalization task, discussed challenges and previous work, and described and analyzed an encoder–decoder neural network model for normalization. In this chapter, I will compare this neural network model to previously established normalization methods.

This evaluation will not be confined to quantitative measures of model performance, but also consider qualitative differences in the model outputs, to help answer questions such as:

- What kinds of errors do the models make? Are most erroneous normalizations just slightly off, or complete nonsense? Can different evaluation measures help to highlight different qualities of the normalized words?
- Do the models have different strengths and weaknesses? Do different models work best on different languages? When model A outperforms model B, does it get all the same words correct as B plus a few more, or is there much less overlap?
- What types of words are most problematic? Can we predict when a normalization is likely to be wrong? Is there a common theme that can help us identify areas of improvement?

To approach these questions, I will first perform an extensive analysis on the *development sets* of all datasets. This includes analyzing individual examples from the datasets to find out more about the qualitative aspects of the automatic normalization, as well as testing different measures and approaches of quantifying a model's performance that could then be tested on the actual test sets.

Sec. 7.1 will first give an overview of the individual normalization methods that will be compared, while Sec. 7.2 evaluates them on the development sets and discusses advantages and disadvantages of different evaluation measures, such as word accuracy and character error rate. The following sections then delve deeper into a qualitative error analysis: Sec. 7.3 presents a

manual error classification on the German and English datasets; Sec. 7.4 looks at word stemming as a means of detecting normalization errors that are morphological in nature; Sec. 7.5 analyzes the models' capabilities to generalize by looking at words that were unknown during training, or words with unknown normalization patterns; Sec. 7.6 investigates ways to automatically predict whether a generated normalization is likely to be wrong; and Sec. 7.7 looks at the overlap of correctly and incorrectly normalized tokens between normalization methods, analyzing whether different normalizers are mostly correct on the same set of historical tokens.

7.1 Overview of normalization methods

All evaluations are performed by training each of the following tools and models separately on the training sets of each historical dataset described in Sec. 3.1:

1. The Norma tool (cf. Sec. 4.2.1), where I distinguish between three different settings:
   a) Norma_{M}: using only the "mapper" component of Norma, as it implements a simple wordlist substitution, which is a trivial normalization method that serves as a "low-effort" baseline;
   b) $\text{Norma}_{\text{R+W}}$: using only the rule-based (R) and Weighted-Levenshtein-based (W) components of Norma, as those are the actual character-based normalization algorithms provided by the tool; and
   c) $\text{Norma}_{\text{ALL}}$: using all three components together, as this is the way Norma is intended to be used in production.

   For the rule-based and Levenshtein-based components, the modern datasets introduced in Sec. 3.5 are used as Norma's lexicons.
2. The cSMTiser tool with default settings (cf. Sec. 4.2.2), using the development set as the tuning set for the Moses decoder, with two different scenarios:
   a) CSMT: training a (character-based) language model on the training data (default); or
   b) $\text{CSMT}_{\text{+LM}}$: training a (character-based) language model on the training data plus the word forms from the modern datasets (cf. Sec. 3.5), as the authors claim this is the easiest way to improve the system's accuracy.[1]
3. The encoder–decoder model with attention (cf. Sec. 6.1.2) using the hyperparameter settings described in Sec. 6.2.5, trained using early stopping (cf. Sec. 5.3.6) by validating on the development set, and evaluated using beam search (cf. Sec. 6.1.3) in the following scenarios:
   a) NN_{Avg}: The average of five independently trained models, as an indicator of the average performance to be expected when training only a single encoder–decoder model;[2]

[1] https://github.com/clarinsi/csmtiser/blob/8545f89/config.py#L43
[2] However, as this is computationally expensive to train and evaluate, I only include this figure in some of the evaluations.

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
$Norma_M$	83.48	82.51	92.60	92.09	73.93	82.39	91.90	80.77	94.12	83.65
$Norma_{R+W}$	77.15	84.11	90.31	88.98	80.82	82.93	87.57	85.86	90.10	83.61
$Norma_{ALL}$	88.26	87.27	94.51	94.52	86.20	86.07	94.57	88.41	92.04	87.88
CSMT	89.39	88.65	95.01	95.08	91.48	**87.12**	95.27	**92.06**	96.26	**91.98**
$CSMT_{+LM}$	86.63	**88.88**	**95.04**	**95.17**	**91.55**	87.02	**95.30**	91.97	**96.28**	91.85
NN_{AVG}	87.90	87.08	93.91	94.04	89.02	84.50	93.99	89.65	95.17	88.75
NN_{ENS}	89.31	88.51	94.81	94.61	90.81	85.63	94.71	90.58	95.62	89.62
NN_{ENS+F}	**89.76**	87.78	94.91	94.34	88.10	85.84	94.87	89.44	91.68	88.02
MFN	*95.46*	*96.41*	*98.71*	*97.40*	*98.62*	*93.12*	*97.74*	*98.49*	*99.13*	*99.73*

Table 7.1: Word accuracy of different normalization methods (cf. Sec. 7.1) on the development sets of the historical datasets, in percent; best result for each dataset highlighted in bold; bottom line gives the theoretical maximum accuracy obtainable when the most frequent normalization (MFN) was chosen for each historical type (cf. Sec. 3.4).

b) NN_{ENS}: An ensemble of five independently trained models (cf. Sec. 6.3.3), as this was shown to perform significantly better than any single model in isolation; and

c) NN_{ENS+F}: The same ensemble evaluated with lexical filtering (cf. Sec. 6.1.3), using the training data plus the modern datasets (cf. Sec. 3.5) as the lexicon.

7.2 Evaluation measures

All evaluations so far—e.g., during the hyperparameter tuning in Sec. 6.2—have looked at *word accuracy*, i.e., the percentage of normalized word forms that exactly match the gold-standard target normalizations from the evaluation dataset. I will first present the same evaluation for all the different models in my comparison, discuss the results, and then go on to consider shortcomings of the accuracy measure and potential alternatives.

Table 7.1 shows the word accuracy of different normalization methods on the development sets of the historical datasets. The results show a pretty clear trend: despite the extensive hyperparameter tuning process and the addition of techniques such as model ensembling, beam search, and lexical filtering, the CSMT-based approach outperforms the neural network on all datasets except one, German/Anselm. A slight caveat is that the systems are evaluated on the same subsets of the data that are also used for tuning (in cSMTiser) or early stopping (in the encoder–decoder models), potentially biasing the results a bit—refer to the final evaluation in Chapter 10 for a comparison without these drawbacks.

On some datasets, the advantage of cSMTiser appears to be quite substantial, with improvements of almost 2.4 pp on Swedish and about 1.5 pp on Icelandic and Slovene/Bohorič compared to the best encoder–decoder setup. In other cases, the gains are smaller, with German/Anselm being

the only dataset that sees a small improvement (+0.37 pp) for the encoder–decoder approach over cSMTiser, and even then only with the addition of the lexical filtering step. Curiously, adding the same modern German lexicons to the language modeling step of cSMTiser *decreases* its performance significantly (by almost 2.8 pp), which is an unusual result; in all other cases, the accuracy scores of the CSMT models with or without the additional modern language data are within approximately ±0.2 pp of each other. As with the lexical filtering in the encoder–decoder model, adding the extra contemporary data to the language model of cSMTiser is not always beneficial, and any gains obtained from this addition are relatively small.

The baselines provided by the Norma tool are also very strong, especially in the $\text{Norma}_{\text{ALL}}$ setting that uses all of Norma's components. On many datasets, it even outperforms the average single encoder–decoder model, and in the case of Icelandic, it also outperforms the encoder–decoder ensemble. However, the $\text{Norma}_{\text{ALL}}$ setting also makes use of the trivial (but effective) wordlist mapping technique, which could conceivably be used to improve the other systems as well, a hypothesis that is investigated in the error distribution analysis in Sec. 7.7. Therefore, it seems fairer to compare the cSMTiser and encoder–decoder results to the $\text{Norma}_{\text{R+W}}$ scenario, i.e., the character-based components of Norma. Here, the accuracy scores are often considerably worse, often in the range of 2–5 pp compared to $\text{Norma}_{\text{ALL}}$ and worst for German/Anselm, where $\text{Norma}_{\text{R+W}}$ only achieves 77.2% accuracy compared to 88.3% by $\text{Norma}_{\text{ALL}}$. This highlights that the mapping approach is actually a crucial part of Norma's performance.

The different scenarios of the encoder–decoder model were already compared in Secs. 6.3.3 and 6.3.4, except that the NN_{AVG} score is now also based on beam-search decoding instead of greedy decoding, which does not change the overall conclusions. The final evaluation in Sec. 10.2 will show that similar effects can be observed on the test set splits.

7.2.1 Character error rate

While word accuracy is simple and easily interpretable, it is also a very crude measure, as it classifies normalizations into "correct" and "incorrect" without any consideration for the type of mismatches in the incorrect forms. For example, if a model correctly normalizes the stem of a word but gets the inflectional suffix wrong (e.g., *kynges* > *kings* instead of the target *king's*), this is treated the same way as a normalization that is completely inappropriate (e.g., *ayeins* > *aliens* instead of the target *against*), even though the former is still somewhat useful while the latter is not. Sometimes, the notion of a single "correct" normalization itself is challengeable, e.g., when the contemporary language allows for multiple acceptable variants (consider the English *-ize*/*-ise* in verbs such as *recognize*/*recognise*). The discussion of normalization guidelines in Sec. 2.4 presented many more examples of this kind.

A more fine-grained evaluation measure that is also frequently applied to the normalization task (e.g., Pettersson et al., 2014a; Ljubešić et al., 2016b; Tang et al., 2018) is the *character error rate (CER)*. In analogy to the word error rate sometimes used in evaluating machine translation systems, the character error rate can be defined as the Levenshtein distance between a predicted normalization and a gold-standard (or reference) normalization divided by the length of the reference normalization; cf. Equation 7.1.

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
$Norma_M$	0.395	0.304	**0.396**	**0.323**	0.415	**0.393**	**0.381**	**0.460**	0.455	0.295
$Norma_{R+W}$	0.396	0.322	0.439	0.423	0.322	0.407	0.466	0.525	0.425	0.331
$Norma_{ALL}$	0.393	0.321	0.471	0.413	0.328	0.422	0.499	0.553	0.436	0.310
CSMT	**0.350**	**0.258**	0.397	0.408	**0.251**	0.421	0.488	0.577	0.541	0.244
$CSMT_{+LM}$	0.379	0.262	0.401	0.413	0.261	0.429	0.490	0.568	0.546	0.240
NN_{ENS}	0.367	**0.258**	0.418	0.435	0.263	0.410	0.455	0.504	0.504	**0.230**
NN_{ENS+F}	0.397	0.306	0.473	0.455	0.292	0.431	0.501	0.524	**0.407**	0.320

Table 7.2: Average character error rate (CER) (as defined in Eq. (7.1)) on the subset of incorrect normalizations; lowest value for each dataset highlighted in bold.

$$\mathrm{CER}(\hat{y}, y) = \frac{\mathrm{LD}(\hat{y}, y)}{|y|} \tag{7.1}$$

Here, $\hat{y}$ is the normalization predicted by a model, while y is the correct reference normalization. The CER for an entire dataset is then calculated as the average CER of all tokens in that dataset.

While this measure appears to complement word accuracy well by providing insight about character-level mismatches, I argue that it is not very useful when applied to a dataset as a whole. This is because for any normalization system that already achieves a high word accuracy, the character error rate will highly correlate with the accuracy score, simply due to the fact that $\mathrm{CER}(\hat{y}, y) = 0$ whenever $\hat{y} = y$. For example, comparing the word accuracy scores in Tab. 7.1 with the same configurations evaluated using CER, they correlate with Pearson's $r \approx -0.93$. In other words, the average CER over the whole dataset conflates the questions of (i) how many word forms are normalized correctly, and (ii) how different the *incorrectly* predicted word forms are from their gold-standard targets. Since word accuracy can already answer the former question, we should focus on using character error rate to address the latter question, which can be done by calculating the average CER on the set of *incorrect normalizations* only.

Evaluation

Table 7.2 shows the results of this evaluation.[3] The distribution of the lowest CER scores, highlighted in bold, shows a different picture compared to that of the global word accuracy scores. Here, the mapper component of Norma is often closest to the correct target normalization, with

[3] The figures for the Slovene datasets in this and most other character-level evaluations are, unfortunately, a bit distorted. The test and development sets of Slovene contain a few instances (about 200) of historical word forms without any gold-standard normalization; this was done in cases where the correct normalization required a change in token boundaries that cannot be performed without knowledge of token context (Nikola Ljubešić, personal communication, June 20, 2017). I did not want to exclude these instances completely so that word-level and character-level evaluations remain comparable; instead, I treat the gold-standard normalizations for these instances as a string of length 1 that does not match any other character in the dataset.

Dataset	Original	Target	Predictions	
			WITHOUT FILTER	WITH FILTER
DE_A	vnvro	unfroh	unvor	unverfroren
DE_R	reinblumen	rheinblumen	reinblumen	reinbekommen
EN	lefftenaunte	lieutenant	leftenant	left
HU	mordosnac	mardosnak	mordosnak	mord
PT	emgelica	angélica	ingélica	inglesa
SL_G	možitvijo	možitvijo	možitvijo	moži
SV	halfbrodher	halvbroder	halfbroder	half roder

Table 7.3: Examples for incorrect normalizations with a higher character error rate (CER) in the encoder–decoder ensemble with filtering

CSMT and NN_{Ens} also having the lowest score on some datasets. The character-level algorithms of Norma, on the other hand, often show a much higher error rate; e.g., on Spanish, $Norma_M$ has a CER of 0.32, while it is 0.42 for $Norma_{R+W}$.

Overall, there is no clear "winner" in terms of lower error rate between the CSMT approach and the neural models. However, comparing the encoder–decoder ensemble with and without lexical filtering, we observe that adding the filter almost always increases the CER, sometimes considerably so (e.g., from 0.42 to 0.47 on English, and from 0.23 to 0.32 on Swedish). This is remarkable, as the filtering is intended to guide the normalization process by restricting the possible outputs to known "valid" word forms, consequently preventing the model from generating nonwords, i.e., string that do not form valid words in the target language. The much higher error rates show that, when the model is wrong, the filtering step can tend to produce normalizations that are *farther* from the desired target form.

Table 7.3 shows some example normalizations for this phenomenon. When it is incorrect, the neural network model with lexical filtering sometimes produces very short normalizations. For example, on English, the NN_{Ens} model normalizes *lefftenaunte* > *leftenant*, which is incorrect as the expected normalization is *lieutenant*, but still shows some reasonable applications of spelling transformations (e.g., *-naunte* > *-nant*). It is also a non-word, which the lexical filtering is supposed to prevent, but the effect is arguably more harmful in this particular case, as the model with filtering only produces *left* instead.

We can also observe the opposite case, i.e., the filtering step producing unreasonably long suggestions, as in the German example *vnvro* > *unfroh*, which the NN_{Ens} model normalizes as *unvor*, but as *unverfroren* when filtering is used. In the German lexicon, the shortest entry that starts with *unv-* is 9 characters long, with most entries being 12 characters or more. Due to this, once the beam search algorithm has removed any alternatives not starting with *unv-* (and the prediction without filtering suggests that the model favors this sequence), it cannot stop generating characters until it has reached one of these entries, however unlikely these now might be.

The examples also illustrate the issue of compounding, which is usually a productive process and therefore challenging for approaches relying on a lexicon. The Swedish *halfbrodher* > *halvbroder* 'half brother', for example, is partly correctly normalized by the encoder–decoder

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
$Norma_M$	0.016	-0.016	0.036	0.026	-0.005	0.080	0.031	0.011	0.043	0.003
$Norma_{R+W}$	-0.036	-0.008	0.048	0.030	-0.145	0.064	0.043	0.023	0.164	0.014
$Norma_{ALL}$	0.007	0.014	0.127	0.093	-0.106	0.122	0.112	0.038	0.178	0.099
CSMT	-0.032	-0.091	0.024	0.029	-0.197	0.129	0.055	-0.053	0.014	-0.059
$CSMT_{+LM}$	-0.031	-0.075	0.025	0.042	-0.190	0.125	0.057	-0.051	0.011	-0.062
NN_{ENS}	0.004	-0.090	0.050	0.079	-0.167	0.114	0.047	-0.083	0.005	-0.052
NN_{ENS+F}	0.039	0.001	0.135	0.139	-0.111	0.150	0.116	0.004	0.154	0.134

Table 7.4: Absolute difference between the character error rate (CER) of the models' incorrect predictions and that of the unnormalized word forms, when compared to the gold-standard normalization; negative scores indicate that the model is better (i.e., has a lower CER), positive scores indicate that unnormalized forms are better.

model as *halfbroder*, but as the target word is not covered by the modern lexical resource, adding lexical filtering produces *half roder* instead. The lexical filter, as it is implemented here, does not try to account for compounding in any way. A possible solution would be to explicitly allow compound formation from words in the lexicon, but care has to be taken with such an approach, as allowing arbitrary conjoined lexicon entries can quickly lead to almost any string being passed through, defeating the purpose of the filter. I do not explore this issue further.

Magnitude of the errors

The examples from Table 7.3 also suggest that, for the model with lexical filtering, it might actually have been better to not modify the source token at all—or, better yet, lift the strict lexicon requirement—than to produce a normalization that is highly inappropriate. This leads us to the question if we can detect these inappropriate candidates somehow without referring to the gold-standard normalization, a question I will come back to in Sec. 7.6.

We can also use the character error rate to assess whether cases like these are representative (for a normalizer's "incorrect" suggestions) or rather outliers. To do that, we can again look at the subset of incorrect predictions, but calculate the CER between the *original* (historical) word forms and the gold-standard normalizations, then subtract that from the CER of the normalizer's predictions. In other words, we estimate whether the incorrect normalizations are still "closer" to the correct target word forms than the unnormalized tokens. If not, the normalization has arguably done more harm than good in these cases.

Table 7.4 shows these absolute differences. Negative scores indicate that on average, the model's incorrect normalizations still have a lower CER than the unnormalized word forms. Remarkably, this is only consistently the case for the Hungarian dataset and, to a lesser extent, for German/RIDGES. On all other datasets, the improvements from the normalizers are either very small, or the error rate is actually lower for the *unnormalized* tokens. This effect appears to be relatively independent from the choice of normalization method.

These figures show that if we could reliably detect normalization candidates that are very likely to be incorrect, the average character error rate could actually be improved by falling back on the historical word form—i.e., not normalizing the token at all—in these cases.

7.2.2 Further alternatives

Character error rate is not the only character-level evaluation measure that could be applied to normalization. I also experimented with a measure based on the length of the *longest common subsequence (LCS)* of the candidate and reference word forms, again normalized by the length of the reference word form. The idea behind this approach is that mismatches in the middle of a token are probably more severe than those at the beginning or the end, as the former are more likely to affect the root of the word form, while the latter could be the result of wrongly normalized affixes.

An evaluation with this LCS-based measure showed that (i) when evaluated on the full dataset, it correlates strongly with both word accuracy and character error rate (Pearson's $|r| > 0.95$); and (ii) when evaluated on the subset of incorrect normalizations, it still correlates strongly with CER (Pearson's $r \approx -0.89$). A possible explanation is that for normalization candidates with few errors, these errors indeed tend to occur more often towards the beginning or end of a word form, causing the length of the LCS to be high while the edit distance is low. All in all, however, the LCS measure did not seem to provide sufficient additional insight over the CER measure to warrant a more detailed analysis.

Due to the parallels of the normalization task with (character-level) machine translation, it stands to reason that evaluation measures commonly used in machine translation could be applied to normalization as well, such as the popular Bleu score (Papineni et al., 2002). However, here the differences between the two tasks are more significant than their commonalities: e.g., measures like Bleu are invariant to word reordering, with the justification that in translations of a sentence, the contained phrases need not necessarily appear in the same position for the translations to be equivalent. The same cannot be said for characters in a normalized word form. Here, an acceptable normalization will probably always be close to the gold-standard normalization both in the number of matching characters and their position and ordering; a property that Levenshtein distance (on which CER is based) already measures well. Furthermore, even within the field of machine translation, measures like Bleu are not uncontroversial (see, e.g., Callison-Burch et al., 2006). In conclusion, they do not seem like a good fit for evaluating the normalization task.

Some papers (e.g., Pettersson et al., 2014a) also evaluate word-level normalization performance in terms of *precision, recall,* and *F-score.* These measures are highly relevant when normalization is done in the context of information retrieval—i.e., finding matching historical spellings in a document given a contemporary word form—and can also be insightful when the normalization process includes *variant detection*—i.e., the algorithm first decides if a historical word form is actually a variant spelling or not—or when the normalizer returns multiple candidates. However, in the evaluation scenario presented here, each historical word form is assigned exactly one normalization candidate; furthermore, compared to labeling tasks such as POS tagging, the number of gold-standard target types is considerably higher and potentially unbounded. For these reasons, I do not believe precision and recall scores are useful measures here.

7.2.3 Limits of quantitative measures

Using character-level measures such as character error rate (CER) gives an impression of the extent of errors a system makes, but not necessarily their quality. While a lower CER means that normalization candidates are closer to their desired targets, even a single character can change the meaning of a word significantly. For example, in the German/Anselm dataset, the neural network ensemble normalizes *vrô* > *froh* 'glad' as *früh* 'early' and *yrre* > *ihre* 'her/their' as *irre* 'mad'. These errors distort the meaning significantly, but result in a Levenshtein distance of 1 to the reference normalization. Similar examples from English include *gode* > *god* normalized as *good* and *veale* > *veil* normalized as *veal*.

On its own, a character-level evaluation measure based on Levenshtein distance does not give an indication of the nature or the distribution of errors. Therefore, the remainder of this chapter will explore options to analyze normalization errors in more detail.

7.3 Error classification

In order to get an overview of the nature and type of normalization errors that occur, I choose to first manually evaluate small samples of the normalized data from the development sets of the German and English corpora.[4] For each dataset and model, I investigate a randomly chosen subset of 100 incorrect normalizations—i.e., normalizations that do not match the gold-standard provided by the dataset—and classify them into one of these five categories:

1. Valid: predictions that should actually be considered correct, either because the gold-standard form is questionable (e.g., because it contains an obvious spelling error) or the gold-standard and prediction can be considered perfectly interchangeable.

   Examples:

   - EN: *king's* vs. *king's* (two different characters used for the apostrophe)
   - DE_R: *vbermaffigẽ* > *übermässigen* predicted as *übermäßigen* (*ss* can be considered an acceptable substitution of *ß*[5])

2. Good: predictions that could be considered correct, depending on context. Generally, I chose this category whenever I would consider the prediction reasonable given only the historical word form and without knowing the gold-standard normalization or the word context.

   Examples:

   - EN: *wyse* > *ways* predicted as *wise* (reasonable alternative based on the spelling)
   - EN: *be* > *are* predicted as *be* (undecidable without context)

[4] For corpora in other languages, I did not feel confident enough in my language abilities to properly judge all incorrect normalizations.

[5] While German orthography has hard rules about when to use *ß* vs. *ss*, some varieties—like Swiss Standard German—*only* use *ss*, and substituting *ß* > *ss* very rarely creates ambiguity.

- DE_A: *fůchent* > *sucht* predicted as *suchend* (indicative verb form vs. present participle; graphematically correct normalization without considering context)

3. Fair: predictions that are incorrect because they result in a non-word, but still transform some parts of the word form correctly.

   Examples:

   - EN: *cholde* > *should* predicted as *chould*
   - EN: *meynteigne* > *maintain* predicted as *meintain*
   - DE_A: *gefchriffte* > *schrift* predicted as *geschrift*

4. Bad: erroneous predictions, either from applying wrong transformations, producing nonsensical word forms, modifying the word when it does not need to be normalized, or not modifying the word form at all when it needs to be changed.

   Examples:

   - EN: *faulsly* > *falsely* predicted as *faulsly* (missing any normalization)
   - EN: *t'acertaine* > *to ascertain* predicted as *trace taint*
   - DE_A: *yecklich* > *jeglich* 'any' predicted as *igel* 'hedgehog'

5. Other: historical tokens that are non-words (e.g., Roman numerals) or clearly foreign-language material (e.g., Latin) and probably should not be counted for an evaluation.

Naturally, this classification involves some degree of individual judgement and is not fully objective. The Good category in particular leaves room for interpretation; e.g., instead of judging based on similarity alone, a historical linguist might consider sound changes or known historical spelling variants to evaluate the plausibility of a candidate word form. I take a more naive approach here, using only my intuitive judgement and, when in doubt, erring on the side of accepting a normalization as "Good".

7.3.1 Results

Figure 7.1 visualizes the results of this analysis for incorrect normalizations of the CSMT model and the encoder–decoder model with and without filtering. I also looked at samples from Norma and the $CSMT_{+LM}$ model, but did not find significantly different patterns, so I did not explore them further. Due to the small sample size and the degree of subjectivity involved, findings here should not be seen as definitive, but rather as indications that could prompt further investigation.

In each configuration, there is a small number of word forms in the Valid category, particularly in the English dataset, where many of these cases stem from different characters used for the same symbol or punctuation marks that—despite the data being tokenized—have not been split off from a word form. For English, this category amounts to approx. 10% of normalization errors for all three models, which seems huge if it was representative for the full dataset. This shows the amount of care that must be taken in preparing training and evaluation datasets, and that despite efforts—such as Unicode normalization (cf. Sec. 3.2)—to eliminate differing

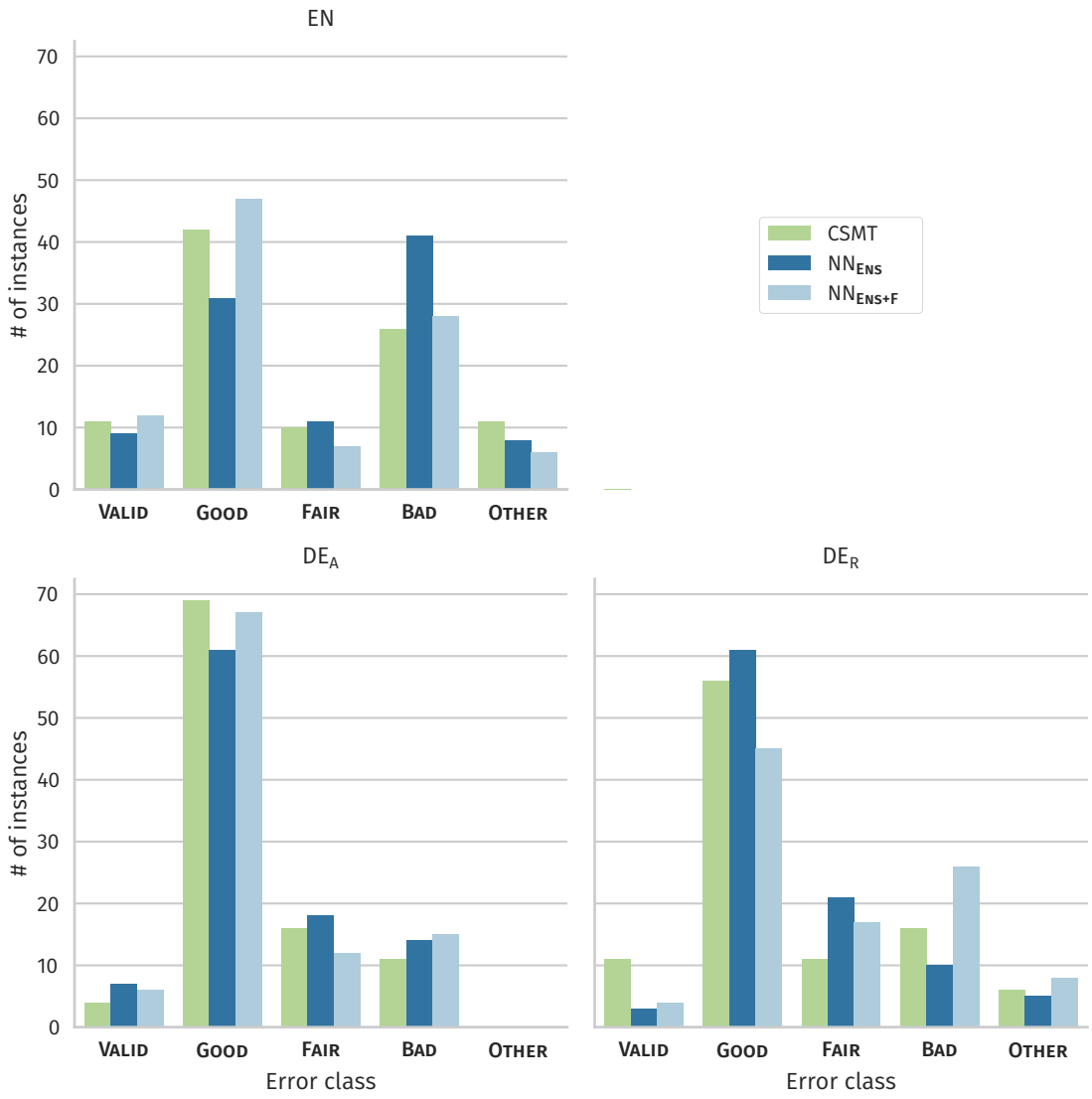


Figure 7.1: Error classification for randomly chosen samples of 100 incorrect normalizations, on the English and German datasets.

representations of the same characters, some cases (e.g., apostrophes) still managed to slip through.

Remarkably, the GOOD category usually represents the majority of all errors, with about 30–45 instances on English and 45–70 instances on the German datasets. This suggests that there are many more reasonable normalization candidates than predicted by the word accuracy score. A lot of these cases involve differences in inflection, which could also explain the higher figures for German, since German is morphologically more complex than English. Furthermore, correct inflectional forms can potentially be determined by considering word context, as features of surrounding words (such as case, gender, the presence of pronouns, etc.) might inform the correct inflection of the word form to be normalized. While the evaluation of MFN accuracy (cf. Tab. 3.3 and the discussion on p. 46) suggested that context might only be required for reaching the last 3% of word accuracy, the analysis here suggests that in practice,

inflectional differences and other ambiguities that are unresolvable without context account for a much higher percentage of errors.

"True" errors, i.e., predictions in the BAD category, are only a minority, although they still make up between 25 and 40 cases on English. In the best case, the models have left a variant word form unchanged; in the worst case, the normalizations can be considered highly inappropriate, such as the example of *t'acertaine* normalized as *trace taint*. Examples of this kind are most frequent with the NN_{ENS+F} model, as the lexical filtering forces the model to choose *something* within the lexicon, no matter how far from the historical word form it may be. This is particularly problematic for proper nouns; e.g., we can find *colford* > *culford* normalized as *coloured*, or *mongommery* > *montgomery* normalized as *mango merry*. Depending on the intended application, the effect of these mistakes can range from mildly amusing to highly embarrassing.

Some errors also reflect inconsistencies or other problems in the annotated dataset. In German/Anselm, the abbreviation *xp̄s* for *christus* 'Christ' is sometimes also normalized as *jesus*, leading to mismatches between these two alternatives. Extinct word forms in Anselm are normalized to Middle High German lexicon forms (cf. Sec. 3.1.2), which also leads to confusion in some cases. For example, *fuffiglich* (and several spelling variants, such as *fufziclich*, *fůffeklich*, *fueffigtlich*, etc.) is normalized to the Middle High German form *süezeclich* 'sweetish'; however, the graphematically similar forms *fufzlich*, *fůzlich*, *fuzzleich* etc. are normalized to contemporary *süßlich* 'sweetish' instead. This is motivated by the normalization guidelines, as the former instances still show an additional morpheme (*-ig/-ic/-ek*) that is not present in the latter forms, but due to their similarity, the encoder–decoder model incorrectly predicts *fuzzleich* > *süezeclich*. This shows generalization from the training data, albeit in an unintended way that is caused by the mixing of Middle High German and Modern German forms.

Finally, comparing the classifications of Fig. 7.1 for different normalizers, the general trends are usually similar, and while there are some outlier results (such as NN_{ENS} having noticeably more BAD predictions on English compared to the other two models), no reliable conclusion can be drawn due to the small sample size. On the contrary, as the proportion of the error categories is more often comparable than it is different, the analysis suggests that if there is a significant difference in error types between the models, it is probably relatively minor.

7.4 Stemming

A scenario that has been put forward above is that of a normalization candidate matching its gold-standard target except for, e.g., an inflectional affix. One way to estimate the extent of these occurrences is to compare the word forms only on the basis of their *word stems*. An easy way to implement this is by using an automatic stemming algorithm.

For my experiments, I choose to perform stemming automatically using Snowball stemmers (Porter, 2001); more precisely, using the language-specific stemming algorithms provided on the Snowball website.[6] These algorithms cover most of the languages in the historical

[6] http://snowballstem.org/

Method	Dataset						
	DE_A	DE_R	EN	ES	HU	PT	SV
$Norma_M$	8.07	16.36	8.19	28.12	3.03	3.79	7.08
$Norma_{R+W}$	14.89	34.02	8.53	31.00	19.73	6.86	19.02
$Norma_{ALL}$	18.31	32.77	8.58	45.92	20.30	11.23	22.43
CSMT	18.32	36.30	7.98	44.50	19.82	6.72	25.00
$CSMT_{+LM}$	13.12	36.57	7.41	46.71	19.33	7.00	25.14
NN_{ENS}	17.24	33.33	8.02	39.65	17.97	5.58	18.88
NN_{ENS+F}	17.92	32.27	8.66	41.27	24.75	12.33	20.82

Table 7.5: Percentage of incorrect normalizations that match the word stems of their gold-standard targets; datasets not represented here had no stemming algorithms for their target languages.

datasets, namely German, English, Hungarian, Spanish, Portuguese, and Swedish—no stemming algorithms were available for Icelandic and Slovene.[7] For an evaluation based on word stems, all gold-standard and predicted normalizations are run through the Snowball stemmer of the respective target language before being compared to see if they match.

Table 7.5 presents the results of this evaluation. They can be interpreted in two ways: (i) as the percentage of incorrect normalizations that have matching word stems; and (ii) as the error reduction (in percent) of the results in Tab. 7.1 if accuracy was evaluated on the basis of these word stems (instead of the full forms). Higher values are arguably better, in the sense that this indicates more word forms that have at least their stems normalized correctly. I will first look more closely at the differences between the datasets and some examples that were matched by the stemming process, before taking a look at differences between the normalization systems.

7.4.1 Dataset comparison

Table 7.6 shows several examples for incorrect normalizations with matching stems from all datasets in this evaluation; some of them will be discussed below.

English is among the datasets with the lowest percentage of matching word stems, which is not too surprising, considering that English has relatively little morphology. Examples that have been matched this way are the above-mentioned conflation of plural *-s* and genitive *-'s* (in *lordes* > *lords* instead of the gold-standard *lord's*), but also more sophisticated cases such as the normalization *recomaundehyde* > *recommend*, which is counted as incorrect since the correct target word form is *recommended*, but matches when using the stemming approach. Cases like these demonstrate why an evaluation based solely on word accuracy falls short: surely the prediction given by the system is highly useful here and should not be treated the same way as, e.g., leaving *recomaundehyde* unnormalized.

[7] The exact transformations performed by the Snowball Stemmer depend on the language-specific algorithm, but usually include suffix stripping and accent removal; cf. the examples in Tab. 7.6 or the descriptions on the Snowball stemmer's website.

Dataset	Original	Target	Prediction	Stem
DE_A	chůſſ	küsse	kuss	kuss
DE_A	füz	fuß	füße	fuss
DE_A	gedingñ	gedinges	gedingen	geding
DE_A	manicvaltec	mannigfaltige	mannigfaltig	mannigfalt
DE_A	vorhte	furcht	fürchte	furcht
DE_R	beweiſung	beweis	beweisung	beweis
DE_R	derſelbig	derselbe	derselbige	derselb
DE_R	halb	halber	halbes	halb
DE_R	ſein	seine	sein	sein
DE_R	ſtå̃men	stämme	stämmen	stamm
EN	begineing	beginning	begining	begin
EN	imagin	imagine	imagin	imagin
EN	lordes	lord's	lords	lord
EN	possiblelie	possibly	possible	possibl
EN	recomaundehyde	recommended	recommend	recommend
ES	anima	anima	ánima	anim
ES	enbie	envíe	envié	envi
ES	esta	está	esta	esta
ES	memo	memorias	memorio	memori
ES	reziberas	recibirás	reciberás	recib
HU	atÿanak	atyjának	atyának	aty
HU	helÿen	helyén	helyen	hely
HU	iersalomba	jeruzsálembe	jeruzsálemba	jeruzsál
HU	wýtezý	vitézi	vitézei	vitéz
HU	yduewzewlendewk	üdvözülendőek	üdvözülendők	üdvözülendő
PT	deixarão	deixarão	deixaram	deix
PT	paçara	passara	passará	pass
PT	respondera	responderá	respondera	respond
PT	sor	senhora	senhor	senhor
PT	tenh	tenha	tenho	tenh
SV	bemälte	bemälde	bemälda	bemäld
SV	thenne	denna	denne	denn
SV	tilsamman	tillsammans	tillsamman	tillsamman
SV	tingskiötta	tingsköta	tingskötta	tingsköt
SV	wendes	vänds	vändes	vänd

Table 7.6: Examples of incorrect normalizations with matching stems; all predictions are from the encoder–decoder ensemble without filtering.

The examples also show some pitfalls of the stemming approach. For example, *imagin* > *imagine* has not been modified at all by the encoder–decoder system, but is still counted as a match here as the stemmer cuts off the final *-e*. Since the stemmer does not consider whether an input string is actually a valid word form of the target language, but simply tries to apply a set of rules to strip off known affixes, stem matching may lead to "false positives" when an incorrect normalization happens to yield the same stem, but is not actually a valid word.

An interesting observation in Table 7.5 is that on German, the RIDGES dataset consistently has about twice the percentage of matching stems of the Anselm dataset. This is almost certainly a consequence of the different normalization guidelines followed by the two corpora, as discussed before in Sec. 3.1.2. While RIDGES consistently adds or modifies morphological inflection depending on word context, as in Ex. (1), the Anselm corpus always chooses the graphematically closest word form as the immediate normalization, even if this leads to an ungrammatical phrase when taken in context, as in Ex. (2):

(1) ſein tinctur
sein-e tinktur
his-FEM.SG tincture.FEM.SG

(2) ſein hant
sein hand
his hand.FEM.SG

The correct inflection here depends on word context, which I do not consider in my experimental setup. Consequently, any normalizer is bound to get some of these instances wrong in the RIDGES dataset, probably choosing the inflected normalized form of *ſein* that was observed most often in the training data. Stemming the normalizations then allows to find and match these instances. On the Anselm dataset, where the gold-standard normalization is unequivocally chosen without regard for word context, these instances would not produce "errors"—i.e., mismatching predictions—in the first place. This example highlights how conventions in producing the gold-standard normalizations can have a strong impact when evaluating (and comparing) automatic normalization results.

Spanish has the highest percentage overall in the stemming evaluation, up to 46.71% for the $\text{CSMT}_{+\text{LM}}$ normalizer (cf. Tab. 7.5), while the Portuguese data—which originates from the same corpus—has some of the absolute lowest scores (e.g., 5.58% for the encoder–decoder ensemble). Looking at the data, the result for Spanish is mainly due to the confusion of accented characters, which happens much less frequently in Portuguese. Indeed, this is another example of word context being important for determining the correct spelling, e.g., to distinguish *ésta* (pronoun) from *está* (verb), *anima* (verb) from *ánima* (noun), *envíe* (subjunctive form) from *envié* (indicative form), etc. It is trivial to find many more examples of this kind in Spanish.

In the Portuguese dataset, a common source of errors is the confusion between the third-person plural indicative verb suffixes *-am* (past/perfect) and *-ão* (future); in several cases, verbs with the *-ão* suffix are mapped to their *-am* forms in the gold-standard normalizations, leading the neural network model to overgeneralize in some cases where the *-ão* suffix should actually be preserved. Again, these cases are likely only decidable with word context.

7.4.2 Model comparison

Comparing the individual normalization methods in Table 7.5, the Norma_{M} system often shows the lowest scores by a large margin. This is not too surprising, as the wordlist mapping is typically a "hit-or-miss" approach: either it has learned a mapping for the historical word form, or it simply returns the historical word unchanged. In general, the latter case is less likely to result in a form with the correct word stem than a normalization method that performs character-level operations. However, Norma_{M} can also apply a learned mapping that does not match the target normalization in a given instance, which is likely to happen when a word form can be normalized with different inflectional forms (as in *lordes* > *lords/lord's*).

Considering the encoder–decoder models with and without lexical filtering, they usually show a similar percentage of incorrect normalizations matched by stemming. Two major exceptions to this are Hungarian and Portuguese, where the model *with* filtering has a significantly higher percentage (18% vs. 25% for Hungarian, 5.6% vs. 12.3% for Portuguese). The Portuguese result is particularly striking, as the word accuracy on this dataset is almost identical for the two systems (94.7% vs. 94.9%; cf. Tab. 7.1). This means that either the quality of errors is considerably different between these systems, or they produce errors on different subsets of the data.

Looking at the Portuguese dataset, it seems that mostly the latter is the case. There are several examples of word forms that need not be changed at all, but are not covered by the lexicon. Therefore, while the model without filtering handles them correctly, the model with filtering is forced to change them. This often results in plural nouns being changed to singular, and vice versa; e.g., plural *castelhanos* and *caixeiros* are changed to singular *castelhano* 'Castilian' and *caixeiro* 'clerk', respectively, because only the latter forms are found in the lexicon, while it is the other way around for *miudeza* 'trifle', which the lexicon only includes in its plural form *miudezas*. This suggests that the overall word accuracy could possibly be higher for the filtering approach if a more complete lexicon was used.

For the CSMT system, the variants with and without the additional contemporary data for language modeling also perform similarly. The biggest outlier is German/Anselm, where the $\text{CSMT}_{\text{+LM}}$ system has a considerably lower score, but this setting was an outlier in terms of word accuracy as well. Also noteworthy is the result on Spanish, where $\text{CSMT}_{\text{+LM}}$ is 2.2 pp better than the plain CSMT system, while also having a slightly better word accuracy (+0.09 pp; cf. Tab. 7.1). In other words, the $\text{CSMT}_{\text{+LM}}$ system performs better in terms of word accuracy than the variant without the added language modeling data, while also having more word stem matches among its incorrect normalizations. This suggests that the qualitative improvement of this system might actually be better than reflected by its word accuracy performance alone.

7.4.3 Conclusion

All in all, the stemming approach appears to be very successful in identifying mismatching, but still partly correct normalizations. Even though it can also lead to false positives—e.g., incorrect predictions matching the stemmer output by chance—it is certainly more sensitive to the type of error than Levenshtein distance. For a quantitative evaluation, it seems like a useful measure to complement plain word accuracy, while also providing a partial insight into the type of errors that the normalizer makes.

7.5 Generalization

A crucial aspect of any supervised machine learning system is its ability to *generalize,* i.e., not only perform well on data that has been seen during training, but also successfully handle previously unseen cases. A common way to estimate this ability is to consider the model's performance separately on tokens that have or have not been seen during training. Here, I will additionally investigate if and how this concept can be transferred to the character level, since most normalizers perform character-level operations.

7.5.1 Word-level analysis

Table 7.7 shows the word accuracy separately for *known* and *unknown* tokens, i.e., historical word forms that are or are not also contained in the training set.[8]

For known tokens, accuracy is consistently high, reaching 95% or more on most datasets. The main exception is Icelandic, which only achieves 89% accuracy in the best case ($\text{CSMT}_{+\text{LM}}$). Recall, though, that the Icelandic dataset shows an unusually high level of ambiguity in its gold-standard normalizations (cf. p. 48), which is likely to be responsible for this result.

Furthermore, the best result on known tokens is almost always achieved by Norma, more precisely its wordlist mapping component (Norma_{M}).[9] On German/Anselm and Icelandic, where a cSMTiser system performed better, Norma is only 0.03–0.13 pp behind. While the difference of the other systems to Norma is also relatively small, this clearly illustrates the effectiveness of a simple list-based replacement approach. In fact, given sufficient training data to achieve good coverage of the replacement list, more sophisticated machine learning approaches seem almost unnecessary for those word forms and could rather focus their efforts on the subset of unknown tokens or, alternatively, on disambiguating those historical words with more than one observed normalization target.

For unknown tokens, the accuracy of all systems is considerably worse. This is particularly true for $\text{Norma}_{\text{All}}$, which never performs better than 69% and is usually more than 10 percentage points (pp) behind the CSMT systems. The Norma_{M} component always leaves the source token unmodified here (since it cannot have learned a mapping for unknown words), so its accuracy is simply indicative of the percentage of tokens that do not need a normalization (cf. the ID columns of Tab. 3.3).

While the results for cSMTiser and the encoder–decoder model were comparable on the subset of known tokens, on unknowns, they show the same trends as in the full evaluation (in Tab. 7.1): the neural network performs better on German/Anselm, while cSMTiser performs better everywhere else, typically with a margin of 2–6 pp. This suggests that the overall higher

[8] This classification is made without any consideration whether the corresponding gold-standard normalization was also seen in the training set; i.e., "known token" only means that the training set contained the historical source token, but not necessarily with the same normalization as in the development/test data.

[9] Since known tokens are, by definition, always contained in Norma's list of word mappings, its other components are never called, so Norma_{M} is identical to $\text{Norma}_{\text{All}}$ here. Likewise, unknown tokens are *never* contained in the wordlist, so $\text{Norma}_{\text{R+W}}$ and $\text{Norma}_{\text{All}}$ are identical in that case.

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
Known Tokens										
Tokens	*40,914*	*8,237*	*15,078*	*10,755*	*12,707*	*5,413*	*24,817*	*4,662*	*18,043*	*1,708*
$Norma_{M/All}$	93.04	**93.64**	**97.59**	**96.62**	**96.39**	88.93	**97.02**	**96.31**	**98.30**	**98.77**
$Norma_{R+W}$	80.55	89.91	93.05	90.62	89.31	85.39	89.47	93.11	96.05	93.15
CSMT	**93.07**	92.90	97.23	96.32	95.97	89.04	96.88	96.10	98.28	98.59
$CSMT_{+LM}$	90.64	93.27	97.27	96.44	96.05	**89.06**	96.85	96.25	98.29	98.59
NN_{Ens}	92.73	93.20	97.33	96.31	95.62	88.01	96.72	95.37	97.76	97.37
NN_{Ens+F}	92.86	93.59	97.49	96.44	95.98	88.42	96.94	96.05	98.13	98.24
Unknown Tokens										
Tokens	*5,082*	*1,475*	*1,256*	*895*	*4,000*	*696*	*1,932*	*1,179*	*2,835*	*537*
$Norma_{M}$	6.49	20.34	32.64	37.65	2.60	31.47	26.19	19.34	67.55	35.57
$Norma_{R+W/All}$	49.74	51.73	57.48	69.27	53.85	63.79	63.15	57.17	52.24	53.26
CSMT	59.78	**64.95**	**68.31**	**80.22**	77.22	**72.13**	74.64	**76.08**	83.42	**70.95**
$CSMT_{+LM}$	54.35	64.34	**68.31**	79.89	**77.25**	71.12	**75.31**	75.06	**83.49**	70.39
NN_{Ens}	61.73	62.31	64.49	74.19	75.50	67.10	68.79	71.67	82.05	64.99
NN_{Ens+F}	**64.80**	55.32	63.93	69.16	63.08	65.80	68.32	63.27	50.65	55.49

Table 7.7: Word accuracy on the development sets, evaluated separately on known and unknown tokens (= tokens where the historical word form has been seen/not seen in the training data); best result per category and dataset highlighted in bold.

accuracy of the CSMT system is mostly due to the better handling of unknown tokens. There is no clear advantage for either CSMT or $CSMT_{+LM}$, though.

The lexical filter sometimes causes a sizeable decrease in accuracy on unknowns; the most extreme case of this is Slovene/Gaj, which has an accuracy of 82% for NN_{Ens} (without the filter), but drops below 51% on NN_{Ens+F}. If this effect only happens on unknown tokens, it directly points to an insufficient lexical coverage: the gold-standard normalizations (from the training set) of known tokens are, by definition, included in the lexicon, so an accuracy decrease on the unknown tokens suggests that this set of contemporary word forms was crucial for the filtering step. Indeed, for all datasets where more than about 5% of tokens in the development set are not covered by the lexicon (DE_R, HU, SL_B, SL_G, and SV; cf. Tab. 3.7), the accuracy on unknowns drops by at least 7 pp with lexical filtering. The opposite is also true: when the coverage is good, the filtering tends to hurt less or even improve performance (in the case of German/Anselm), with the only exception of the Spanish dataset (only 1.64% of tokens missing in the lexicon, but 5 pp decrease in accuracy).

7.5.2 Character-level analysis

The analysis above shows that models do generalize well to historical word forms that were not seen in the training set, although performance—as expected—is significantly lower. However, whether the full token was or was not seen during training is a very coarse criterion, since most models operate on a character level. Comparing character-level properties of the training and evaluation data could help us to better understand *which* properties of unknown tokens cause the most difficulty for the systems.

To analyze this, I turn to the character-aligned versions of the datasets that were introduced in Sec. 3.3 and define three categories of features:

1. Character alignment: a character in the historical word form and the contemporary character(s) it is aligned with.
2. Character tri-gram: a character tri-gram in the *historical* word form.
3. Character tri-gram + alignment: the combination of a character tri-gram in the historical word form and the contemporary character(s) that the *middle* character of the tri-gram is aligned with (i.e., a character alignment with its immediate context).

For each category, we can look at the subset of word pairs (i.e., a token and its gold-standard normalization) in the evaluation that have "unknowns" in that category. For example, a word pair has an "unknown character alignment" if its character-aligned version contains at least one character alignment that was not seen in any word pair of the training dataset. By looking specifically at the subset of tokens that fulfill this criterion, we can find out if the average word accuracy on this subset is noticeably lower, giving us an indication that this property of a word pair could be particularly challenging for a normalizer.

Table 7.8 presents the results of the evaluation for unknowns in each of the three categories. The more elements a category specifies, the more likely it becomes for instances of that category to not be covered in the training set: e.g., a specific character tri-gram plus alignment is more likely to be "unknown" than the alignment on its own. Consequently, while there are between 180 and 1,790 instances of "unknown character tri-gram plus alignment" per dataset, there are only between 9 and 174 instances for "unknown character alignments".

To give an example, in the Portuguese dataset, the alignment *o* > *u* is attested in many word pairs of the training set, such as *todo* > *tudo*, *deos* > *deus*, or *podera* > *puder*, but not within the historical character tri-gram *jos*, as in the example *josto* > *justo* from the development set. Therefore, while the alignment itself is known, the latter instance would fall in the "unknown character tri-gram plus alignment" category. On the other hand, the alignment *c* > *d* in the word pair *perca* > *perda* was never seen in the training set in any context, and would therefore classify as an "unknown character alignment".

From the three categories, unknown alignments pose the greatest challenge for all systems, with accuracy scores no greater than 26%, and sometimes falling below 8% (or even zero, on the nine instances from Swedish). When the combination of character tri-gram plus alignment is unknown, the systems fare significantly better, and accuracy scores improve even further when only unknown character tri-grams are considered—despite this subset regularly being only about half the size of the former. This reinforces the assumption that it is the unseen

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
Unknown character tri-gram + alignment										
Tokens	*1,790*	*523*	*398*	*221*	*812*	*180*	*524*	*330*	*459*	*170*
$Norma_{R+W}$	24.41	28.87	24.37	39.82	25.49	36.11	30.15	25.15	25.93	30.59
CSMT	34.58	**34.23**	**38.19**	**47.51**	**48.28**	39.44	36.83	**43.33**	**37.25**	44.12
$CSMT_{+LM}$	28.32	**34.23**	37.19	44.80	48.15	**41.11**	38.55	40.91	**37.25**	**44.71**
NN_{Ens}	41.28	31.93	30.40	32.13	45.44	37.78	30.34	39.70	34.86	41.18
NN_{Ens+F}	**46.03**	**34.23**	35.68	38.91	46.55	**41.11**	**42.65**	38.79	20.04	38.82
Unknown character tri-gram										
Tokens	*790*	*265*	*149*	*94*	*330*	*106*	*178*	*185*	*209*	*109*
$Norma_{R+W}$	35.95	40.75	31.54	57.45	33.94	53.77	46.07	38.38	44.98	36.70
CSMT	50.76	53.96	**54.36**	65.96	66.97	57.55	**65.73**	**65.41**	**76.56**	62.39
$CSMT_{+LM}$	40.13	**54.34**	53.69	**67.02**	**67.27**	**61.32**	65.17	63.78	**76.56**	**63.30**
NN_{Ens}	57.47	50.57	51.01	53.19	62.42	58.49	57.87	64.86	74.64	56.88
NN_{Ens+F}	**59.24**	44.91	42.28	54.26	55.76	59.43	62.36	56.22	33.49	41.28
Unknown character alignment										
Tokens	*174*	*84*	*48*	*62*	*88*	*28*	*96*	*28*	*53*	*9*
$Norma_{R+W}$	8.62	13.10	10.42	12.90	7.95	3.57	6.25	0.00	**3.77**	0.00
CSMT	7.47	7.14	8.33	**22.58**	13.64	3.57	3.12	0.00	1.89	0.00
$CSMT_{+LM}$	3.45	8.33	8.33	20.97	12.50	3.57	4.17	3.57	1.89	0.00
NN_{Ens}	23.56	5.95	10.42	4.84	13.64	0.00	3.12	3.57	0.00	0.00
NN_{Ens+F}	**24.71**	**15.48**	**14.58**	14.52	**26.14**	**7.14**	**12.50**	**7.14**	1.89	0.00

Table 7.8: Word accuracy on the development sets, evaluated separately on known and unknown tokens (= tokens where the historical word form has been seen/not seen in the training data); best result per category and dataset highlighted in bold.

Dataset	Original	Target	Predictions			Alignment
			$Norma_{R+W}$	CSMT	NN_{Ens}	
DE_A	iōgheren	jüngern	jungern	✓	✓	ō → ü
EN	felishippe	fellowship	felipe	feliship	✓	i → low
ES	annelo	anhelo	✓	✓	annelo	n → h
HU	zolǵlnac	szolgálnak	✓	szolglnak	✓	ǵ → gá
IS	skiftast	skiptast	✓	skiftast	skiftast	f → p
SL_B	weinrutizo	vajnrutico	minuto	weinrutico	beinrutico	w → v

Table 7.9: Examples of predictions for word pairs with an unknown character alignment; a checkmark (✓) indicates that the model predicted the correct target form; the unknown character alignment is highlighted and specified in the last column.

Original	Target	Predictions			
		$\text{Norma}_{\text{R+W}}$	CSMT	NN_{Ens}	$\text{NN}_{\text{Ens+F}}$
alquansado	alcançado	✓	✓	alquançado	✓
carragado	carregado	encorajado	✓	carragado	✓
colcada	colocada	coçada	✓	colsada	✓
josto	justo	✓	✓	josto	✓
lisbão	lisboa	libano	lisbam	lisbam	✓
magistade	majestade	magistrado	✓	magistade	✓
senõ	senão	✓	✓	senho	✓

Table 7.10: Example predictions on Portuguese that are only correct with lexical filtering added to the neural network model; all examples taken from the "unknown character tri-gram + alignment" subset. The unknown alignments are highlighted within the word forms; a checkmark (✓) indicates that the model predicted the correct target form.

alignments which are most problematic, either when they have not been seen at all, or not in the context that they appear in during the evaluation. Previously unseen character tri-grams by themselves are much less of a challenge for the models.

Table 7.9 shows some examples for word pairs from the "unknown character alignment" category. In German/Anselm, *iōgheren* > *jüngern* 'disciples' was normalized correctly by both CSMT and NN_{Ens}, despite the transformation $\bar{o} \longrightarrow \ddot{u}$ being previously unseen in the training data. However, the training set contains 21 different spelling variants for the modern *jüngern*, among them the relatively close variant form *iongheren*. Apparently, the surrounding characters provide enough context for the normalizers to favor the correct prediction.

In some instances, Norma's dictionary matching component using weighted Levenshtein distance proves to be advantageous where other methods fail. In the Icelandic dataset, the pair *skiftast* > *skiptast* contains the previously unseen mapping $f \longrightarrow p$. This causes both CSMT and NN_{Ens} to fail at generating the normalization, while $\text{Norma}_{\text{R+W}}$ is successful (and, not shown in Tab. 7.9, the $\text{NN}_{\text{Ens+F}}$ model as well). Since the rule-based component can, by definition, never apply a transformation that is not explicitly learned, this means the Levenshtein component must be responsible for this normalization. A similar example comes from German/Anselm with the word pair *baykenſtreich* > *backenstreich* 'slap (in the face)', which contains the unknown alignment $y \longrightarrow c$. While *backenstreich* does not appear in the training set, it does appear in the lexicon. Consequently, Norma generates it correctly, while other methods produce nonsensical suggestions (CSMT: *beikenstreich*, NN_{Ens}: *beichenspreich*, $\text{NN}_{\text{Ens+F}}$: *beichte reich*).

7.5.3 Local vs. global probabilities

Another interesting case study comes from the Portuguese dataset. Here, the neural network model with filtering sees an accuracy improvement of 12 pp over the model without filtering in the "unknown character tri-gram + alignment" scenario (cf. Tab. 7.9); similar tendencies can be observed for some of the other datasets as well. Table 7.10 presents a selection of word

pairs that are incorrect in the NN_{ENS} model, but correct when lexical filtering is added. Some of them contain multiple "unknown" instances, which is why they are represented by underlining instead of a separate column; e.g., in the word pair *magistade* > *majestade*, both the alignment *g* > *j* in the context of *agi* and the alignment *i* > *e* in the context of *gis* are unseen in the training data.

Considering the examples from Tab. 7.10, the predictions of the NN_{ENS} model always diverge from the correct target form in the vicinity of the alignment that is "unknown" in this context. Strikingly, the cSMTiser system gets almost all of these instances correct without the need for a lexical filter. One hypothesis to explain this observation is that the encoder–decoder model conditions its predictions more strongly on local features of the input string. When it encounters ⟨*q*⟩ in the context of *alquansado*, it strongly prefers to leave it as ⟨*q*⟩, since this is by far the most common alignment in the training set. While the alignment *q* ⟶ *c* occurs in the training set as well, it is never seen within this particular historical character tri-gram (*lqu*), so the neural network assigns it a lower probability here. The lexical filtering then imposes an external constraint on the decoder's output which shifts the probabilities in favor of the correct normalization.

Possibly, the CSMT system gives more weight to the global composition of the output, e.g. via its language modeling component; in other words, while it might also assign a higher (conditional) probability to the character normalization *q* ⟶ *q*, this might be corrected by a relatively low probability of the resulting output string. The encoder–decoder model, on the other hand, can only condition its prediction on the input string and the *previously* generated characters;[10] i.e., when it predicts the ⟨*q*⟩ in *alquançado*, it can only base this off the knowledge that the input form is *alquansado* and the previously generated characters are *al*. In particular, it cannot "backtrack" if the output sequence as a whole becomes unlikely towards the end; beam search is intended to mitigate this problem, but in these cases, it appears not to help enough.

This observation suggests that the encoder–decoder approach could possibly benefit from a stronger (character-level) language modeling component. While the lexicon filter works as a solution in this particular case, we already saw that it can be harmful in other cases (e.g., see Sec. 6.3.4), so an approach with less strict constraints on the output sequence might be preferable. In any case, a more thorough analysis of this phenomenon would be required, though I will leave this to future work.

7.6 Predicting errors

We have seen before (e.g., in Sec. 7.2.1) that when a predicted normalization is wrong, it can sometimes do more harm than good. Depending on the application scenario, if a normalization candidate is highly inappropriate, it might be better to leave the word form unnormalized instead. This leads to the question if we can detect candidates that are likely to be wrong without referring to the gold-standard annotation in our test set.

[10] Remember that only the encoder is bi-directional, while the decoder operates strictly from left to right; cf. p. 6.1.1 and the discussion of "input feeding" for more background on this.

7.6.1 String length and edit distance

One possible assumption about correct normalizations is that they are usually close to the historical source form. Pettersson, Megyesi, and Nivre (2013) investigate this assumption on the training set of the Swedish "Gender and Work" dataset and find that (i) when comparing string lengths of historical tokens and their gold-standard normalizations, 99.5% of tokens fall into the range of $[-4, +1]$ (i.e., the normalizations are up to four characters shorter or one character longer than their source form); and (ii) 98.8% of (gold-standard) normalizations have a Levenshtein distance of 4 or less to the historical source form. Inspired by this finding, we can ask if similar ranges can be found for all datasets, and if a normalization candidate is more likely to be incorrect when it falls outside a given range of one of these measures.

Figure 7.2 plots these two measures, i.e., *edit distance* and *string length difference* of the gold-standard normalization to the historical word form, for the development sets of all datasets. For the latter, the majority of normalizations in all datasets are of the same length as the source token, with absolute length differences of two or more characters being relatively infrequent. Edit distance shows a more nuanced picture: here, a distance of zero means that the normalization equals the historical word form, i.e., no changes are required. This is the majority case in only half of the datasets, which parallels the "ID accuracy" shown in Tab. 3.3.[11] While for most datasets, the percentage of tokens declines as the Levenshtein distance increases, two major exceptions are German/Anselm and Hungarian, where a distance of one or two is actually more frequent than the "unchanged" case. In general, with the exception of Hungarian, distances greater than four are very infrequent, which is the same threshold that Pettersson, Megyesi, and Nivre (2013) found.

The frequency distribution of these measures, however, does not tell us how useful they are in identifying incorrect normalizations. The fact that a certain value (for edit distance or length difference) is rare does not imply that a predicted normalization with this property is more likely to be wrong—after all, a hypothetical model with 100% accuracy would also include such instances.

To test the predictive power of these measures, we can train a logistic regression classifier (e.g., Smith, 2011, p. 88 f.)[12] using the measured value—string length difference[13] or Levenshtein distance—as its input, and the target classes "correct" or "incorrect" as its output. The idea is that if there is a good "threshold" above which a predicted normalization is more likely to be wrong than correct, training a logistic regression classifier should identify this threshold.

For each normalization system and dataset, I train a classifier on the automatically normalized development set of that dataset. Afterwards, I evaluate the classifier on the same set. If it cannot achieve good performance on the same subset of the data it was trained on, it is unlikely to be useful for classifying new, unseen normalizations.

[11] Although the numbers in Tab. 3.3 were generated on the training sets while those of Fig. 7.2 are based on the development sets, their overall scale is quite similar.

[12] For all experiments, I use the implementation of logistic regression provided by the `sklearn` library (Pedregosa et al., 2011), version 0.19.0.

[13] I use the *absolute value* of the length difference here, as my assumption is that a normalization candidate becomes more likely to be incorrect the higher the *absolute* length difference is, regardless of whether is shorter or longer than the historical form.

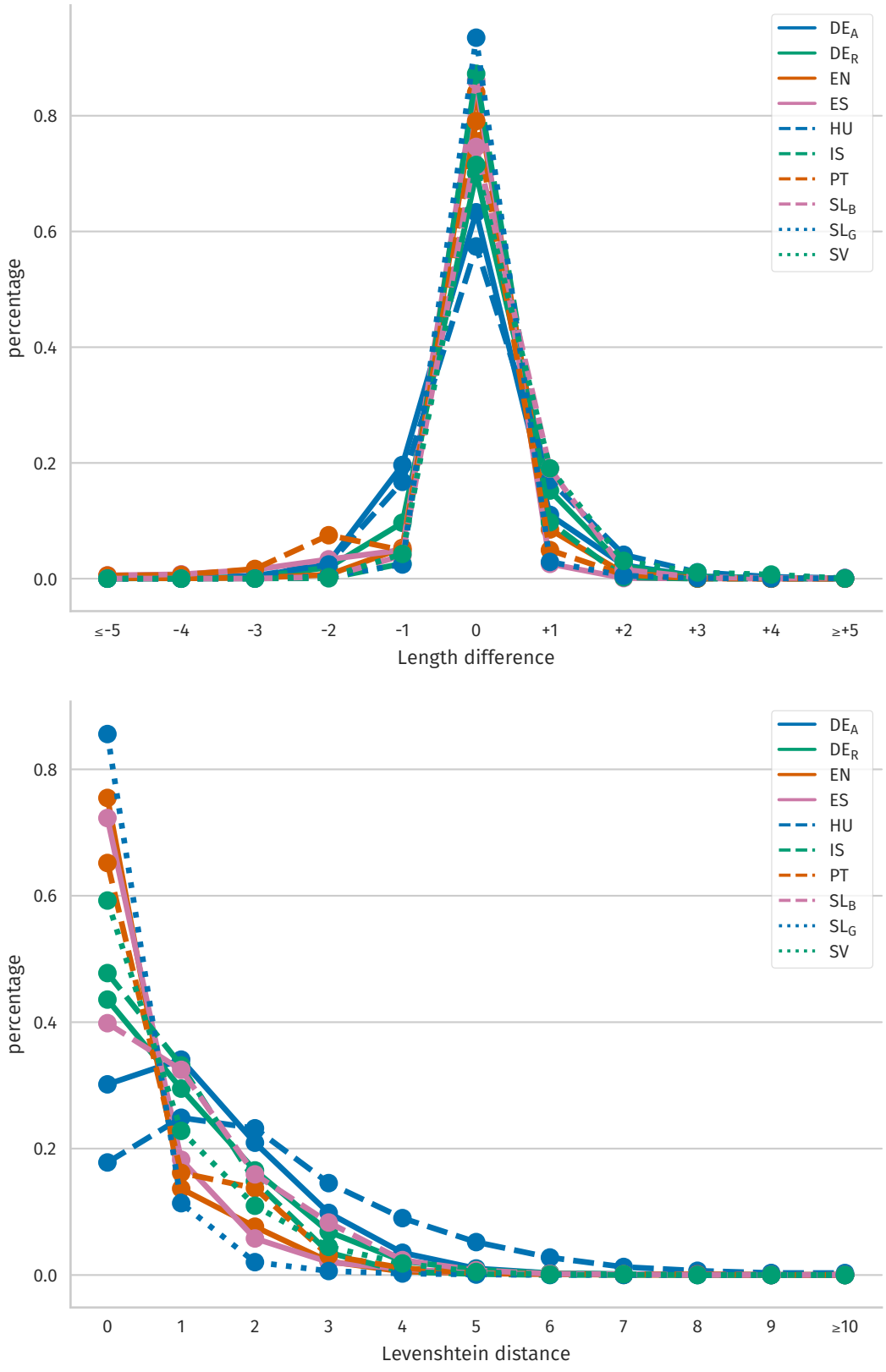


Figure 7.2: String length difference and Levenshtein distance between historical tokens and their gold-standard normalizations, given as the percentage of tokens in the development set of each dataset with a certain length difference (where positive numbers mean the normalization is longer than the historical word form) or edit distance.

Dataset	Length difference				Levenshtein distance			
	Tokens	Precision	Recall	F-score	Tokens	Precision	Recall	F-score
Norma$_{\text{ALL}}$								
DE$_{\text{A}}$	527	0.56	0.05	0.10	261	0.57	0.03	0.05
DE$_{\text{R}}$	216	0.83	0.15	0.25	183	0.70	0.10	0.18
EN	170	0.72	0.14	0.23	213	0.57	0.14	0.22
ES	19	0.11	0.00	0.01	34	0.06	0.00	0.01
HU	725	0.65	0.20	0.31	472	0.49	0.10	0.17
IS	62	0.82	0.06	0.11	42	0.60	0.03	0.06
PT	16	0.25	0.00	0.01	80	0.17	0.01	0.02
SL$_{\text{B}}$	338	0.80	0.40	0.53	267	0.63	0.25	0.36
SL$_{\text{G}}$	445	0.78	0.21	0.33	1,090	0.71	0.47	0.56
SV	94	0.69	0.24	0.36	82	0.61	0.18	0.28
CSMT								
DE$_{\text{A}}$	67	0.09	0.00	0.00	46	0.24	0.00	0.00
DE$_{\text{R}}$	11	0.18	0.00	0.00	7	0.43	0.00	0.01
EN	56	0.29	0.02	0.04	55	0.22	0.01	0.03
ES	4	0.00	0.00	0.00	4	0.00	0.00	0.00
HU	0	–	0.00	–	1	0.00	0.00	0.00
IS	0	–	0.00	–	28	0.25	0.01	0.02
PT	0	–	0.00	–	3	0.00	0.00	0.00
SL$_{\text{B}}$	0	–	0.00	–	1	1.00	0.00	0.00
SL$_{\text{G}}$	0	–	0.00	–	18	0.22	0.01	0.01
SV	0	–	0.00	–	1	0.00	0.00	0.00
NN$_{\text{ENS+F}}$								
DE$_{\text{A}}$	156	0.44	0.01	0.03	332	0.56	0.04	0.07
DE$_{\text{R}}$	108	0.71	0.06	0.12	179	0.71	0.11	0.19
EN	118	0.60	0.09	0.15	193	0.48	0.11	0.18
ES	19	0.11	0.00	0.01	41	0.22	0.01	0.03
HU	251	0.66	0.08	0.15	313	0.58	0.09	0.16
IS	5	1.00	0.01	0.01	55	0.64	0.04	0.08
PT	15	0.20	0.00	0.00	89	0.26	0.02	0.03
SL$_{\text{B}}$	39	0.90	0.06	0.11	74	0.68	0.08	0.14
SL$_{\text{G}}$	455	0.80	0.21	0.33	1,011	0.71	0.42	0.52
SV	45	0.71	0.12	0.20	110	0.64	0.26	0.37

Table 7.11: Precision, recall, and F-score of a logistic regression classifier on detecting incorrect normalizations, based on either string length difference or Levenshtein distance between historical word form and normalization candidate; “Tokens” gives the number of normalizations classified as incorrect.

Table 7.11 shows the results of this evaluation for some of the models, in terms of tokens predicted as "incorrectly normalized", as well as precision, recall, and F-score (e.g., Powers, 2011) on the task of identifying the "incorrect" normalizations.[14]

String length difference appears to be a very poor predictor for normalizer correctness. For CSMT, only a negligible amount of incorrect normalizations (if any) are correctly identified using this criterion, regardless of the dataset.[15] For $\text{Norma}_{\text{ALL}}$ and the neural network ensemble with filtering ($\text{NN}_{\text{ENS+F}}$), the results are slightly better, although the F-score is rarely above 0.25, with the maximum F-score being 0.53 (on Slovene/Bohorič with $\text{Norma}_{\text{ALL}}$).

Levenshtein distance is roughly comparable in terms of F-scores. For $\text{Norma}_{\text{ALL}}$ and $\text{NN}_{\text{ENS+F}}$, this classifier achieves a comparatively high F-score (of 0.56 and 0.52, respectively) on the Slovene/Gaj dataset. This is arguably the most "modern" dataset, as it requires the least amount of changes between historical form and normalization, both on a word level and on a character level (cf. the "ID" columns in Tab. 3.3). It also has the lowest curve in the Levenshtein distance distribution of Fig. 7.2—i.e., it has a lower percentage of gold-standard normalizations with a distance of one or greater than any other dataset. It appears that Levenshtein distance can be a useful predictor mainly when the dataset is modern enough that most word forms require no or very few modifications, but does still not work well in the general case.

For all configurations, the low F-scores are mainly a result of a poor recall. Precision is considerably higher than recall in all cases except those where precision is zero or undefined (due to zero instances being classified as "incorrect"). This suggests that for almost all values of length difference and edit distance, a considerable amount of instances are indeed correct normalizations, and a classification based solely on these criteria will therefore always fall short. To put it another way, while it is indeed the case that most gold-standard normalizations fall into a given range of length difference and edit distance (as Fig. 7.2 shows), it is *not* true that predicted normalizations that fall outside these ranges are more likely to be wrong.

7.6.2 Normalizer scores

A different approach to error prediction is to not rely on properties of the output word forms, but on scores generated by the normalization systems. Each system in this comparison produces some kind of score for each candidate it generates:

- The Moses decoder used by cSMTiser outputs a log-probability score for the sequence, i.e., the full normalized word form.[16]
- The final dense layer of the neural network outputs a probability for each character it generates, from which an average probability of all characters in a normalized word form can be derived.

[14] I also performed these evaluations with various *combinations* of features, i.e., using both string length difference and Levenshtein distance, adding the absolute length of the strings as a feature, using Levenshtein distance normalized by word length, etc. The classifier did not show improvements from any of these alternatives.

[15] In general, Norma_{M}, $\text{CSMT}_{\text{+LM}}$, and the neural network ensemble (NN_{ENS}) also produce very similar results to those for CSMT, which is why I left them out of Tab. 7.11.

[16] See, e.g.: http://www.statmt.org/moses/?n=Moses.Tutorial

- The Norma tool outputs confidence scores in the range of [0, 1] (Bollmann, 2012), although these cannot always be interpreted as probabilities, and the exact formula for deriving them differs between each of Norma's components.

For the neural network, I also transform the sequence probability to a log-probability score;[17] i.e., if y is the predicted normalization and y_i is the i-th character of that normalization, the final score $f(y)$ is calculated as:

$$f(y) = \log\left(\frac{\sum_i p_i(y_i)}{|y|}\right) \tag{7.2}$$

Table 7.12 shows the results of training a logistic regression classifier in the same manner as before, using these normalizer scores as features. For Norma_{M}, the wordlist mapping component of Norma, the score appears to be a pretty good predictor for incorrect normalizations in most cases, achieving an F-score of 0.69 or higher on most datasets. The best result is achieved for Hungarian, where more than 4,000 incorrect normalizations are identified with a precision of 0.97 and a recall of 0.89.

The confidence score for Norma_{M} is calculated as the relative frequency of the normalization over all instances of the historical source form. This means that it is related to the ambiguity score defined in Sec. 3.4.1; since Norma_{M} always chooses the most frequent normalization (MFN) for a given historical word form w, its confidence score can be expressed as follows (using the notation of Eq. (3.4)):

$$\text{score}(w) = \frac{\text{MFN}(w)}{c(w)} \tag{7.3}$$

It is therefore related to the ambiguity score $\alpha(w)$ via the following transformation:

$$\text{score}(w) = \frac{1}{2^{\alpha(w)}} \tag{7.4}$$

However, comparing the classifier performance for Norma_{M} in Tab. 7.12 with the datasets' ambiguity figures shown in Tab. 3.4, the numbers do not appear to correlate in any way.[18] This unintuitive result is actually a consequence of another property of the wordlist mapper: if the historical source form has not been seen in the training data, it is left unnormalized with $score(w) = 0$. Looking at the threshold the logistic regression classifier has learned, it turns out that it almost exclusively labels these unnormalized cases as incorrect (with the only notable exception being the German Anselm dataset, where about 170 normalization candidates with $score(w) > 0$ also fall in the "incorrect" category). In other words, the classifier performance rather correlates with the "ID accuracy" of a dataset which specifies how often, on average, the

[17] Log-probabilities worked slightly better than plain probabilities for the encoder–decoder model, and significantly better for the CSMT model. For the Norma scores, taking the logarithm of the confidence score produced worse results, so I am using the raw scores here.

[18] The ambiguity scores in Tab. 3.4 were calculated on the training set, but calculating them on the development set does not change the conclusion.

Dataset	Norma_{M}				$\text{Norma}_{\text{R+W}}$			
	Tokens	Precision	Recall	F-score	Tokens	Precision	Recall	F-score
DE_{A}	5,254	0.93	0.64	0.76	2,232	0.55	0.12	0.19
DE_{R}	1,475	0.80	0.69	0.74	0	–	0.00	–
EN	1,256	0.67	0.70	0.69	0	–	0.00	–
ES	895	0.62	0.61	0.61	0	–	0.00	–
HU	4,009	0.97	0.89	0.93	125	0.94	0.04	0.07
IS	696	0.69	0.44	0.54	0	–	0.00	–
PT	1,933	0.74	0.66	0.70	0	–	0.00	–
SL_{B}	1,179	0.81	0.85	0.83	0	–	0.00	–
SL_{G}	0	–	0.00	–	1,116	0.63	0.34	0.44
SV	537	0.64	0.94	0.77	106	0.59	0.17	0.27
Dataset	**CSMT**				$\text{CSMT}_{\text{+LM}}$			
	Tokens	Precision	Recall	F-score	Tokens	Precision	Recall	F-score
DE_{A}	1,543	0.66	0.21	0.32	2,959	0.83	0.40	0.54
DE_{R}	196	0.60	0.11	0.18	177	0.60	0.10	0.17
EN	173	0.59	0.13	0.21	184	0.59	0.13	0.22
ES	40	0.62	0.04	0.08	81	0.54	0.08	0.14
HU	285	0.58	0.12	0.19	300	0.56	0.12	0.20
IS	71	0.49	0.04	0.08	43	0.49	0.03	0.05
PT	124	0.73	0.07	0.13	268	0.64	0.14	0.23
SL_{B}	101	0.68	0.15	0.24	66	0.65	0.09	0.16
SL_{G}	106	0.37	0.05	0.09	107	0.36	0.05	0.09
SV	72	0.61	0.24	0.35	74	0.62	0.25	0.36
Dataset	NN_{Ens}				$\text{NN}_{\text{Ens+F}}$			
	Tokens	Precision	Recall	F-score	Tokens	Precision	Recall	F-score
DE_{A}	1,817	0.71	0.26	0.39	1,843	0.71	0.28	0.40
DE_{R}	136	0.77	0.09	0.17	498	0.86	0.36	0.51
EN	88	0.68	0.07	0.13	297	0.85	0.30	0.45
ES	6	1.00	0.01	0.02	156	0.90	0.21	0.34
HU	214	0.76	0.11	0.19	1,064	0.87	0.47	0.61
IS	19	0.84	0.02	0.04	126	0.80	0.12	0.20
PT	119	0.76	0.06	0.12	413	0.85	0.26	0.39
SL_{B}	18	1.00	0.03	0.06	239	0.92	0.35	0.51
SL_{G}	9	1.00	0.01	0.02	1,183	0.93	0.63	0.75
SV	0	–	0.00	–	114	0.93	0.39	0.55

Table 7.12: Precision, recall, and F-score of a logistic regression classifier on detecting incorrect normalizations, based on the normalizer-specific confidence or probability score of a normalization candidate; “Tokens” gives the number of normalizations classified as incorrect.

Dataset	$\text{Norma}_{\text{ALL}}$			CSMT			$\text{NN}_{\text{ENS+F}}$		
	ΔLen	Leven	Score	ΔLen	Leven	Score	ΔLen	Leven	Score
DE_{A}	0.15	0.11	**0.36**	0.00	0.01	**0.34**	0.06	0.12	**0.41**
DE_{R}	0.32	0.24	**0.40**	0.01	0.03	**0.18**	0.19	0.25	**0.52**
EN	0.30	0.26	**0.33**	0.06	0.04	**0.21**	0.21	0.21	**0.49**
ES	0.01	0.00	**0.28**	0.00	0.00	**0.08**	0.01	0.04	**0.42**
HU	0.32	0.18	**0.43**	0.00	0.00	**0.23**	0.21	0.20	**0.60**
IS	0.20	0.11	**0.22**	0.00	0.02	**0.12**	0.07	0.14	**0.27**
PT	0.02	0.03	**0.40**	0.00	0.00	**0.22**	0.02	0.05	**0.45**
SL_{B}	**0.53**	0.35	0.23	0.00	0.04	**0.30**	0.21	0.21	**0.54**
SL_{G}	0.38	0.55	**0.64**	0.00	0.03	**0.12**	0.39	0.51	**0.75**
SV	0.37	0.29	**0.52**	0.00	-0.01	**0.36**	0.26	0.36	**0.58**
Average	*0.26*	*0.21*	***0.38***	*0.01*	*0.02*	***0.22***	*0.16*	*0.21*	***0.50***

Table 7.13: Matthews correlation coefficient for predicting correct/incorrect normalizations of selected normalizers, evaluated per dataset and feature (ΔLen = absolute string length difference; Leven = Levenshtein distance; Score = the normalizer-specific confidence or probability score); best result for each dataset and normalizer highlighted in bold.

correct normalization is actually the unnormalized source form—this also explains the high F-score for Hungarian, which scores lowest by far using this measure (cf. Tab. 3.3).

Considering the other normalizers in Table 7.12, the classifier performance is usually much worse, with the occasional exception of individual datasets; e.g., for $\text{CSMT}_{\text{+LM}}$, the classifier achieves an F-score of 0.54 on German/Anselm, while the average F-score on the remaining datasets is only 0.18. For the neural network model, the classifier consistently achieves a notably higher F-score in the setting with filtering ($\text{NN}_{\text{ENS+F}}$) compared to the one without filtering (NN_{ENS}). Lexical filtering has the effect of reducing the average probability scores of normalizations, since if it changes the prediction, this means that (at least) the candidate with the highest probability has been filtered out. Moreover, if the correct target form is not covered by the lexicon, it is more likely that the normalization generated by the $\text{NN}_{\text{ENS+F}}$ model has a low probability score. This, in turn, could result in incorrect normalizations coinciding with low scores more often, explaining the better classifier performance in this scenario.

7.6.3 Conclusion

The evaluations in the previous sections have shown that string length difference and Levenshtein distance are poor predictors of incorrect normalizations, while scores reported by the normalizers work better in some instances. However, precision/recall/F-score for predicting incorrect normalizations do not consider the "true negative" rate, i.e., the precision of actually predicting correct normalizations as correct. Therefore, for a final comparison of all three criteria, I choose to calculate the *Matthews correlation coefficient* (Powers, 2011), which captures the correlation between the actual and the predicted labels.

Table 7.13 shows the Matthews coefficients for three of the normalizers, with the best result for each dataset and normalizer highlighted in bold. A score of 0 indicates that there is no correlation, i.e., the classifier does not perform better than random chance, while a perfect prediction would achieve a maximum value of 1. Negative values indicate disagreement between predicted and actual labels.

Using the normalizer scores as features consistently beats the simpler string-based criteria, no matter which normalizer is used, reinforcing the results from the previous sections. In general, error prediction is worst on CSMT for all types of predictors, works better on the combined components of $\text{Norma}_{\text{ALL}}$, and best with $\text{NN}_{\text{ENS+F}}$ using its probability score as feature. However, even in the latter case, performance varies substantially among datasets, with a Matthews coefficient between 0.27 (for Icelandic) and 0.75 (for Slovene/Gaj). In other words, while there is a correlation between normalizer scores and the correctness of the predictions, it is not consistent across datasets and—in most cases—not particularly strong.

Overall, these results suggest that predicting errors in the manner presented here might be useful, e.g., to highlight potentially problematic word forms for human annotators in a manual correction step during corpus creation, but should probably not be relied upon to find erroneous normalizations in a fully automatic fashion.

7.7 Error distribution

Most evaluation measures, including word accuracy and character error rate, reduce the performance of a model to a single number. This means they focus solely on the *quantity* of errors that a system makes, without regard for their *distribution*. Hypothetically, two models that both achieve a word accuracy of 50% could arrive at this score with a completely disjoint set of predictions—e.g., the first model could normalize only the first half of the dataset correctly while the second model is only correct on the second half. More generally, two models can achieve an equal or similar accuracy/CER/etc. with different subsets of correct normalizations.

The question of how the sets of correct (or incorrect) predictions overlap is interesting for a few reasons. First of all, it is an indicator of how the models differ in what they have learned and what kind of phenomena they can handle. Furthermore, if two models differ significantly in their error distributions, it might be beneficial to combine them in some way to take advantage of their individual strengths.[19] This is the principle of the Norma tool, after all, which combines three different normalization components to achieve a better performance than any component would achieve in isolation (Bollmann, 2013a); a similar argument can be made for the neural network ensembling technique. On the other hand, if the predictions of two models match almost everywhere, they can be considered equivalent for all practical purposes.

[19] Indeed, Hämäläinen et al. (2018) find that combining different normalization approaches is highly beneficial; their result was only published after the original submission of this thesis.

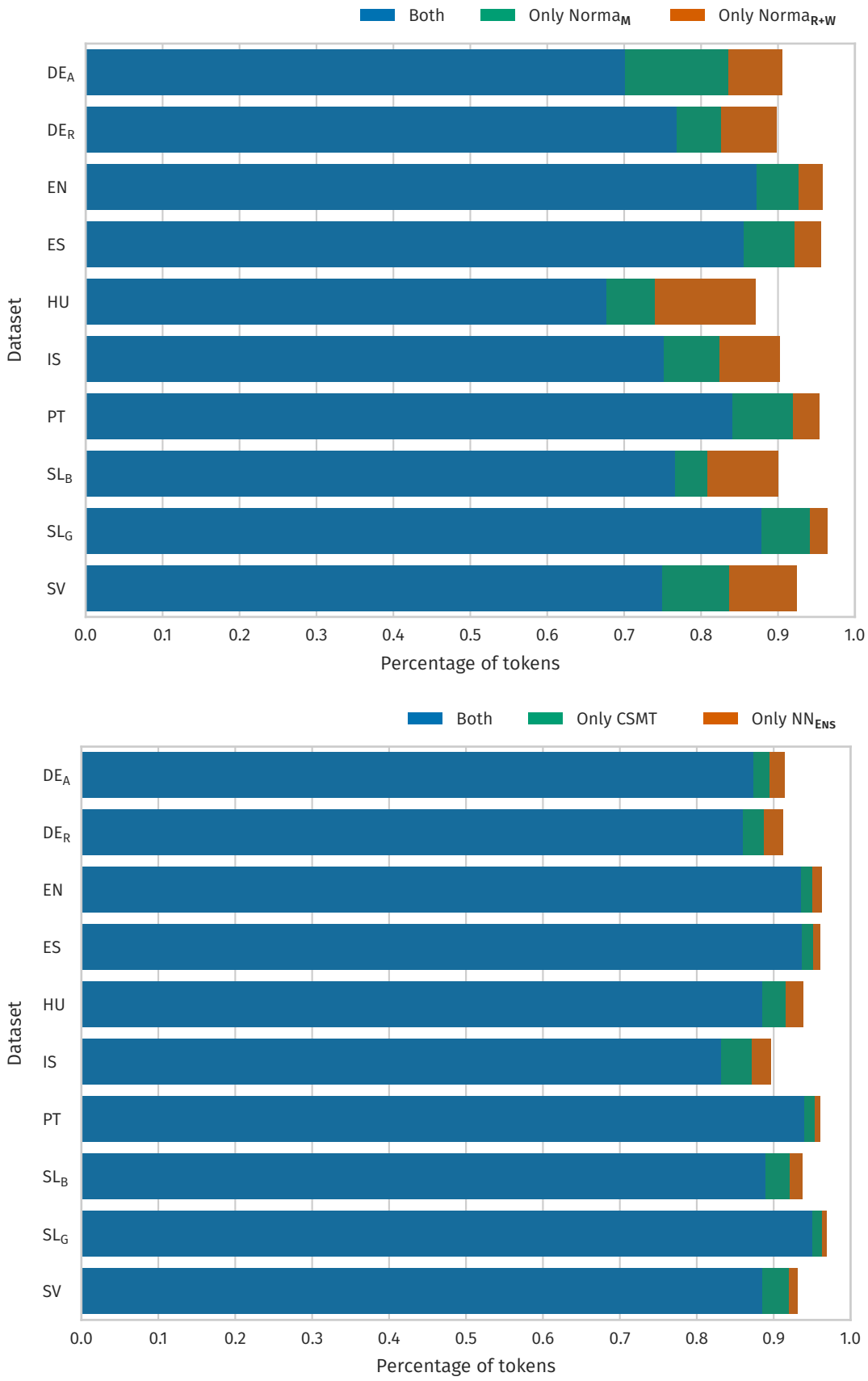


Figure 7.3: Comparison of two normalizers with regard to the subset of tokens that are correctly normalized by either both or only one of them; top graph compares Norma$_{M}$ and Norma$_{R+W}$, bottom graph compares CSMT and NN$_{ENS}$.

	Norma_{M}	$\text{Norma}_{\text{R+W}}$	$\text{Norma}_{\text{All}}$	CSMT	$\text{CSMT}_{\text{+LM}}$	NN_{Ens}	$\text{NN}_{\text{Ens+F}}$
Norma_{M}	–	7.18%	1.50%	1.10%	1.21%	1.46%	2.29%
$\text{Norma}_{\text{R+W}}$	6.58%	–	0.85%	1.78%	2.04%	2.42%	1.34%
$\text{Norma}_{\text{All}}$	5.73%	5.68%	–	1.62%	1.78%	2.34%	1.21%
CSMT	7.59%	8.86%	3.88%	–	0.69%	2.40%	3.65%
$\text{CSMT}_{\text{+LM}}$	7.43%	8.87%	3.77%	0.43%	–	2.33%	3.59%
NN_{Ens}	7.14%	8.70%	3.78%	1.59%	1.79%	–	2.50%
$\text{NN}_{\text{Ens+F}}$	7.02%	6.67%	1.71%	1.89%	2.10%	1.56%	–

Table 7.14: Percentage of tokens that are normalized correctly by the row normalizer, but not the column normalizer, averaged across all datasets

Figure 7.3 shows the distribution of correct normalizations between pairs of normalizers, i.e., the percentages of tokens that are normalized correctly by either both models, only the first model, or only the second model.[20]

The first graph compares Norma_{M} and $\text{Norma}_{\text{R+W}}$ in this way, since the combination of Norma's components is already known to be beneficial. Indeed, the percentage of tokens that only one of Norma_{M} or $\text{Norma}_{\text{R+W}}$ gets correct[21] is relatively high, ranging from approximately 8% (e.g., on English and Slovene/Gaj) to 20% (e.g., on German/Anselm and Hungarian). Compare this to the word accuracy evaluation in Tab. 7.1: here, both systems achieve a very similar accuracy on Icelandic (82.39% vs. 82.93%) and Swedish (83.65% vs. 83.61%), while the combined $\text{Norma}_{\text{All}}$ system is better by about 3–4 pp. Fig. 7.3 shows that about 15% of tokens for Icelandic and 18% for Swedish are normalized correctly by only of these two components, providing an explanation for the accuracy improvement when they are combined.

The second graph of Fig. 7.3 compares the CSMT system with the neural network ensemble. If this comparison showed similar trends, this could be seen as an indication that combining these two approaches might also lead to better performance. However, the proportions are considerably smaller here, with the highest percentage of correct, but non-overlapping normalizations being about 7% (on Icelandic). The subsets of normalizations only predicted correctly by NN_{Ens} are particularly small, suggesting that the neural network model does not improve much on tokens that CSMT cannot handle. Unfortunately, this result provides little justification for attempting to combine these two approaches, suggesting that both learn rather similar transformations instead.

Table 7.14 provides an overview of all possible normalizer pairs. For reasons of simplicity, I only show the average percentage of correct, non-overlapping tokens across all datasets; more precisely, the given numbers reflect the percentage of tokens that only the row normalizer gets correct (but the column normalizer gets incorrect). One way to interpret these numbers is to see them as the maximum accuracy improvement that the row normalizer could potentially contribute when combined with the column normalizer.

[20] Consequently, the difference of these combined sets to 100% is the percentage of tokens that both models normalize incorrectly.

[21] In other words, the combined green and orange subsets in Fig. 7.3.

Indeed, the combination of Norma_{M} and $\text{Norma}_{\text{R+W}}$ shows the highest amount of synergy when both directions are considered: combining the potential improvement of Norma_{M} over $\text{Norma}_{\text{R+W}}$ (7.18%) and vice versa (6.58%) results in a proportion of 13.76% of tokens that only one of the two normalizers gets correct—the highest value among all normalizer combinations. At the same time, both the CSMT and the neural network normalizers would provide a higher percentage of "newly correct" normalizations to either of Norma's components.

The potential contribution of the wordlist mapping (Norma_{M}) to CSMT/$\text{CSMT}_{\text{+LM}}$ or the NN_{Ens} model, however, is comparatively low (< 1.5%). This suggests that it is less beneficial to use the simple mapping approach together with either of these systems, as both CSMT and the neural network already correctly normalize most of the tokens that could be provided by the mapper—or, at least, considerably more than the $\text{Norma}_{\text{R+W}}$ system does. Also, the given percentage is only the *maximum* improvement that could be achieved by combining two methods; in practice, using the wordlist mapping technique in the same fashion as Norma—i.e., using the prediction provided by Norma_{M} when the historical token is contained in the wordlist, and the prediction from CSMT/NN_{Ens} otherwise—is likely to result in a much lower improvement.

The $\text{NN}_{\text{Ens+F}}$ column (in Tab. 7.14) shows slightly higher percentages than NN_{Ens}, however, this is more likely to be a result from more incorrect predictions caused by the lexical filter. It would be more sensible to limit or remove the use of the filter first before considering a combination with other normalizers. The combination of the two CSMT systems has the two lowest scores in the comparison, again showing that their predictions are very similar with or without the additional language modeling data.

7.8 Summary

This chapter provided an extensive evaluation and comparison of the neural network ensemble with the previously established CSMT model and the Norma tool (Sec. 7.1), analyzing the strengths and weaknesses of each approach.

I discussed different approaches to evaluating historical normalization, comparing the word accuracy measure with the popular character error rate (CER) metric (Sec. 7.2). I argued that CER by itself does not add much value over plain word accuracy, but that applying CER to the subset of *incorrect* normalizations can provide useful insight into the output of the models. One such observation was that incorrect predictions made by the models were often worse (in terms of CER) than their unnormalized historical forms. This prompted me to investigate methods to detect normalization candidates that are likely to be wrong, but I did not find a promising approach to achieve this (Sec. 7.6).

A manual error classification study on a small subset of the German and English datasets suggested that most normalizations counted as "incorrect" are actually reasonably good (Sec. 7.3), showing the need for better evaluation methods. Stemming—i.e., comparing the normalization output with its reference normalization on the basis of their word stems—can be a potentially useful approach here, as it filters out errors resulting purely from, e.g., wrong inflectional forms (Sec. 7.4). When using an automatic stemmer, this may of course also result in false

positives due to mistakenly conflated word forms, but in my opinion, it still constitutes an improvement over using word accuracy or CER alone.

Comparing different normalization methods, character-based statistical machine translation (CSMT) achieved the highest word accuracy on almost all datasets, with the exception of German (Anselm) where the neural network ensemble was slightly better. More in-depth analysis showed that this is mostly due to superior performance of tokens that were not seen in the training data ("unknowns"); in other words, better generalization. On the subset of known tokens, using the wordlist mapping component of Norma resulted in the highest accuracy on eight of the datasets, suggesting that this simple lookup method should be a part of any normalization system (Sec. 7.5).

While the evaluation showed that CSMT outperforms an extensively tuned, state-of-the-art neural network ensemble, I also observed that the neural model seems to generalize better on previously unseen character alignments (Sec. 7.5.2). Further analysis suggested that the neural network might be improved by a stronger language modelling component (Sec. 7.5.3). Ensembling the different systems, on the other hand, did not appear to be promising (Sec. 7.7).

Based on the experiments in this chapter, my recommendation for obtaining a solid historical normalization pipeline is to combine a simple wordlist mapping algorithm with character-based statistical machine translation (CSMT). For evaluation, I recommend evaluating word accuracy, character error rate (CER) on *incorrect* tokens, and also comparing automatically stemmed word forms.

Chapter 8

Multi-task learning

> *[Y]ou learn to play tennis in a world that tasks you to learn many other things. You also learn to walk, to run, to jump, to exercise, to grasp, to throw, to swing, to recognize objects, to predict trajectories, to rest, to talk, to study, to practice, etc. [...] Perhaps the similarities between the thousands of tasks you learn are what enable you to learn any one of them—including tennis.*
>
> — Caruana (1993)

So far, all machine-learning approaches discussed here had one property in common: they are trained on a single (historical) language at a time. In this chapter, I investigate ways to train on multiple languages simultaneously, with the aim of increasing the model's capabilities for generalization and improving performance in low-resource scenarios.[1]

The concept of *multi-task learning (MTL)* is usually traced back to Caruana (1993, 1997). Its main idea is summarized thusly:

> Multitask learning is an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of *related* tasks as an inductive bias. It does this by learning tasks in parallel while using a shared representation; what is learned for each task can help other tasks be learned better. (Caruana, 1997)

MTL has been applied successfully to a variety of NLP tasks. Collobert et al. (2011) demonstrate its effectiveness for part-of-speech tagging, chunking, named entity recognition, and semantic role labeling; Klerke et al. (2016) use eye-tracking data to improve sentence compression models; Plank (2016) uses keystroke logs to improve shallow syntactic parsing; etc. Closer to the learning scenario considered here, MTL has also been used to train machine translation models on multiple languages simultaneously (e.g., Dong et al., 2015; Luong, Le, et al., 2015). For historical normalization, Bollmann and Søgaard (2016) apply MTL to multiple source texts of the German Anselm corpus, while Bollmann et al. (2017) use grapheme-to-phoneme transduction as an auxiliary task with encoder–decoder models.

Here, I will focus on the multi-language scenario, training the encoder–decoder models described in Chapter 6 on multiple of the historical datasets (from Sec. 3.1) in parallel. For practical

[1] Since the original time of writing this thesis, the main findings of this chapter have also been published in Bollmann et al. (2018).

reasons, I will only experiment with pairwise combinations of these datasets.[2] This results in more data points to compare and allows for a cleaner analysis of the interaction between datasets; for example, can the normalization accuracy for historical Spanish be improved by pairing it with Hungarian data, or is it better to use a closely related language like Portuguese? Note that "datasets" do not equate "languages" in our case, since some languages—German and Slovene—are represented in more than one dataset. However, pairing these datasets can also be insightful, as the two German datasets follow different normalization guidelines and cover different text genres (cf. Sec. 3.1.2) and the Slovene datasets provide data from the same language at different historical stages.

Sec. 8.1 will introduce the MTL architectures that will then be compared on a subset of the datasets in Sec. 8.2. Sec. 8.3 will present a more extensive evaluation of the best MTL architecture on all datasets.

8.1 Models

In this work, I am concerned with multi-task learning using the encoder–decoder architecture (cf. Chapter 6). However, there are several ways to utilize this architecture in a MTL setting. While the key idea is always to share many (if not all) model parameters between the separate tasks, implementations can differ in which layers are shared and how exactly the different tasks—or, in our case, datasets—are distinguished.

I will consider three main variants here:

1. a multi-task learning setup that shares the full encoder and decoder, but keeps separate prediction layers for each dataset ($\text{MTL}_{\text{Split}}$, Sec. 8.1.1);
2. a multi-task learning setup that shares all parameters, but distinguishes datasets by feeding a special input symbol into the encoder ($\text{MTL}_{\text{Input}}$, Sec. 8.1.2); and
3. a single encoder–decoder model that is jointly trained on multiple datasets without any means of distinguishing between them (Joint, Sec. 8.1.3).

The third variant is not strictly a MTL setup at all, but rather just combines the training splits of several datasets and trains a single-task model with minor modifications (cf. Sec. 8.1.3); it exists primarily as a "control group" for the other two models, and can help determine whether potential improvements stem from the MTL-specific modifications or rather just from the amalgamation of the training datasets.

Further variants are possible: Luong, Le, et al. (2015) propose to use task-specific encoders and decoders, e.g., using one encoder per source language and one decoder per target language in a multi-lingual machine translation setting. However, here the normalization task differs from machine translation in that we are almost never interested in normalizing a historical language to anything but its contemporary equivalent, so there is usually a one-to-one relation between

[2] I will assume the use of exactly two datasets in many of the examples and explanations to follow, but all presented concepts can be applied to three or more datasets as well.

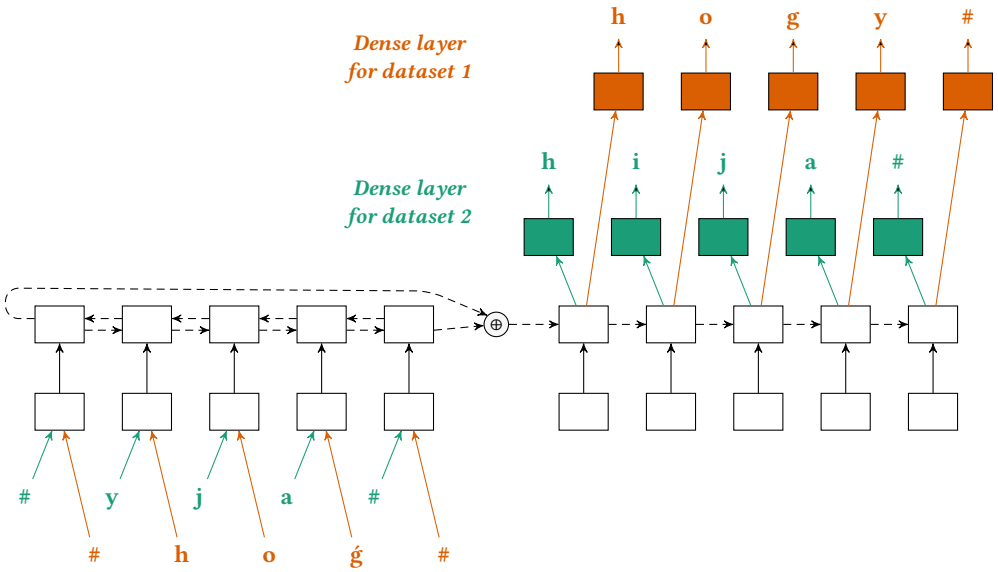


Figure 8.1: Multi-task learning using the encoder–decoder model (cf. Fig. 6.2) with separate prediction layers ($\text{MTL}_{\text{SPLIT}}$); orange components are specific to dataset 1 (here: Hungarian), green components are specific to dataset 2 (here: Spanish), other components are shared between the datasets; decoder inputs not shown for reasons of clarity.

source and target data.[3] In my preliminary testing, I also found it disadvantageous to keep the full encoder or decoder layers dataset-specific, so I do not investigate these variants further here.

8.1.1 $\text{MTL}_{\text{SPLIT}}$: Using separate prediction layers

A common approach for MTL is to share all parts of a model except for the final prediction component (Goldberg, 2017, p. 240 f.). In the encoder–decoder model discussed here, this means sharing everything except the final dense layer; instead, there is now one dense layer *per dataset* that is specific to that dataset. Figure 8.1 shows a visualization of this MTL model. Essentially, this setup requires the encoder, decoder, and embedding layers to learn dataset-independent representations, while only the final prediction layer can learn dataset-specific properties. This is identical to the approach used in Bollmann et al. (2017).

Training is done on all of the involved datasets simultaneously. More precisely, when training with a batch size of 50, each training step consists of processing 50 samples from each of the datasets, then performing a parameter update based on the joint loss function. For example, if L_{es} and L_{hu} are the loss functions (cf. Sec. 5.3.1) for the Spanish and the Hungarian parts of

[3] A many-to-one setting is also conceivable, e.g., by treating the two Slovene datasets as separate source languages (since they were written in different alphabets) that should be normalized to the same target language, i.e., modern Slovene. However, we would not want to normalize, e.g., Old Spanish to Modern Hungarian, as that could no longer be modeled as normalization, but is rather a proper translation task.

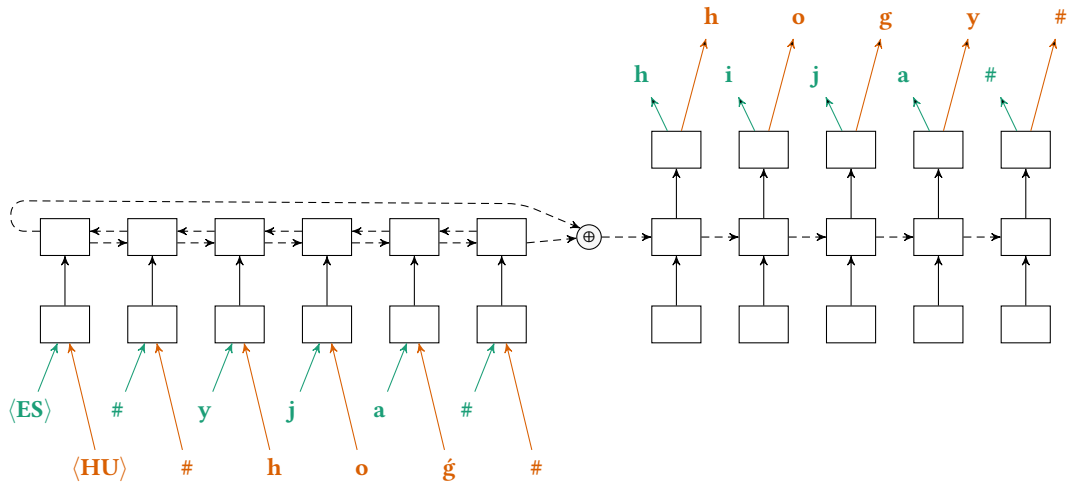


Figure 8.2: Multi-task learning using the encoder–decoder model (cf. Fig. 6.2) with task-specific input symbols ($\text{MTL}_{\text{INPUT}}$); all model components are shared, and each encoder input is prefixed with a symbol identifying which dataset will be processed (here: ⟨ES⟩ for Spanish, ⟨HU⟩ for Hungarian); decoder inputs not shown for reasons of clarity.

the model, respectively, the loss function for the multi-task model trained on both datasets in parallel is:

$$L(\hat{y}_{es}, \hat{y}_{hu}, y_{es}, y_{hu}) = L_{es}(\hat{y}_{es}, y_{es}) + L_{hu}(\hat{y}_{hu}, y_{hu}) \tag{8.1}$$

As a consequence, each training update is based on an equal amount of training samples from each involved dataset. Since datasets will typically have different numbers of samples in total, "epochs" are now defined as having seen a fixed number of samples from each one; here, 50,000 is typically used. After training has finished, separate models are saved for each dataset including only the components necessary for processing this dataset; i.e., prediction and evaluation works exactly the same way as in the single-task setup.

Early stopping (cf. Sec. 5.3.6) is still used in the multi-task scenario. After each training epoch, the model is validated against the held-out validation sets from each dataset, and snapshots of the model's state are saved independently for each dataset if its validation accuracy improved. This means that even if the ideal number of epochs to reach the highest accuracy is different for the datasets, only the best state for each dataset will be used in the end. Training ends only after the validation accuracy for *each* dataset has stopped improving.

8.1.2 $\text{MTL}_{\text{INPUT}}$: Using input identifiers

The second MTL variant is inspired by recent work in cross-lingual morphological knowledge transfer (Kann et al., 2017; Jin and Kann, 2017). The model here is the same encoder–decoder model as in the single-task setup, but the inputs are prepended with special symbols serving as "dataset identifiers". Figure 8.2 shows a visualization of this approach.

On first glance, this might not seem very different from the $\text{MTL}_{\text{SPLIT}}$ variant described above, as the only difference in the model is that there is only a single prediction layer now. However, since the knowledge of the dataset is now encoded in the inputs, all parts of the model—including the encoder and decoder—have access to this knowledge and can, in principle, learn to condition their behavior based on which dataset they are processing. In the $\text{MTL}_{\text{SPLIT}}$ variant, on the other hand, all components before the final prediction layers are forced to learn dataset-independent transformations, since they have no way of distinguishing between the datasets.

Training works the same way as for the $\text{MTL}_{\text{SPLIT}}$ model from Sec. 8.1.1. However, as this is technically implemented as a single-task model, the training data is preprocessed to ensure the same balanced composition of samples from each dataset as in the $\text{MTL}_{\text{SPLIT}}$ variant. Early stopping and model snapshots, however, are no longer done on a per-dataset basis, but are conditioned on the average accuracy of the involved datasets.

8.1.3 Joint training

Finally, I consider a variant that trains a single-task encoder–decoder model jointly on multiple datasets. This is essentially a kind of baseline for the other multi-task setups, and serves to discern the effect of combining the training datasets from that of the MTL-specific modifications described in Secs. 8.1.1 and 8.1.2. In other words, if this JOINT model performed equally well as the $\text{MTL}_{\text{SPLIT}}$ and $\text{MTL}_{\text{INPUT}}$ models, the added complexity of those MTL variants would not be beneficial, and any potential improvements would result only from the combination of the training datasets.

To make the comparison fairer, the same balancing approach is used for the training samples as in Sec. 8.1.2; i.e., with a batch size of 50, for each batch, 50 training samples *per dataset* are selected and used in the calculation of the training update. This ensures that the training results are not distorted by an unbalanced representation of the datasets in the training data sequence. Early stopping also works the same way as in Sec. 8.1.2.

8.2 Model comparison

The three MTL variants described in the previous sections—$\text{MTL}_{\text{SPLIT}}$, $\text{MTL}_{\text{INPUT}}$, and JOINT—can be combined with either the base or attentional encoder–decoder model (cf. Sec. 6.1). Even though Sec. 6.3.2 already compared these two encoder–decoder variants and found the attentional model to be superior, it is possible that the same does not hold true when training in a multi-task setting. For example, Bollmann et al. (2017) found that combining MTL with attention led to an overall decrease in accuracy. Therefore, I choose to evaluate both the base and attentional variants in this comparison.

As mentioned previously, I will focus on *pairwise* combinations of datasets. This means that in each MTL scenario, the model is trained on two datasets at the same time, and the resulting model(s) are evaluated on both of these datasets separately. I choose to perform this comparison on the same reduced selection of datasets as described in Sec. 6.2.1, and for mainly the same reasons as well: training all pairwise combinations of ten datasets in six different configurations

is computationally expensive, and the uniform size of the datasets removes the potential effect of different training data sizes from the evaluation. I also train the respective single-task models on each of these datasets in order to quantify the relative increase or decrease in accuracy from using the MTL setups.

Results

Figure 8.3 shows the percentage error reduction or increase (compared to the single-task model) for each of the tested configurations; Table 8.1 summarizes the results by giving the absolute single-task performance as well as the best and average change from the different dataset pairings, either as relative change in percentage points (Tab. 8.1a) or as percentage change of the error rate (Tab. 8.1b).

The results show a clear trend: in general, the $\text{MTL}_{\text{Split}}$ variant performs better than the $\text{MTL}_{\text{Input}}$ variant, while both perform considerably better than the simple Joint training scenario. In fact, the Joint models show a consistent *decrease* in accuracy; on average, training in this fashion decreases accuracy by 1.06 pp (base model) and 1.42 pp (attentional model) compared to just training on the respective dataset alone. Compared to that, the $\text{MTL}_{\text{Split}}$ model achieves an average *increase* of 0.33 pp (base model) and 0.49 pp (attentional model), corresponding to an error reduction of 2.35% and 3.75%, respectively. The $\text{MTL}_{\text{Input}}$ approach lies in between the other two, but averages a slight decline in accuracy overall.

The individual datasets behave quite differently in terms of their average improvement from multi-task learning. The German and English datasets generally seem to profit most; e.g., English achieves an error reduction of up to 13.7% (Fig. 8.3, attentional model with $\text{MTL}_{\text{Split}}$, when trained jointly with Hungarian). The highest reduction overall is observed for Slovene/Gaj, which achieves 15.8% in the same constellation. On the other hand, the Hungarian dataset consistently performs worse in all MTL setups except one, the attentional model with $\text{MTL}_{\text{Split}}$. In other constellations, such as Icelandic or Slovene/Gaj in the $\text{MTL}_{\text{Split}}$ and $\text{MTL}_{\text{Input}}$ scenarios, the results vary between improvements and losses.

In general, the impact of MTL appears to depend more on the dataset that is being evaluated than the dataset it is paired with, or at least the results are mostly inconclusive in this regard. For example, depending on the model configuration, the German/Anselm dataset profits most from being trained together with Hungarian ($\text{MTL}_{\text{Split}}$/base), Hungarian and English ($\text{MTL}_{\text{Split}}$/attentional), English ($\text{MTL}_{\text{Input}}$/attentional), or Slovene ($\text{MTL}_{\text{Input}}$/base and Joint/base). When training Hungarian together with Icelandic, the accuracy changes range from minor improvements (-3%, $\text{MTL}_{\text{Split}}$/attentional) to significant losses (+23.8%, $\text{MTL}_{\text{Input}}$/attentional); similar effects can be observed for other pairings as well, such as Icelandic with English (-5.9% on $\text{MTL}_{\text{Split}}$/base, but +15.4% on $\text{MTL}_{\text{Input}}$/attentional).

Interpretation

Compared to the inherent variance of the training process (analyzed in Sec. 6.3.1, and particularly Tab. 6.1), many of the changes observed here seem relatively minor. Combined with the highly varied results, this makes it difficult to establish a correlation between MTL performance

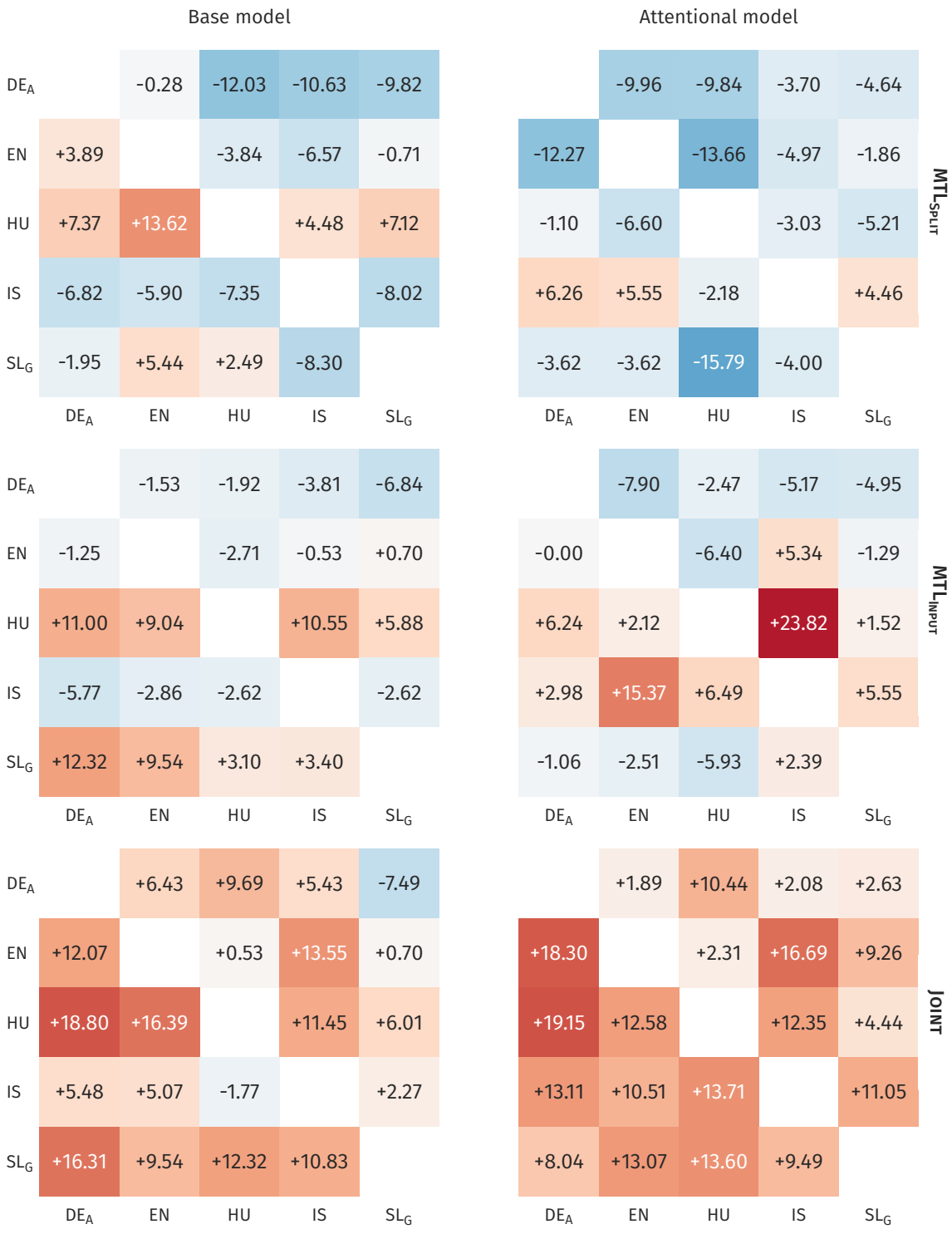


Figure 8.3: Percentage change of error of the multi-task models compared to the single-task setup for pairwise training experiments; numbers are for evaluation of the row dataset when trained together with the column dataset. Negative scores (highlighted in blue) are improvements, positive scores (highlighted in red) are increases of the error rate.

Dataset	Single	Maximum change			Average change		
		$\text{MTL}_{\text{Split}}$	$\text{MTL}_{\text{Input}}$	Joint	$\text{MTL}_{\text{Split}}$	$\text{MTL}_{\text{Input}}$	Joint
Base Model							
DE_{A}	78.76%	**+2.28**	+1.36	+1.48	**+1.57**	+0.72	-0.87
EN	88.64%	**+0.70**	+0.30	-0.06	**+0.19**	+0.11	-0.87
HU	85.92%	-0.66	-0.88	-0.90	-1.27	-1.42	-2.19
IS	82.76%	**+1.28**	+0.94	+0.30	**+1.13**	+0.58	-0.51
SL_{G}	93.74%	**+0.48**	-0.20	-0.66	**+0.02**	-0.49	-0.88
Average	*85.96%*	***+0.82***	*+0.30*	*+0.03*	***+0.33***	*-0.10*	*-1.06*
Attentional Model							
DE_{A}	79.24%	**+1.88**	+1.52	-0.40	**+1.35**	+1.01	-0.96
EN	89.02%	**+1.32**	+0.66	-0.26	**+0.81**	+0.05	-1.51
HU	87.08%	**+0.80**	-0.20	-0.60	**+0.49**	-1.35	-1.84
IS	85.02%	**+0.32**	-0.46	-1.76	-0.57	-1.28	-2.07
SL_{G}	94.28%	**+0.78**	+0.32	-0.50	**+0.35**	+0.10	-0.72
Average	*86.92%*	***+1.02***	*+0.37*	*-0.70*	***+0.49***	*-0.29*	*-1.42*

(a) Single-task accuracy and relative change of accuracy (in percentage points)

Dataset	Single	Maximum change			Average change		
		$\text{MTL}_{\text{Split}}$	$\text{MTL}_{\text{Input}}$	Joint	$\text{MTL}_{\text{Split}}$	$\text{MTL}_{\text{Input}}$	Joint
Base Model							
DE_{A}	21.24%	**-10.73%**	-6.40%	-6.97%	**-7.39%**	-3.39%	+4.10%
EN	11.36%	**-6.16%**	-2.64%	+0.53%	**-1.67%**	-0.97%	+7.66%
HU	14.08%	+4.69%	+6.25%	+6.39%	+9.02%	+10.09%	+15.55%
IS	17.24%	**-7.42%**	-5.45%	-1.74%	**-6.55%**	-3.36%	+2.96%
SL_{G}	6.26%	**-7.67%**	+3.19%	+10.54%	**-0.32%**	+7.83%	+14.06%
Average	*14.04%*	***-5.84%***	*-2.14%*	*-0.21%*	***-2.35%***	*+0.71%*	*+7.55%*
Attentional Model							
DE_{A}	20.76%	**-9.06%**	-7.32%	+1.93%	**-6.50%**	-4.87%	+4.62%
EN	10.98%	**-12.02%**	-6.01%	+2.37%	**-7.38%**	-0.46%	+13.75%
HU	12.92%	**-6.19%**	+1.55%	+4.64%	**-3.79%**	+10.45%	+14.24%
IS	14.98%	**-2.14%**	+3.07%	+11.75%	+3.81%	+8.54%	+13.82%
SL_{G}	5.72%	**-13.64%**	-5.59%	+8.74%	**-6.12%**	-1.75%	+12.59%
Average	*13.08%*	***-7.80%***	*-2.83%*	*+5.35%*	***-3.75%***	*+2.22%*	*+10.86%*

(b) Single-task error and percentage change of error

Table 8.1: Comparison of multi-task learning models on the reduced datasets; "Single" gives the performance of the single-task encoder–decoder model; other columns give the "maximum" or "average" change for MTL from all evaluated dataset pairs. Highest improvements highlighted in bold.

and properties of the datasets (e.g., as discussed in Sec. 3.4). The combined results for all evaluated dataset pairs suggest that the attentional model with the $\text{MTL}_{\text{SPLIT}}$ variant is the most promising approach overall, as it has the highest average error reduction and the attentional model already performs better than the base model in the single-task scenario. Beyond that, it would be desirable to base conclusions about specific dataset pairings on results from multiple training runs or even ensembles (as in Sec. 6.3.1), to make sure the observed effect is not simply a result of random variation. I will come back to this below in the full evaluation.

Admittedly, my implementation of the $\text{MTL}_{\text{SPLIT}}$ setup does have one advantage over the $\text{MTL}_{\text{INPUT}}$ variant: the former can save different model snapshots for each of the two datasets, while the latter only saves a single model state based on the *average* performance across both datasets. This is done for technical reasons, as the $\text{MTL}_{\text{INPUT}}$ variant is essentially just a single model that processes different types of input data. However, a more sophisticated implementation could conceivably perform a per-dataset validation for this variant, too, and save separate "best" model states for each dataset just as with the $\text{MTL}_{\text{SPLIT}}$ variant. It could be worthwhile to investigate whether the slightly worse performance of $\text{MTL}_{\text{INPUT}}$ is mainly due to this difference in training.

Another thing to note is that the combination of $\text{MTL}_{\text{SPLIT}}$ and the attentional model does not generally result in reduced accuracy (cf. Tab. 8.1), but rather helps on average, contrary to the result reported in Bollmann et al. (2017). This could be due to a variety of factors, such as the size of the training sets—around 5,400 on average in Bollmann et al. (2017), while the experiments here use 50,000 tokens—or the nature of the auxiliary task—Bollmann et al. (2017) use grapheme-to-phoneme conversion, while this work uses normalization on a different dataset and (usually) language. More research is needed here to arrive at a conclusion.

8.3 Full evaluation

In the previous section, I compared several types of MTL models on a reduced selection of datasets, and found that the $\text{MTL}_{\text{SPLIT}}$ approach with an attentional model provided the highest improvements on average. However, for some datasets or dataset combinations, the effect sizes were relatively small and the results highly varied. Therefore, I choose to perform a more extensive evaluation of this particular MTL model that is more comparable to the single-task setup evaluated in Sec. 6.3. This means:

1. Training and evaluating pairwise combinations of all datasets described in Sec. 3.1 and using the full datasets instead of reduced subsets.
2. Performing multiple training runs for each dataset combination and combining them to form a model ensemble, as this was shown to yield substantial improvements for the single-task setup (cf. Sec. 6.3.3).
3. Using beam search decoding, as this was shown to outperform greedy decoding for the single-task setup (cf. Sec. 6.3.4).

Since training the $\text{MTL}_{\text{SPLIT}}$ models is much more computationally expensive than training the single-task ones, I only train three individual models for each configuration (instead of

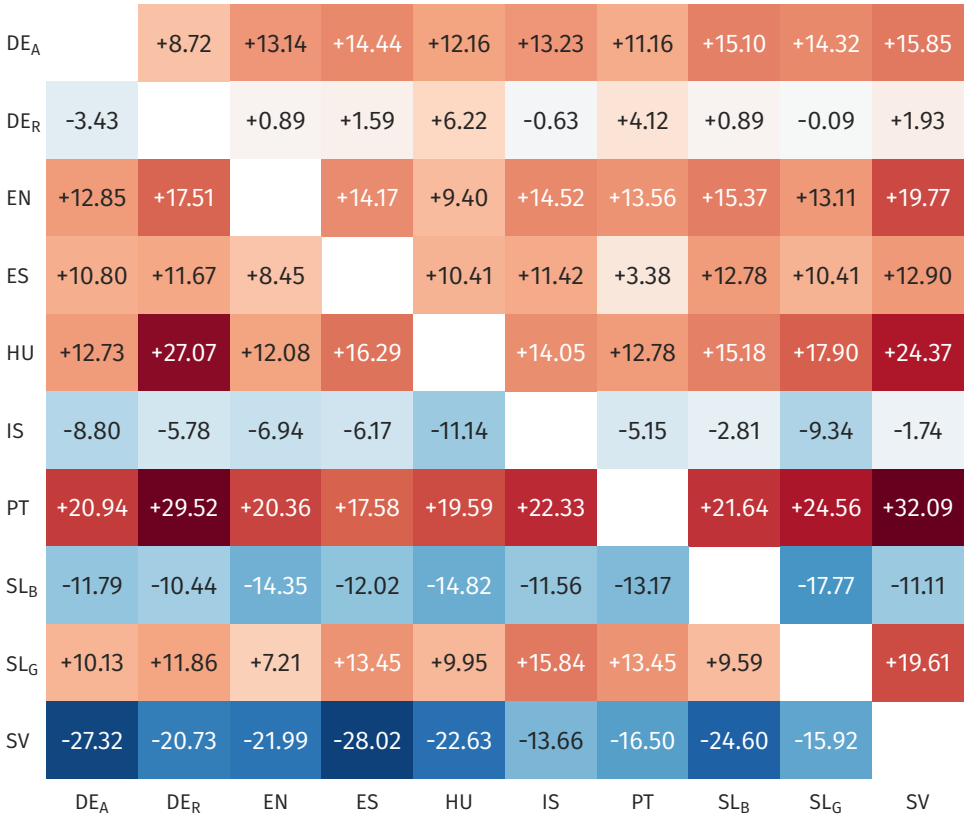


Figure 8.4: Percentage change of error of the $\text{MTL}_{\text{SPLIT}}$ ensemble with attention compared to the single-task setup for pairwise training experiments; numbers are for evaluation of the row dataset when trained together with the column dataset. Negative scores (highlighted in blue) are improvements, positive scores (highlighted in red) are increases of the error rate.

the five models trained in Sec. 6.3.3).[4] Since the training sets are now of different sizes, I define one epoch to consist of 50,000 tokens; i.e., training iterates over each training dataset independently (by shuffling it and going through all of its tokens once, then re-shuffling it and going through all tokens again, etc.), and whenever the model has seen 50,000 tokens of each dataset, validation is performed to decide if the training should be stopped (cf. the description of early stopping in Sec. 5.3.6). As before, model snapshots are saved independently for each dataset (cf. p. 148).

Figure 8.4 shows the relative change of error for the $\text{MTL}_{\text{SPLIT}}$ ensemble (of three models) compared to the single-task encoder–decoder ensemble (of five models) when both are evaluated using beam search decoding.[5] Two results become immediately apparent: (i) the usefulness of multi-task learning depends more on the dataset that is being evaluated than the one it is

[4] For comparison, training the single-task encoder–decoder model with attention *five times* on each dataset took about 62 hours, while training the corresponding $\text{MTL}_{\text{SPLIT}}$ model *three times* on each dataset combination took about 321 hours, or almost two weeks.

[5] I also compared the *average* scores of the individual models without ensembling, and the overall trends were the same.

trained together with; and (ii) for most datasets, the multi-task setup is now *detrimental* rather than beneficial.

One hypothesis about multi-task learning is that its usefulness directly correlates with either synergistic or complementary properties of the datasets. In other words, it is conceivable that the performance on one dataset improves most with an MTL setup when it is paired with another dataset that is either (i) very similar, or (ii) provides an additional signal that is useful for, but not covered in, the first dataset. The results in Fig. 8.4 show that there can indeed be considerable variation depending on the exact dataset combination; e.g., the error reduction on Swedish ranges from 13.66% (when trained jointly with Icelandic) to 28.02% (when trained jointly with Spanish). At the same time, the question whether MTL helps at all appears to depend most on the dataset being evaluated: German/RIDGES is the only dataset that shows both declines and improvements in the error rate—depending on which dataset it is paired with—while for all other datasets, the error rate either *always* improves or *always* worsens.

The second result is especially surprising after the first evaluation in Sec. 8.2, where the $\text{MTL}_{\text{SPLIT}}$ model with attention performed best on average and led to improvements—i.e., a reduction of errors compared to the single-task model—in 17 of the 20 dataset combinations it was evaluated on. In contrast, the evaluation in Fig. 8.4 shows that MTL leads to consistent improvements only for the Icelandic, Slovene/Bohorič, and Swedish datasets, and decreases performance in all other cases (with the occasional exception on German/RIDGES).

Considering the dataset statistics in Tab. 3.1, it appears that the *size of the training corpus* is the most likely reason for these results. This also explains the stark contrast of this evaluation to the one in the previous chapter, where the size of the training sets was held constant throughout all experiments. The three corpora that benefit from MTL are also among the four with the smallest training sets, each with less than about 50,000 tokens; the fourth corpus in that group is German/RIDGES, the only other dataset where MTL is beneficial in at least some scenarios. Moreover, the dataset that profits most from MTL, Swedish, is the one with the smallest training set of all (ca. 25,000 tokens). In the same vein, the dataset that shows the highest error *increase* with MTL, Portuguese, has one of the largest training sets (ca. 220,000 tokens). On the other hand, German/Anselm performs slightly less worse than Portuguese even though its training set is larger, so while there is a correlation between MTL performance and training set size, this does not appear to be the only decisive factor.

We can also consider individual dataset pairings and ask whether their performance is correlated with the similarity of these datasets (as defined in Sec. 3.4.2). Fig. 8.4 by itself already shows the intuitive result that pairings of identical or closely related languages result in the best performance:[6] this is true for Spanish/Portuguese as well as German Anselm/RIDGES, and also for the Bohorič part of the Slovene corpus when trained together with the Gaj part (though not the other way around). This coincides with the similarity scores of these datasets from Fig. 3.2. Slovene/Gaj, on the other hand, performs best when paired with English, which is the dataset it is most similar to according to the comparison in Fig. 3.3. However, calculating Pearson's correlation coefficient on either of these similarity matrices with the error change matrix from Fig. 8.4 shows no correlation in the general case ($|r| < 0.1$).

[6] "Best" here always means "lowest percentage change of error", even when that change is positive, i.e., the error actually increased.

Overall, the evaluation here suggests that multi-task learning tends to help most when the training set is small, and can actually lead to a decline in accuracy otherwise. The choice of the "auxiliary" dataset to pair it with seems to be less important in comparison, although choosing data from related languages is the most promising approach.

Chapter 9

Low-resource training

> *For neural networks to generalize well, there generally must be a large amount of data[…] Deep learning currently […] works best when there are thousands, millions or even billions of training examples[.]*
>
> — Marcus (2018)

The historical datasets used here for training and evaluating automatic normalization systems are, to some extent, exceptional: the smallest dataset, Swedish, has about 56,000 manually normalized tokens (25,000 of which are used for training), while the largest one, German/Anselm, has more than 325,000 tokens (cf. Tab. 3.1). These corpora can conceivably be reused when the goal is to normalize new texts in one of the covered languages, but for many languages, digitized historical texts with gold-standard normalization annotations are not easily available. Producing these normalizations manually—in order to train one of the normalization systems presented in this thesis—is a time-consuming process. Therefore, a relevant question for these application scenarios is how the systems perform with smaller amounts of training data.

The evaluation of multi-task learning suggested that the size of the training corpus is a significant factor for its performance, with smaller training sets benefiting more from an MTL approach than larger ones (cf. Sec. 8.3). This raises two questions: (i) Does the encoder–decoder approach in general perform better with larger amounts of training data, or in other words, is its performance significantly reduced with training sets that are small? (ii) Can we use multi-task learning (with auxiliary datasets from other languages) to obtain a better performance even with a small training set in our main language?

To simulate a low-resource training scenario, I create new training sets for each historical corpus by taking only the first 5,000 tokens from their respective original training sets; the development and test sets are not changed. The choice of using 5,000 tokens is somewhat arbitrary; ultimately, this number was chosen as a compromise between making the sets significantly smaller than any of the "full" training sets (5,000 tokens is about a fifth of the smallest training set) and still keeping them large enough for a supervised learning algorithm to extract meaningful generalizations out of them. The tokens were taken from the beginning of the datasets—instead of being chosen at random—in order to better resemble a real application scenario, where it is more likely that full texts or text passages are normalized manually to obtain training data instead of a random selection of words from the full corpus.

The evaluation here mainly repeats parts of the analyses performed on the full datasets, with the aim to find parallels or differences in the low-resource scenario. Sec. 9.1 investigates if there

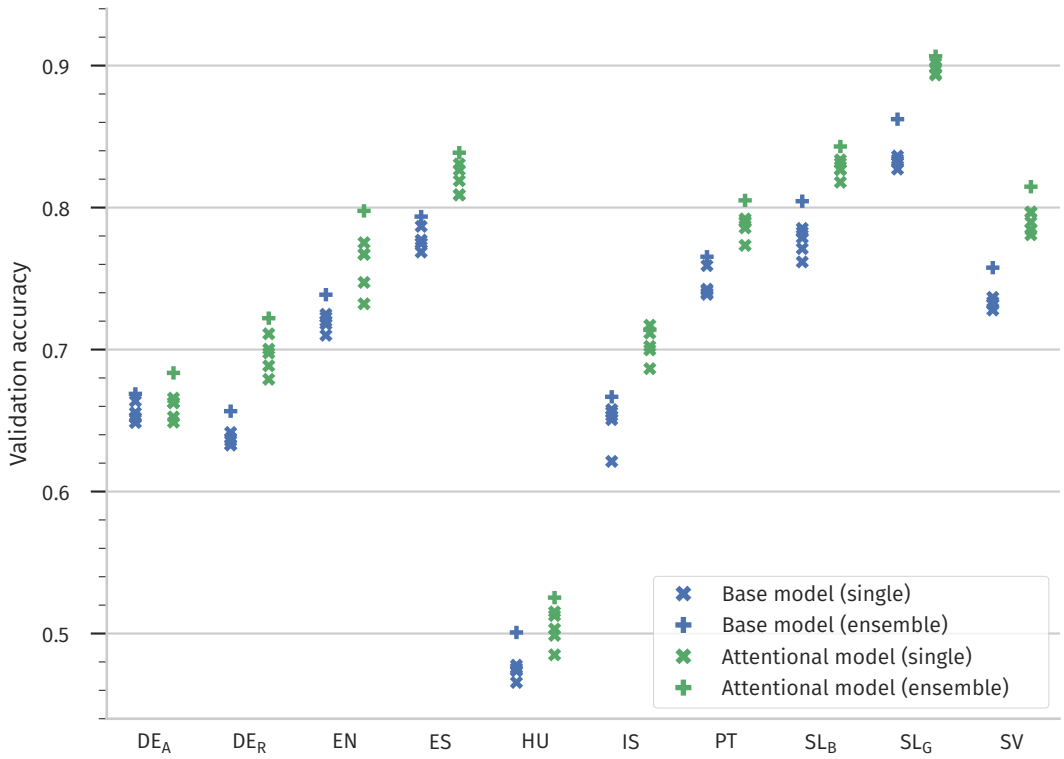


Figure 9.1: Validation accuracy of individual models and model ensembles in the low-resource scenario, trained on five different initializations per dataset and model type

is a higher variance among different random restarts and if ensembling is still advantageous. Sec. 9.2 compares the different normalization methods. Sec. 9.3 evaluates whether multi-task learning can be particularly beneficial with small datasets, while Sec. 9.4 sums up the findings.

9.1 Variance and ensembling

The analysis of the encoder–decoder model in Sec. 6.3 showed that (i) different random restarts of the training procedure can yield different results (in terms of test accuracy), and (ii) combining these independently trained models to form a model ensemble substantially improves on any single model's result. While low-resource training should behave similarly in this regard, the *variance* of the individual models' results might be greater, due to the training process being potentially less stable with lower amounts of data. Therefore, I choose to replicate some of the analyses from Sec. 6.3 for the low-resource scenario.

Figure 9.1 shows the validation accuracy of the individual models (analogous to Fig. 6.9) and the ensembles, while Table 9.1 presents the average accuracy and sample standard deviation of the individual runs as well as the ensemble accuracy (analogous to Tabs. 6.1 and 6.2). Naturally, the average accuracy scores are much lower, with the worst performance being observed on Hungarian (ca. 50% with the attentional model compared to almost 89% on the full training set; cf. Tab. 6.1) and the best on Slovene/Gaj (90% vs. 95%). At the same time, the variance of

Dataset	Base model			Attentional model		
	Avg.	s	Ensemble	Avg.	s	Ensemble
DE_A	65.44%	0.5756	66.88%	65.85%	0.7328	68.36%
DE_R	63.61%	0.3656	65.66%	69.53%	1.2130	72.21%
EN	71.91%	0.5729	73.86%	75.78%	1.7631	79.77%
ES	77.65%	0.6562	79.36%	81.88%	0.9915	83.86%
HU	47.34%	0.4643	50.07%	50.30%	1.2095	52.53%
IS	64.78%	1.5079	66.67%	70.35%	1.1775	71.40%
PT	74.44%	0.8358	76.54%	78.62%	0.7537	80.51%
SL_B	77.59%	0.9604	80.44%	82.73%	0.6045	84.30%
SL_G	83.24%	0.3414	86.22%	89.73%	0.3422	90.66%
SV	73.25%	0.3221	75.77%	78.97%	0.7130	81.47%

Table 9.1: Statistics for the low-resource scenario over five independent training runs per dataset and model type; "Avg." gives the average word accuracy from all training runs, s denotes the sample standard deviation, and "Ensemble" gives the word accuracy from the ensemble of all five runs.

these scores indeed appears to be larger: while Tab. 6.1 reported standard deviations no greater than 0.73, with most being below 0.5, the low-resource scenario produces results with standard deviations mostly *above* 0.5, up to a maximum of 1.76.

Comparing the base encoder–decoder model (without attention) to the model with the attention mechanism, we can see that the attentional variant clearly outperforms the non-attentional one on every dataset except German/Anselm. This was not the case in the analysis using the full training sets: there, the attentional model was significantly better on only four of the datasets (cf. Sec. 6.3.2). Here, on the other hand, the differences between the two variants are statistically significant with $p < 0.01$ for all datasets except Anselm. This is an unintuitive result: the attention mechanism adds complexity to the model, e.g., by way of additional weights that need to be trained, while in a low-resource scenario, there is usually not enough data to inform a large amount of parameters. The results here show that 5,000 tokens are already sufficient for the attention mechanism to outperform the base model, though.

Finally, the ensembling approach is also highly beneficial in the low-resource scenario. The accuracy of the model ensemble is consistently higher than the average accuracy—and, in most cases, also higher than the best accuracy—of the individual models. Overall, it is advisable to use a model ensemble whenever possible, especially since performing multiple training runs is less computationally expensive in the case of smaller training sets.

9.2 Comparative evaluation

The preliminary evaluation in Tab. 7.1 compared different normalization methods and found that, overall, the CSMT approach yielded the best results. It is not clear if the same effect holds in the low-resource scenario, since normalizers might behave differently when given only low

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
$Norma_M$	54.85	64.02	80.08	81.37	33.23	65.17	78.41	65.04	88.59	75.90
$Norma_{R+W}$	65.82	76.25	85.54	86.33	59.21	73.42	82.40	82.23	85.61	77.82
$Norma_{ALL}$	69.70	77.11	**86.00**	88.05	60.26	75.25	**85.86**	83.26	85.90	80.58
CSMT	70.35	78.46	84.92	87.17	54.76	**77.26**	84.03	86.47	**92.09**	**85.12**
$CSMT_{+LM}$	70.32	**78.49**	84.85	87.36	54.89	77.23	83.92	**86.49**	92.06	85.08
NN_{AVG}	66.39	69.89	76.26	82.03	50.78	70.79	78.50	82.94	89.82	79.44
NN_{ENS}	69.46	72.97	80.16	84.04	53.22	71.71	80.29	84.57	90.52	81.56
NN_{ENS+F}	**74.49**	77.82	85.80	**88.15**	**65.97**	76.62	85.02	86.20	88.68	83.25

Table 9.2: Word accuracy of different normalization methods (cf. Sec. 7.1) in the low-resource scenario, in percent, evaluated on the development sets of the historical datasets; best result for each dataset highlighted in bold.

amounts of training data. In particular, neural networks are often said to perform best with a high amount of training data, while the Norma tool is specifically designed to learn from very small samples of manual normalizations.

Table 9.2 compares the word accuracy of different normalizers when trained on the 5,000-token training sets. Norma's results are much stronger in this scenario compared to the full evaluation of Tab. 7.1; $Norma_{ALL}$ achieves the highest accuracy on English and Portuguese, outperforms the NN_{AVG} model on all datasets except one (Slovene/Gaj), and is also better than CSMT and NN_{ENS} on several datasets. This confirms Norma's strengths in learning from sparse amounts of training data.[1]

The cSMTiser tool also shows a good performance overall, achieving the best result on half of the datasets. Furthermore, it outperforms the neural networks without lexical filtering—NN_{AVG} and NN_{ENS}—on every dataset, without exception. Clearly, the CSMT approach can handle low-resource scenarios better than the neural networks can. Admittedly, it is possible that the hyperparameter settings derived in Sec. 6.2 are not optimal for this case, and that the neural networks might perform better with different settings; however, no parameter tuning was done for cSMTiser at all, so the latter certainly shows better performance "out of the box", which can be an important consideration for practical applications.

Comparing CSMT to $CSMT_{+LM}$, the scenario that augments the contemporary language model with additional training data from the modern datasets, there is again no clear advantage for either approach. It seems that either the language modeling component is not as critical, or the 5,000 tokens for training are already enough to learn a good character-based language model. The analysis in Sec. 7.5.3 suggested that the language model might indeed be a crucial factor of CSMT's performance, making the second explanation appear more likely.

[1] Indeed, Bollmann (2013a) showed that Norma can already perform reasonably well with about 100–500 tokens for training, so possibly, if the size of the training set was reduced even further, Norma's advantage over the other methods would become even more pronounced.

While the encoder–decoder model does not perform well in this evaluation, adding the lexical filtering step usually results in a significant boost in word accuracy. On many datasets, lexical filtering increases the accuracy by about 5 pp, with the biggest increase being observed for Hungarian (66% for $\text{NN}_{\text{Ens+F}}$ vs. 53% for NN_{Ens}; this is particularly remarkable because accuracy goes down with filtering in the full evaluation; cf. Fig. 7.1). Consequently, the $\text{NN}_{\text{Ens+F}}$ model has the highest accuracy of all methods on three datasets: German/Anselm, Spanish, and Hungarian. Slovene/Gaj is the only dataset where the filtering step leads to a slight decrease in accuracy (from 90.52% to 88.68%).

Apparently, lexical filtering is a significant aid for the decoding step of the encoder–decoder model. In a way, the filter can be seen as modifying the "recurrent" part of the decoder: some output classes (at timestep t) are filtered out based on the previously predicted normalized characters (at timesteps $0, \ldots, t-1$). The fact that this leads to such a significant improvement suggests that the neural network has learned the input–output correspondences (of historical characters to normalized characters) better than it has learned the shape of a valid target word form, i.e., the "language modeling" part of the task.

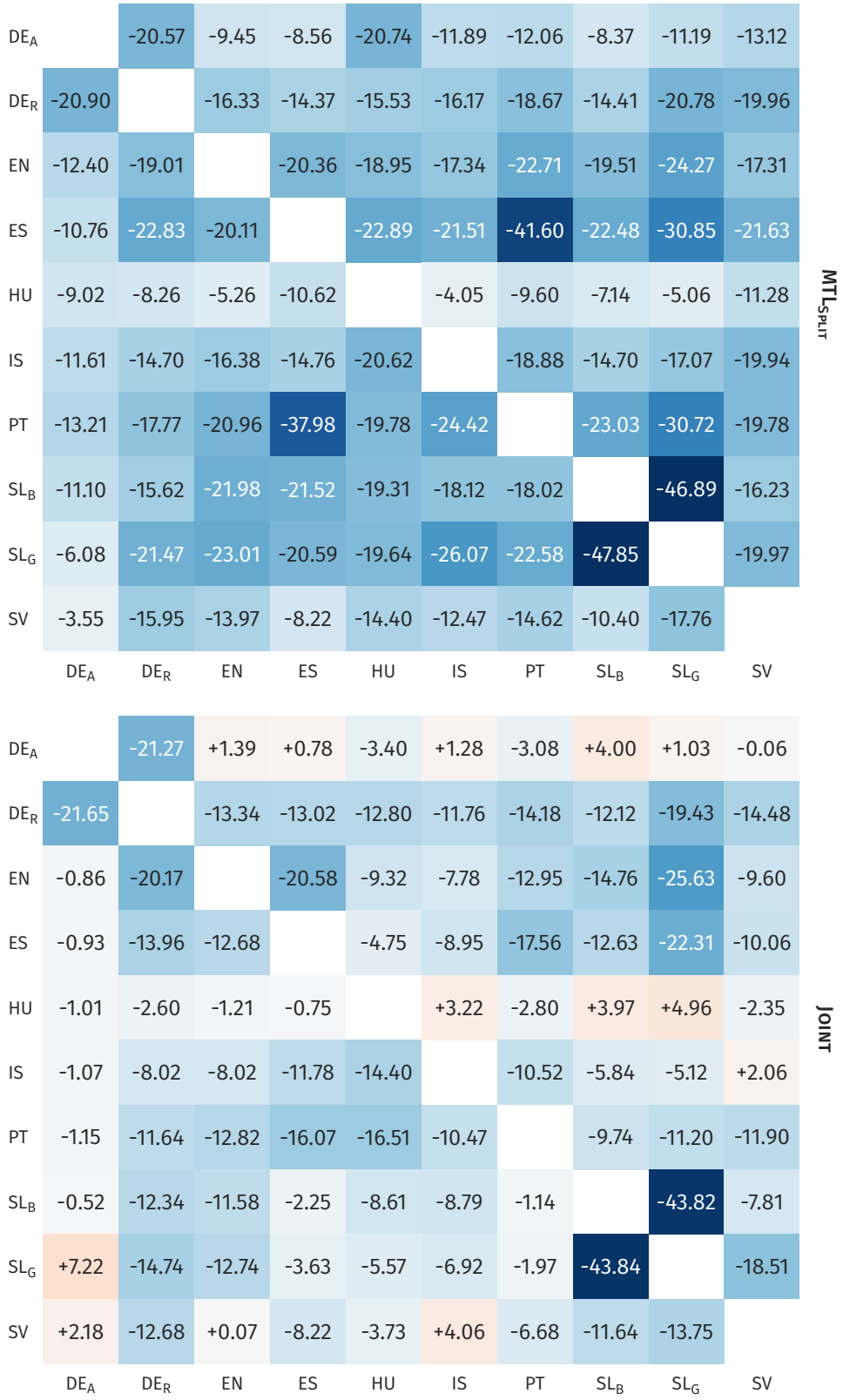
$\text{Norma}_{\text{R+W}}$ uses the same modern lexical resources to generate and filter its predictions as $\text{NN}_{\text{Ens+F}}$ does, yet in some cases, it performs significantly worse. This shows that the neural network model can often utilize the lexical resource better than Norma, which in turn suggests that it has learned better input–output transformations from the training data. Again, this comparison suggests (similar to the analysis in Sec. 7.5.3) that the neural network model is mainly restricted by a comparatively weak modeling of the output character transitions, i.e., the character-based language modeling on the normalized word forms.

9.3 Multi-task learning

Chapter 8 introduced the concept of multi-task learning (MTL) and applied it to training on two historical datasets simultaneously. The evaluation in Sec. 8.3 showed that the datasets that profited the most from training together with another dataset were those with lower amounts of training data. This suggests that MTL might be particularly beneficial in the low-resource scenario, where training sets are particularly small.

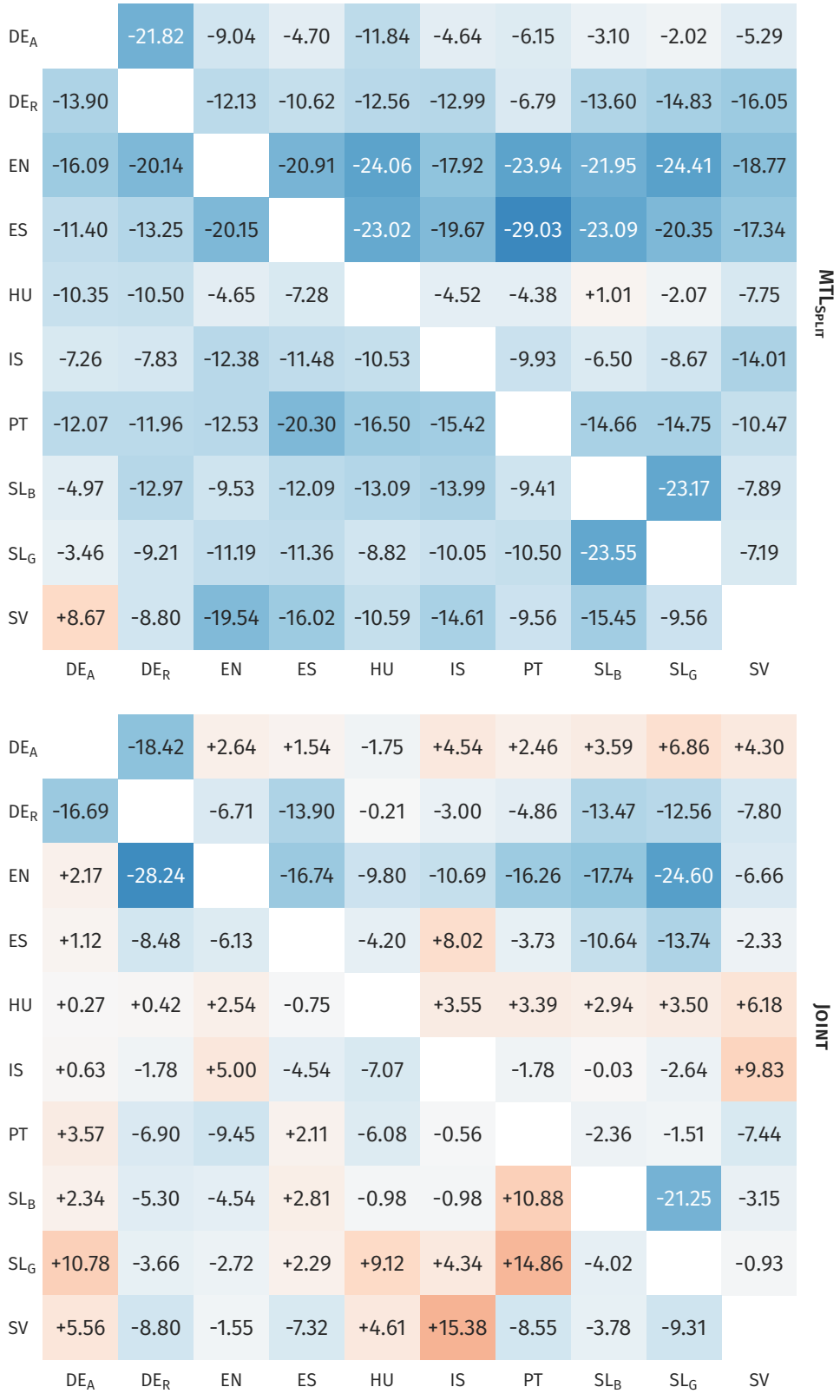
To investigate this, I once more train and evaluate on all pairwise combinations of historical datasets, with the change that the dataset being evaluated only uses its low-resource training set instead of the full one. In other words, the pairings are now asymmetrical: a low-resource training set is combined with the full training set of another dataset, and the resulting model is evaluated only on the former dataset. This procedure resembles a reasonable application scenario: if a researcher is interested in normalizing a language for which no manually normalized resource exists, they could conceivably create a small batch of manual normalizations for this language and then combine it with a corpus in another language (e.g., one of the corpora evaluated here) using a MTL setup.

Training in this asymmetrical setup works exactly as described in Sec. 8.3, except that the batch size is now set to 5,000 (the size of the low-resource training sets). To recap, this means that the training algorithm always sees an equal amount of tokens from both datasets, and while tokens are presented in a random order, no part of a training set is used twice before all of its



(a) Base model

Figure 9.2: Percentage change of error of the multi-task models (cf. Sec. 8.1) compared to the average of the single-task models for pairwise training experiments in the low-resource scenario; numbers are for evaluation of the row dataset (using the low-resource training set) when trained together with the column dataset (using the full training set).



(b) Attentional model

Figure 9.2: Percentage change of error of the multi-task models (cont.); negative scores (highlighted in blue) are improvements, positive scores (highlighted in red) are increases of the error rate.

tokens have been seen. For example, if a low-resource training set (of 5,000 tokens) is paired with an auxiliary set of 50,000 tokens, training for 10 epochs means that the model will have seen each token from the low-resource set exactly 10 times and each token from the auxiliary set exactly once.

Furthermore, I will again compare both the encoder–decoder model with and without attention as well as the $\text{MTL}_{\text{Split}}$ variant (cf. Sec. 8.1.1) and the simple Joint training (cf. Sec. 8.1.3). This is because it is particularly conceivable with low amounts of training data that the simpler joint training approach—i.e., simply combining the small training set with a larger one without any MTL technique—is already beneficial, and my aim is to discern the effect of the training data combination from that of the MTL-specific modifications.

On the other side, I do not reconsider the $\text{MTL}_{\text{Input}}$ variant and I do not train model ensembles, even though the latter was shown to be beneficial in almost all instances so far. Both choices are only due to computational constraints, not theoretical ones.[2]

9.3.1 Evaluation

Figure 9.2 shows the results of the low-resource MTL evaluation, again presented as the percentage change of error compared to the single-task model. Since the MTL models are now the result of a single training run (instead of an ensemble), I compare them to the *average* accuracy of the individual single-task model (i.e., NN_{Avg}). Also, in line with the other MTL evaluations performed so far, all models are compared here using greedy decoding instead of beam search.

Some immediate observations are that (i) $\text{MTL}_{\text{Split}}$ is consistently better than Joint; (ii) the base model improves more than the attentional one with either MTL approach; and (iii) with the possible exception of the Joint approach with the attentional model, MTL consistently improves the accuracy over the single-task models.

The first observation—$\text{MTL}_{\text{Split}}$ mostly outperforming the Joint training method—is consistent with previous results from Sec. 8.2 which showed the same trend. This mostly confirms that the modifications to the encoder–decoder model for multi-task learning as described in Sec. 8.1.1 are advantageous to a simple combination of the training datasets. However, some individual exceptions to this trend exist, mostly with closely related datasets: e.g., combining German/Anselm and German/RIDGES usually yields better results in the Joint setup than in the $\text{MTL}_{\text{Split}}$ one.

The observation that the base model sees greater improvements from MTL than the attentional one is to be expected: Sec. 6.3.2 showed that the attention mechanism often improves performance over the base model, with the consequence that the margin for further improvement via MTL is lower, simply due to the percentage of incorrect normalizations being lower. Still, even in the MTL settings evaluated here, the attentional model often performs better than the base model in terms of absolute word accuracy, on average showing an improvement of about

[2] Intuitively, it seems that training in a low-resource setup should be considerably faster, but this is not the case here: (i) since the low-resource set is combined with a full training set from another corpus, convergence of the model weights is now also influenced by the larger dataset, prolonging the training time; and (ii) due to the asymmetrical nature of this experimental setup, twice the amount of models have to be trained to evaluate each pairwise dataset combination compared to the previous MTL setups.

3 pp. The final evaluation in Chapter 10 will show and compare the actual word accuracy scores.

Lastly, the MTL approaches—in particular $\text{MTL}_{\text{SPLIT}}$—show consistent improvements over the single-task baseline. For the base model, $\text{MTL}_{\text{SPLIT}}$ improves over the single-task models in every single instance, while for the attentional model, there are only two dataset combinations where the error increases. Even though the JOINT training approach performs worse, it still often results in an error reduction over the single-task setup. This is a stark contrast to the "full" evaluation of $\text{MTL}_{\text{SPLIT}}$ in Fig. 8.4, where improvements where only seen for a few datasets. At the same time, it confirms the hypothesis these results suggested, which is that the size of the training set is a good predictor for improvements via multi-task learning. The small training sets indeed profit from MTL, mostly regardless of which other dataset they are paired with.

Furthermore, the improvements observed here are also of a higher magnitude than those in the full evaluation. In Fig. 8.4, the largest improvement from $\text{MTL}_{\text{SPLIT}}$ was observed for Swedish, with an error reduction of about 28%. With the low-resource setting, error reductions greater than 20% are not uncommon, and the highest observed percentage is almost 48% (for Slovene/Gaj). This is not completely unexpected, though, since the single-task baseline for low-resource training is naturally lower than that for training on the full datasets and, therefore, the room for improvement is bigger.

Considering individual dataset pairings, it is apparent that pairs of related datasets perform particularly well, more so than in the previous evaluations. This is true for Spanish and Portuguese, Slovene/Gaj and Slovene/Bohorič, and—to a lesser extent—also for German/Anselm and German/RIDGES. Conversely, Hungarian—which is the only Finno-Ugric language in the evaluation—generally profits the least from MTL in terms of error reduction. While there is considerable variance in the results for other dataset pairings as well, they are often not consistent: e.g., in the $\text{MTL}_{\text{SPLIT}}$ evaluation with the base model (cf. Fig. 9.2a), German/Anselm shows the best result when paired with Hungarian, but this effect disappears in the JOINT approach and in the attentional model. There is, unfortunately, no apparent correlation to the similarity scores from Sec. 3.4.2 either, besides the aforementioned cases of related languages.

9.4 Summary

The evaluation with a simulated low-resource scenario, using only 5,000 tokens from each dataset for training, has shown that—unsurprisingly—word accuracy is generally worse and results from several training runs of the encoder–decoder model show more variance. In the comparative evaluation, the Norma tool is stronger than before, while cSMTiser is still the best choice overall. The neural network approach is comparatively weakest in this setting, although the lexical filtering step mitigates this considerably. Furthermore, the option of using multi-task learning with a larger auxiliary dataset might make the neural network models more competitive, as it was shown to result in significant accuracy improvements. The final evaluation (in Chapter 10) will compare the performance of selected MTL models to the other systems when also adding beam search and lexical filtering.

Chapter 10

Evaluation

Evaluation is where your dreams are torn to shreds[.]
— Dan Simonson, http://blog.thedansimonson.com/?p=510

In previous chapters, all evaluations and analyses have been performed on the development sets of the historical corpora. This was done in order to preserve the test sets for a single, final evaluation that aims to confirm the key findings of the previous analyses.

Some of the normalizers also make use of the development data during training, making an evaluation on the same set of tokens potentially biased. More precisely, the cSMTiser system uses this data as the tuning set for the Moses decoder, while the neural network models use it to decide when to stop training (via early stopping; cf. Sec. 5.3.6). The held-out test sets, on the other hand, aim to provide an independent, less biased sample for the evaluation.[1]

First, I describe the methodology used for this final evaluation, i.e., which methods are compared and how their differences are assessed (Sec. 10.1). I compare all methods using the traditional word accuracy measure (Sec. 10.2), before performing the same evaluation using word stems instead of full word forms (Sec. 10.3), as this was found to be the most promising measure of normalization quality beyond the word level. I also seperately consider tokens that were "known" or "unknown" during training in order to evaluate the generalization capabilities of the systems (Sec. 10.4). Finally, I perform the same evaluations for the low-resource scenario presented in Chapter 9 to determine whether the same observations hold as on the full datasets (Sec. 10.5).

[1] Two notes about the independence/bias of the test sets:

1. All test sets originate from the same corpus as their associated training and development sets; this alone makes them not truly independent, as they are likely to consist of similar text genres, vocabulary, etc. However, it is rare to find two historical corpora of the same language that also (i) cover similar historical language stages, and (ii) provide gold-standard normalizations created with similar normalization guidelines. For this reason, a train/dev/test split of a single corpus is probably the best that can be done at the moment.

2. Normalization is performed on tokens in isolation, and the test set is likely to contain many of the same tokens as the training and development sets. However, it seems too artificial to remove all of these instances, as any real-world application scenario for a normalization system will face the same situation. Instead, the separate evaluation on "known" and "unknown" tokens can serve as an estimate for the models' abilities for memorization vs. generalization.

Evaluation Dataset		Auxiliary Dataset	
		Full	Low-resource
DE_A	German (Anselm)	German (RIDGES)	German (RIDGES)
DE_R	German (RIDGES)	German (Anselm)	Swedish
EN	English	Hungarian	Slovene (Gaj)
ES	Spanish	Portuguese	Portuguese
HU	Hungarian	English	German (RIDGES)
IS	Icelandic	Hungarian	Swedish
PT	Portuguese	Spanish	Spanish
SL_B	Slovene (Bohorič)	Slovene (Gaj)	Slovene (Gaj)
SL_G	Slovene (Gaj)	English	Slovene (Bohorič)
SV	Swedish	Spanish	English

Table 10.1: Dataset pairings for the test set evaluation of MTL models; for each dataset that is being evaluated ("Evaluation Dataset"), the MTL-NN_{Ens}/MTL-NN_{Ens+F} models were trained together with the dataset ("Auxiliary Dataset") that performed best in this combination according to Fig. 8.4 (for the Full datasets) or Fig. 9.2b (top; for the Low-resource datasets).

10.1 Methodology

All models are trained on the training sets and evaluated on the test sets of the historical corpora (cf. Sec. 3.1).

For all evaluations, I mostly compare the same methods and use the same denotations as introduced in Sec. 7.1. This includes the Norma tool in three different configurations ($Norma_M$, $Norma_{R+W}$, $Norma_{All}$), the cSMTiser tool (CSMT), optionally trained with additional language modeling data ($CSMT_{+LM}$), and the ensemble of five individually trained encoder–decoder models with attention (NN_{Ens}), optionally with lexical filtering (NN_{Ens+F}). These encoder–decoder models are identical to the ones in previous evaluations, i.e., they use the attention mechanism, beam search decoding, and the hyperparameters summarized in Sec. 6.2.5. I do not separately report the average scores of the individual encoder–decoder models.

Additionally, I include some of the models obtained via multi-task learning (MTL) (cf. Chapter 8) in the evaluation. The MTL_{Split} approach, i.e., using separate prediction layers for each of the two languages, generally performed best and will therefore be used here. To reduce the large amount of comparisons that result from all pairwise combinations of the datasets, I choose to only consider the *best pairing* for each dataset to be evaluated, determined from the preliminary evaluation on the development sets; i.e., Fig. 8.4 for the full datasets and Fig. 9.2b (top) for the low-resource scenario. Table 10.1 gives an overview of these pairings. All evaluations use the model ensembles with attention; in other words, the MTL models evaluated here are identical to the single-task models in architecture and hyperparameters except for the modifications specific to the MTL_{Split} setup.

In addition to reporting the individual normalizer's scores on the different datasets, I also perform tests to assess the magnitude of the differences between them. I follow Benavoli et al.

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
$Norma_M$	83.86	82.15	92.45	92.51	74.58	82.84	91.67	81.76	93.90	83.80
$Norma_{R+W}$	77.32	83.39	90.65	89.36	81.20	83.92	86.92	86.70	89.68	82.95
$Norma_{ALL}$	88.02	86.55	94.60	94.41	86.83	86.85	94.19	89.45	91.44	87.12
CSMT	88.82	88.06	95.21	95.01	91.63	**87.10**	95.09	93.18	95.99	91.13
$CSMT_{+LM}$	86.69	88.19	**95.24**	**95.02**	**91.70**	86.83	**95.18**	**93.30**	**96.01**	91.11
NN_{ENS}	89.16	88.07	94.80	94.83	91.17	86.45	94.64	91.61	95.19	90.27
NN_{ENS+F}	**89.78**	87.12	95.10	94.39	88.70	86.65	94.56	89.95	90.80	87.02
$MTL\text{-}NN_{ENS}$	87.61	**88.40**	94.31	94.50	89.36	86.76	93.43	92.76	94.78	**91.28**
$MTL\text{-}NN_{ENS+F}$	89.00	87.42	95.00	94.32	88.18	86.68	94.15	90.33	90.65	87.24
MFN	*94.64*	*96.46*	*98.57*	*97.40*	*98.70*	*93.46*	*97.65*	*98.71*	*98.96*	*98.97*

Table 10.2: Word accuracy of different normalization methods (cf. Sec. 7.1) on the test sets of the historical datasets, in percent; best result for each dataset highlighted in bold; bottom line gives the theoretical maximum accuracy obtainable when the most frequent normalization (MFN) was chosen for each historical type (cf. Sec. 3.4).

(2017) in using a *Bayesian signed-rank test* (Benavoli et al., 2014) to compare the performance of two methods.[2] Their approach compares the differences in accuracy scores between two methods and allows for a *range of practical equivalence (ROPE)*: an interval around 0 that defines when a difference between two methods is minor enough to be practically irrelevant. How to define this interval involves some degree of subjectivity in deciding what should be considered a relevant difference. For my comparisons, I choose to consider a difference of 0.5 pp or less between two accuracy scores as "practically equivalent"; i.e., the ROPE is $[-0.5, +0.5]$. The average size of the test sets is roughly 20,000 tokens; a difference of 0.5 pp between two methods will therefore, on average, represent about 100 tokens more that are normalized correctly by the better method.

The Bayesian signed-rank test uses the observed accuracy scores—or, rather, the difference between them—to estimate the probabilities of either method A being better than method B ($p(A > B)$), method B being better than A ($p(B > A)$), or the two methods being equivalent within the chosen ROPE ($p(A \approx B)$). For further details, see Benavoli et al. (2017, Sec. 4.2).

10.2 Accuracy

Table 10.2 shows the word accuracy on the test sets for different normalization methods. The trends are mostly identical to those observed in the development set evaluation (in Tab. 7.1): the cSMTiser tool provides the highest accuracy for most datasets and consistently outperforms the encoder–decoder models, with the exception of the German/Anselm dataset. In general, the results are very close to those of the earlier evaluation, with more than half of the accuracy

[2] I use the Python implementation provided by the authors at https://github.com/BayesianTestsML/tutorial.

scores on the test sets being within a range of ±0.5 pp to their counterparts on the development sets.

The MTL evaluation also follows the same trends that were observed in Fig. 8.4: datasets that show an error reduction from MTL in Fig. 8.4 also achieve a higher accuracy in the test set evaluation (compared to the single-task models), while those that show an error increase also show a lower test accuracy; this is true both when comparing MTL-NN_{ENS} to NN_{ENS} and when comparing the same models with lexical filtering, MTL-NN_{ENS+F} and NN_{ENS+F}.

However, the previous analysis only showed the percentage change in error compared to the single-task setup, but not the resulting accuracy or how they compare to the other normalization systems. Here, we can see that only on two of the datasets, Swedish and German/RIDGES, the MTL_{SPLIT} model actually outperforms cSMTiser. Even though Icelandic and Slovene/Bohorič get slightly higher percentagewise improvements from MTL than German/RIDGES, their resulting accuracies are still lower than those of CSMT/$CSMT_{+LM}$.

The two corpora on which the MTL encoder–decoder models perform better than CSMT, Swedish and German/RIDGES, are also those with the smallest training sets (of 24,458 and 41,857 tokens, respectively). This suggests that for datasets below ca. 50,000 tokens, the MTL training approach with an auxiliary dataset can be beneficial over training cSMTiser on the single dataset. At the same time, the improvements over $CSMT_{+LM}$ are comparatively small (+0.17 pp and +0.21 pp, resp.).

German/Anselm is, again, an outlier in that the single-task NN_{ENS}/NN_{ENS+F} models achieve a higher word accuracy than the CSMT models. Conceivably, this could be related to the nature of the Anselm corpus, which consists of many different versions of the same text. A quantitative measurement that shows this exceptional property is the type/token ratio (TTR) and historical/normalized type ratio (HNR) in Tab. 3.2; the high HNR illustrates that the number of historical variants per normalized type is exceptionally high in the Anselm corpus. Possibly, this situation leads to a slight advantage for the neural network, though this interpretation must be regarded with caution as Anselm is the only dataset in this evaluation that exhibits this behavior.

Table 10.3 shows the estimated probabilities for pairwise comparisons of the methods using the Bayesian signed-rank test (cf. Sec. 10.1). Table 10.3a first compares the different variants of each method: $Norma_{ALL}$ is clearly shown to be the best setting for the Norma tool, while CSMT and $CSMT_{+LM}$ are practically equivalent with a probability of 96%. For the neural networks with or without filtering, results are slightly less clear (probably due to the filtering having different effects depending on the dataset), but on average the approaches without filtering appear to be the safer choice.

Table 10.3b compares the best configurations for each method with each other. All methods are shown to be better than $Norma_{ALL}$ with a probability greater than 91%. In the other comparisons, the results are less clear. The signed-rank test estimates a probability of 65% that CSMT words, if CSMT does perform better than NN_{ENS}, its advantage is relatively and NN_{ENS} are equivalent within the defined ROPE of 0.5 pp; in other minor. On the other hand, it is clear from this evaluation that NN_{ENS} does not improve over the CSMT model.

All in all, these results reaffirm the status of CSMT as the current state-of-the-art method for historical text normalization.

Method A	Method B	Probabilities		
		$p(A > B)$	$p(A \approx B)$	$p(B > A)$
Norma_{M}	$\text{Norma}_{\text{R+W}}$	73.54	0.05	26.41
Norma_{M}	$\text{Norma}_{\text{ALL}}$	0.02	0.00	99.98
$\text{Norma}_{\text{R+W}}$	$\text{Norma}_{\text{ALL}}$	0.00	0.00	100.00
CSMT	$\text{CSMT}_{\text{+LM}}$	3.54	96.46	0.00
NN_{ENS}	$\text{NN}_{\text{ENS+F}}$	81.61	18.39	0.00
$\text{MTL-NN}_{\text{ENS}}$	$\text{MTL-NN}_{\text{ENS+F}}$	86.12	10.37	3.52

(a) Comparison between variants of each method

Method A	Method B	Probabilities		
		$p(A > B)$	$p(A \approx B)$	$p(B > A)$
$\text{Norma}_{\text{ALL}}$	CSMT	0.00	0.24	99.76
$\text{Norma}_{\text{ALL}}$	NN_{ENS}	0.00	4.13	95.87
$\text{Norma}_{\text{ALL}}$	$\text{MTL-NN}_{\text{ENS}}$	0.15	8.23	91.62
CSMT	NN_{ENS}	34.51	65.49	0.00
CSMT	$\text{MTL-NN}_{\text{ENS}}$	80.48	19.52	0.00
NN_{ENS}	$\text{MTL-NN}_{\text{ENS}}$	22.55	76.18	1.28

(b) Comparison between best variants of each method

Table 10.3: Method comparison on the accuracy scores from the full evaluation, expressed as probabilities (in percent) of one method being better than the other ($p(A > B)$, $p(B > A)$) or both methods being approximately equivalent ($p(A \approx B)$), as estimated by a Bayesian signed-rank test using a ROPE of 0.5 pp.

10.3 Stemming

Sec. 7.4 investigated the hypothesis that many of the "incorrect" normalizations only differ from their correct target form in terms of inflection, and tested this by applying a stemming algorithm to both the predicted normalization and the gold-standard form and comparing the results. This was shown to be a promising way to measure the quality of at least a subset of the incorrect normalizations, so I will perform a similar evaluation on the test sets as well.

Table 10.4 shows the accuracy scores when comparing word stems instead of full word forms. As before, the Icelandic and Slovene datasets are not included in this comparison as the Snowball stemmer lacks a stemming algorithm for these languages (cf. Sec. 7.4). Note that the accuracy obtained this way will always be strictly equal to or higher than the accuracy on full forms (in Tab. 10.2), as word pairs with matching full forms will necessarily have matching word stems as well.

In general, the rankings between the different normalization methods do not change much when evaluating on word stems. The same trends that were observed in Tab. 10.2 can be seen in Tab. 10.4 as well: $\text{NN}_{\text{ENS+F}}$ is the best method on German/Anselm, MTL-NN_{ENS} is best on German/RIDGES and Swedish, while the CSMT models outperform the other systems on the other datasets. A minor difference is that on Spanish, CSMT is now slightly better than $\text{CSMT}_{\text{+LM}}$, although the word accuracy in Tab. 10.2 was almost identical to begin with.

Figure 10.1 shows a graphical comparison for some of the normalization methods. It mainly reinforces the findings from above: even for a dataset like German/RIDGES, where the accuracy scores of three normalizers are in close range of each other and the gain from stemming is relatively high, the ranking of the methods is the same in both comparisons. An exception to this can be found on the Spanish dataset, where MTL-NN_{ENS} is slightly better than $\text{Norma}_{\text{ALL}}$ in terms of word accuracy, but slightly worse in terms of word stem accuracy—although these differences are below 0.1 pp.

The percentage of incorrect normalizations with matching word stems is roughly comparable to the results for the development sets in Tab. 7.5. The highest percentages are again observed for Spanish, where they are usually around 40–45%, followed by German/RIDGES with 29–34%. Note that while the absolute gains from word stem matching—i.e., the light part of the bars in Fig. 10.1—are higher for German/RIDGES than Spanish, the overall word accuracy is much lower, so these instances account for a lower percentage of all incorrect normalizations.

The Bayesian signed-rank test yields very comparable results on the word accuracy and stem accuracy. There are some differences to the scores reported in Tab. 10.3: in particular, CSMT and NN_{ENS} are now estimated to be equivalent with a probability above 96%. However, this result appears for both word and stem accuracy, meaning that it is caused by the exclusion of the Icelandic and Slovene datasets rather than the effect of stemming. Since this is not a fair comparison with the results from the previous section, I do not explicitly report the scores of this test here.

All in all, the effect of word stem matching is mostly dependent on the dataset, but does not differ much between normalization methods. Sec. 7.4 analysed possible causes for the observed differences. Therefore, while the stemming approach does provide some insight about the type

Method	Dataset						
	DE_A	DE_R	EN	ES	HU	PT	SV
$Norma_M$	85.22	85.14	93.08	94.59	75.30	92.05	85.00
$Norma_{R+W}$	80.43	88.35	91.49	92.74	84.83	87.78	85.81
$Norma_{ALL}$	90.06	90.45	95.13	96.84	89.46	94.84	89.79
CSMT	90.82	92.20	95.60	**97.20**	93.34	95.39	92.70
$CSMT_{+LM}$	88.32	92.19	**95.65**	97.16	**93.40**	**95.50**	92.67
NN_{ENS}	91.03	92.08	95.22	96.86	92.79	94.96	91.89
NN_{ENS+F}	**91.67**	90.90	95.60	96.65	91.60	95.23	89.63
$MTL\text{-}NN_{ENS}$	89.81	**92.34**	94.75	96.75	91.17	93.83	**92.83**
$MTL\text{-}NN_{ENS+F}$	91.19	91.15	95.50	96.89	91.48	94.91	90.15

Table 10.4: Accuracy on word stems, in percent, evaluated for different normalization methods on the test sets of the historical datasets for which a stemming algorithm was available (cf. Sec. 7.4); best result for each dataset highlighted in bold.

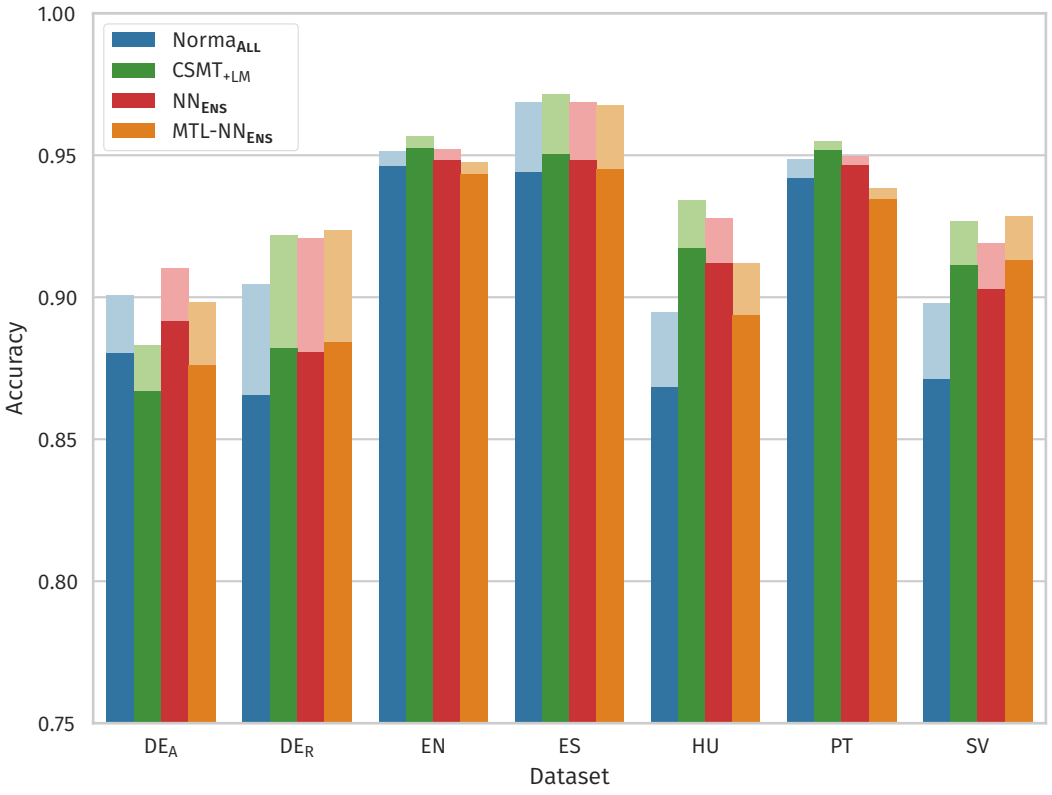


Figure 10.1: Accuracy comparison for full words vs. word stems, on selected normalization methods on the test sets; solid, dark bars show the word accuracy, light bars on top show the accuracy (gain) when evaluating on word stems. (Note that the vertical axis starts at 75%.)

of errors on the predictions, it does not change the overall recommendations for or against a particular normalizer.

10.4 Known vs. unknown tokens

Sec. 7.5 looked at the performance of the normalizers on different subsets of the data, based on phenomena that were or were not observed in the training data. Here, I will only consider the simplest of these criteria, namely whether the historical source token has been seen in the training data ("known" token) or not ("unknown" token).

One motivation for repeating this particular evaluation on the test sets is that the "unknown" tokens were not strictly unknown for all of the normalization systems; particularly, the CSMT models use the development sets as tuning data, so it is conceivable that they have an advantage over the other systems when evaluating on the same data. Incidentally, the CSMT models were also the most successful ones on the "unknown" sets in Tab. 7.7.

Table 10.5 shows the results of repeating this evaluation on the test sets. Overall, they yield a very similar outcome to the previous evaluation in Sec. 7.5: on the subset of known tokens, Norma_{M} generally performs best, while on the subset of unknown tokens, CSMT and $\text{CSMT}_{\text{+LM}}$ obtain the best results.[3] The main difference to the analysis on the development sets is that this evaluation now includes MTL models, which perform mostly better than the single-task models on German/RIDGES and Swedish, both on known and unknown tokens.[4]

Table 10.6 reports the probability scores of a Bayesian signed-rank test on some of the method pairs. For the known tokens, the methods are mostly equivalent; $\text{Norma}_{\text{ALL}}$, CSMT, and NN_{ENS} all learn to normalize them well, with MTL-NN_{ENS} being slightly behind the single-task model in this regard. For the unknown tokens, all methods are clearly better than Norma, while CSMT also outperforms the neural networks with a probability greater than 96%.

The overall conclusion here remains the same: simple wordlist mapping is very effective on the subset of known tokens (although not more effective than just using CSMT), while the strong performance of the CSMT models is mostly due to their advantage on the unknown tokens, corresponding to a better ability to generalize to previously unseen historical word forms.

10.5 Low-resource scenario

Chapter 9 described a low-resource training scenario where the training set for each dataset was cut down to a size of 5,000 tokens. In this section, I evaluate these models on the test sets. As in the previous sections, this serves the purpose of (i) confirming the findings from the previous evaluation on the development sets, and (ii) comparing the performance of selected MTL models to that of the other models. Since the low-resource training sets might conceivably profit from different dataset pairings for multi-task learning as the full training sets, the chosen pairings

[3] Considering the subset of tokens that are "unknown" in *both* training and development sets—since CSMT also makes use of the development set—does not change the overall picture.

[4] The CSMT models are slightly better on known tokens of the Swedish dataset, but the difference is minimal.

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
Known Tokens										
Tokens	*41,576*	*8,091*	*16,326*	*11,458*	*12,790*	*5,367*	*24,965*	*4,807*	*18,446*	*22,207*
$Norma_{M/All}$	**92.36**	93.66	**97.46**	**96.59**	**96.81**	**89.51**	**97.04**	**97.15**	**98.17**	97.61
$Norma_{R+W}$	80.52	89.91	93.19	91.09	89.43	86.21	89.14	93.74	96.11	92.12
CSMT	92.18	93.25	97.10	96.33	96.33	89.27	96.80	96.73	98.09	**97.66**
$CSMT_{+LM}$	90.52	93.45	97.15	96.42	96.33	88.99	96.82	96.90	98.07	**97.66**
NN_{Ens}	91.91	93.29	97.19	96.37	96.18	88.67	96.72	95.92	97.56	97.01
NN_{Ens+F}	91.99	93.63	97.38	96.53	96.44	89.10	96.92	96.63	97.92	97.51
$MTL\text{-}NN_{Ens}$	91.36	93.41	96.91	96.13	95.47	88.99	95.99	96.73	97.30	97.61
$MTL\text{-}NN_{Ens+F}$	91.80	**93.73**	97.30	96.43	96.13	89.19	96.65	97.03	97.84	97.64
Unknown Tokens										
Tokens	*4,423*	*1,496*	*1,318*	*1,021*	*3,989*	*670*	*2,113*	*1,162*	*3,047*	*6,977*
$Norma_M$	3.91	19.92	30.42	46.72	3.28	29.40	28.25	18.07	68.07	39.85
$Norma_{R+W/All}$	47.25	48.13	59.18	69.93	54.83	65.52	60.62	57.57	50.71	53.73
CSMT	57.25	59.96	**71.78**	**80.22**	76.59	**69.70**	74.96	**78.49**	83.30	70.35
$CSMT_{+LM}$	50.69	59.76	71.70	79.24	**76.86**	69.55	**75.91**	78.40	**83.56**	70.26
NN_{Ens}	63.24	59.83	65.17	77.57	75.13	68.66	70.00	73.75	80.87	68.78
NN_{Ens+F}	**69.03**	51.87	66.84	70.42	63.88	67.01	66.64	62.31	47.69	53.65
$MTL\text{-}NN_{Ens}$	52.43	**61.30**	62.06	76.20	69.77	68.96	63.18	76.33	79.49	**71.13**
$MTL\text{-}NN_{Ens+F}$	62.76	53.28	66.54	70.62	62.67	66.57	64.55	62.65	47.13	54.15

Table 10.5: Word accuracy on the test sets, evaluated separately on known and unknown tokens (= tokens where the historical word form has been seen/not seen in the training data); best result per category and dataset highlighted in bold.

Method A	Method B	Probabilities on Knowns			Probabilities on Unknowns		
		$p(A > B)$	$p(A \approx B)$	$p(B > A)$	$p(A > B)$	$p(A \approx B)$	$p(B > A)$
$Norma_{All}$	CSMT	0.00	100.00	0.00	0.00	0.00	100.00
$Norma_{All}$	NN_{Ens}	42.08	57.92	0.00	0.00	0.00	100.00
$Norma_{All}$	$MTL\text{-}NN_{Ens}$	76.60	23.40	0.00	0.00	0.00	100.00
CSMT	NN_{Ens}	1.75	98.25	0.00	96.57	0.02	3.41
CSMT	$MTL\text{-}NN_{Ens}$	13.01	86.99	0.00	99.86	0.03	0.11
NN_{Ens}	$MTL\text{-}NN_{Ens}$	0.31	99.65	0.04	92.79	0.09	7.13

Table 10.6: Method comparison on the accuracy scores for known and unknown tokens, expressed as probabilities (in percent) estimated by a Bayesian signed-rank test (cf. Sec. 10.1).

Method	Dataset									
	DE_A	DE_R	EN	ES	HU	IS	PT	SL_B	SL_G	SV
$Norma_M$	54.91	64.24	80.37	82.39	32.28	64.85	78.17	65.87	88.38	75.28
$Norma_{R+W}$	65.79	75.87	86.27	86.52	58.27	74.29	81.73	81.76	85.08	76.92
$Norma_{ALL}$	69.78	76.58	**86.83**	88.26	59.13	76.03	85.29	82.95	85.43	79.78
CSMT	70.62	78.66	85.20	88.05	53.97	77.09	83.28	87.45	**91.65**	84.60
$CSMT_{+LM}$	70.49	**78.71**	85.04	88.17	54.00	77.04	83.09	**87.57**	91.63	**84.64**
NN_{ENS}	69.59	72.90	80.79	84.40	52.83	71.49	79.82	84.72	90.30	82.01
NN_{ENS+F}	74.91	77.82	86.24	88.34	65.84	77.12	84.42	86.33	87.95	83.41
MTL-NN_{ENS}	72.63	74.23	81.53	86.70	55.36	72.88	82.14	86.46	91.14	82.39
MTL-NN_{ENS+F}	**77.19**	77.82	86.33	**89.98**	**66.75**	**78.33**	**85.92**	87.17	88.58	83.01

Table 10.7: Word accuracy of different normalization methods in the low-resource scenario, in percent, evaluated on the test sets of the historical datasets; best result for each dataset highlighted in bold.

for the MTL-NN_{ENS} and MTL-NN_{ENS+F} models are now based on the results in Fig. 9.2b (top); cf. Tab. 10.1 for an overview.

Table 10.7 presents the word accuracy for models in the low-resource scenario. The general trends are comparable to those from the development set evaluation (cf. Sec. 9.2), although the neural network models—particularly in the MTL setup—outperform CSMT on more datasets than before, e.g., on Icelandic and Portuguese. Also, the MTL models almost always perform better than their single-task counterparts, confirming the results of Fig. 9.2.

However, for the datasets where the encoder–decoder models show the highest accuracy, the lexical filtering is much more important than the multi-task setting is. For example, adding lexical filtering to the NN_{ENS} model for Hungarian results in an error reduction of 27.6%, while additionally using MTL—i.e., going from NN_{ENS+F} to MTL-NN_{ENS+F}—only gives an error reduction of 2.7%. Similar observations can be made on almost all other datasets, with German/RIDGES and Swedish even showing no improvement from MTL with filtering. Furthermore, whenever MTL-NN_{ENS+F} performs better than the CSMT models, the corresponding single-task NN_{ENS+F} model already did so as well; the NN_{ENS} model without filtering, on the other hand, is *always* worse than CSMT.

The results of the Bayesian signed-rank test in Table 10.8 basically confirm these observations. The neural network models with lexical filtering are shown to be clearly superior to those without filtering in Table 10.8a, while the other results are essentially the same as in the full evaluation (cf. Tab. 10.3a). Consequently, I use the models *with* filtering this time for the comparison between methods in Table 10.8b. Here, the NN_{ENS+F} model is now estimated to be better than CSMT with a probability of 63%, while the probabilities for the multi-task learning model over the other methods are even higher (≥ 87%).

These results suggest that both lexical filtering and multi-task learning can greatly improve the performance when training on small datasets, although the filtering step is the more significant aspect of the two. Looking at the individual results, though, CSMT still outperforms

Method A	Method B	Probabilities		
		$p(A > B)$	$p(A \approx B)$	$p(B > A)$
Norma_{M}	$\text{Norma}_{\text{R+W}}$	0.03	0.00	99.97
Norma_{M}	$\text{Norma}_{\text{ALL}}$	0.00	0.00	100.00
$\text{Norma}_{\text{R+W}}$	$\text{Norma}_{\text{ALL}}$	0.00	0.09	99.91
CSMT	$\text{CSMT}_{\text{+LM}}$	0.00	100.00	0.00
NN_{ENS}	$\text{NN}_{\text{ENS+F}}$	0.01	0.00	99.99
$\text{MTL-NN}_{\text{ENS}}$	$\text{MTL-NN}_{\text{ENS+F}}$	0.05	0.01	99.94

(a) Comparison between variants of each method

Method A	Method B	Probabilities		
		$p(A > B)$	$p(A \approx B)$	$p(B > A)$
$\text{Norma}_{\text{ALL}}$	CSMT	14.25	1.33	84.42
$\text{Norma}_{\text{ALL}}$	$\text{NN}_{\text{ENS+F}}$	0.14	0.64	99.22
$\text{Norma}_{\text{ALL}}$	$\text{MTL-NN}_{\text{ENS+F}}$	0.00	0.03	99.97
CSMT	$\text{NN}_{\text{ENS+F}}$	25.93	11.05	63.01
CSMT	$\text{MTL-NN}_{\text{ENS+F}}$	9.37	1.22	89.41
$\text{NN}_{\text{ENS+F}}$	$\text{MTL-NN}_{\text{ENS+F}}$	0.00	12.91	87.09

(b) Comparison between best variants of each method

Table 10.8: Method comparison on the accuracy scores from the low-resource scenario, expressed as probabilities (in percent) of one method being better than the other ($p(A > B)$, $p(B > A)$) or both methods being approximately equivalent ($p(A \approx B)$), as estimated by a Bayesian signed-rank test (cf. Sec. 10.1).

$\text{MTL-NN}_{\text{ENS+F}}$ on some datasets (e.g., 84.6% vs. 83.0% on Swedish). One possible factor might be the lexical coverage as provided in Tab. 3.7: on all datasets where CSMT outperforms $\text{NN}_{\text{ENS+F}}$/ $\text{MTL-NN}_{\text{ENS+F}}$, the coverage of the lexicon is comparatively low.[5] As a consequence, the lexical filtering is less effective since many of the correct target normalizations cannot be reached. Therefore, lexical coverage could possibly be used as a criterion to decide between these two models.

[5]Hungarian is again an exception here, since the coverage is low there as well, but $\text{NN}_{\text{ENS+F}}$ shows a huge improvement over CSMT.

Chapter 11

Conclusion

> *Consistency is not always a virtue; but spelling becomes a will-o'-the-wisp without it.*
> — George Bernard Shaw, from his Preface to R. A. Wilson's *The Miraculous Birth of Language* (1941)

In this thesis, I presented and analyzed an encoder–decoder neural network model for automatic normalization of historical texts. I evaluated the model on a diverse collection of historical datasets from English, German, Hungarian, Icelandic, Portuguese, Slovene, Spanish, and Swedish, and compared it with the previously established normalization tools Norma (implementing wordlist mapping, a rule-based approach, and a distance-based approach) and cSMTiser (using character-based statistical machine translation, CSMT). The general trend observed in the evaluations is that cSMTiser is better than the other methods on most of the datasets, and should still be considered the state of the art for historical text normalization.

Starting from a basic encoder–decoder network with LSTM layers inspired by Sutskever et al. (2014), I tuned several hyperparameters of the model and the training algorithm and gradually built upon it by adding an attention mechanism, using beam search decoding and lexical filtering, and using ensembles of five independently trained models. Attention and beam search decoding often resulted in improvements to normalization accuracy; when they did not help, they did not significantly degrade accuracy either. Lexical filtering was more ambiguous: depending on the dataset, its effect was sometimes beneficial, sometimes detrimental. Model ensembling provided a consistent and significant increase in accuracy.

Furthermore, I experimented with different training scenarios: I used multi-task learning (MTL) to train an encoder–decoder model on two datasets in parallel; and I retrained all models on low-resource training sets of only 5,000 tokens each. I also combined these two scenarios by using multi-task learning to improve the accuracy for a low-resource training set by pairing it with another, full dataset. The general observation is that a dataset profits more from multi-task learning the smaller its training set is, and that multi-task learning can even reduce the normalization accuracy with large training sets. Pairings of closely related languages sometimes showed notably better performance, although I could not find any correlation with other similarity measures between datasets.

Despite all the individual improvements to the encoder–decoder model, the comparative evaluation revealed that it is still outperformed by the CSMT system in most scenarios. This is true for both the full evaluation and the low-resource training; only when multi-task learning

is used in combination with lexical filtering in the low-resource scenario, the encoder–decoder model performs better than CSMT on a majority of datasets. These results indicate that the MTL+filtering approach might be practical in certain situations where little training data is available. It is, of course, not a fair comparison of the underlying machine learning algorithms, as the neural network is given much more data to work with in this case. This again emphasizes the strong performance of CSMT in this evaluation.

I hope to have made a convincing case here that the lower performance of the encoder–decoder model does not stem from a lack of optimization nor from a lack of effort to improve the architecture. While this is mostly a negative result, I believe it is still an important contribution, not least because it is in contrast to the general trend of neural networks achieving state-of-the-art performance in many NLP tasks (Goldberg, 2017). Furthermore, neural network models similar to the one analyzed here have been shown to perform very well in many related tasks such as machine translation (Sutskever et al., 2014; Ling et al., 2015; Chung, Cho, et al., 2016), grapheme-to-phoneme conversion (Rao et al., 2015), language correction (Xie et al., 2016), and—of course—historical normalization (Bollmann et al., 2017; Korchagina, 2017).

As an exception to this, Schnober et al. (2016) analyze encoder–decoder models for monotone string translation tasks such as spelling correction, lemmatization, or grapheme-to-phoneme conversion, and in many cases find them to be inferior to traditional approaches, e.g., pruned conditional random fields (PCRFs). While I have not experimented with PCRFs here, historical normalization can certainly be interpreted as a monotone string translation task. In this sense, the results reported in this thesis are supporting these findings by Schnober et al. (2016).

11.1 Evaluating automatic normalization

Substantial parts of this thesis were devoted to the issue of evaluation. The discussion of normalization guidelines has shown that different corpora can employ widely differing guidelines when it comes to the preparation of manually curated, or ”gold-standard”, normalization annotation. They may differ in how they handle inflectional changes, whether they normalize proper nouns, or whether and how they modify extinct words. Many of these decisions can influence the performance of an automatic normalization system trained on this data. However, how best to take them into account when comparing results on different datasets is a challenging question; the qualitative analysis has not referred back to these individual differences as much as I would have liked, although the different treatment of inflection in the two German datasets did produce a noticeable effect in the word stem evaluation (see below).

Regarding evaluation measures for automatic normalization, the most suitable measure to me still appears to be word accuracy. While character error rate is also often employed, I believe it is not very informative over word accuracy as (i) it often correlates with it, and (ii) it does not address the issue that not every character mismatch is alike. For example, take the Spanish gold-standard normalization *está*: the (hypothetical) normalization candidates *esta* and *estx* are identical in terms of character error rate, though it is doubtful that the latter is more useful or ”correct” than, e.g., a much longer non-word string, while the former can actually be a good prediction. One potential way to address this is to perform automatic word stemming on both the predicted and the gold-standard target forms; of all the different qualitative analyses

investigated here, I believe this is the most useful as it (i) provides useful additional insight into the data and (at least a subset of) the normalization errors, and (ii) is simple to implement for many languages where an automatic stemming algorithm is already available.

Manual error analysis can also be a very valuable, though time-consuming option. The small error classification I performed on the German and English datasets, however, revealed that the predicted normalization candidates are reasonable and useful in many more cases than the word accuracy measure suggests. Such an error analysis can suggest ways to do more nuanced automatic evaluation—e.g., by considering word stems—or inform the direction of future work to improve normalization by highlighting specific problem areas. At the very least, it can help to better understand and assess the output of the automatic normalizer.

11.2 Improving the neural network model

Do the results from this thesis imply that neural networks are inferior to classical methods when it comes to historical normalization? Not at all. We can conclude that this particular neural network architecture, using the layers and decoding techniques presented here, is most likely inferior to classical CSMT. However, most of the research on neural networks for NLP is relatively new, and countless variants and additional techniques have been proposed. It is entirely conceivable that one of those might prove to be the key for making neural networks perform better than CSMT on historical normalization.

At least one concrete area for improvement has been suggested by the qualitative evaluation, in the case study from Portuguese in Sec. 7.5.3: having a stronger, bi-directional language modeling component in the network's decoder. In the architecture presented here, the decoder part of the model only conditions its output on previously generated characters; i.e., it operates strictly from left to right. The observed normalization errors from Portuguese suggested that this might partly account for its weaker performance compared to CSMT. Liu, Finch, et al. (2016) note that this is a fundamental shortcoming of RNNs, causing them to predict strong prefixes but weaker suffixes, and suggest *target-bidirectional decoding* as a solution. Their approach combines a forward and backward RNN using approximate search techniques and could potentially be useful for the normalization task as well.

In this thesis, I focused exclusively on LSTM layers (Hochreiter and Schmidhuber, 1997) as the components in my encoder–decoder architecture. However, several other options exist. Gated recurrent units (GRUs; Cho, Merrienboer, Gülçehre, et al., 2014) are a structurally simpler alternative to LSTMs that have also been successfully used for many tasks, including neural machine translation (e.g., Chung, Cho, et al., 2016).[1] A structurally very different alternative are *convolutional layers,* which apply the same set of transformations over a sliding window across a sequence, essentially functioning as "n-gram detectors" for NLP (cf. Goldberg, 2017, Chapter 13). Bradbury et al. (2016) propose the "Quasi-RNN", a convolutional layer intended as a replacement for RNNs in sequence modeling tasks. Convolutional architectures have also

[1] I briefly experimented with GRU layers, but found no noticeable advantage over LSTMs and did not explore them further.

successfully been applied to morphological reinflection (Östling, 2016) and machine translation, both on a word level (Gehring et al., 2017) and on a character level (Lee et al., 2017).[2]

The attention mechanism is another aspect that could be varied. Luong, Pham, et al. (2015) compare a variety of approaches to attention-based neural machine translation, including *local* attention mechanisms that only focus on a small part of the input sequence at a time. Attention is often used with the idea that it allows the model to learn some form of alignment between the input and output sequences. Liu, Utiyama, et al. (2016) experiment with supervised attention for machine translation, i.e., explicitly training the model on what to attend at each timestep. This could be applied to normalization by using character alignments (cf. Sec. 3.3) as the basis for the attention model. These alignments are also *monotonic,* which can be explicitly modeled in the attention component: Tjandra et al. (2017) describe a local monotonic attention mechanism for speech processing, essentially making the model predict how far to advance the "center of attention" at each timestep, while Aharoni and Goldberg (2017) propose hard monotonic attention for morphological inflection generation.[3]

Finally, adjustments could be made to the training and/or decoding process. In the input-feeding approach used here (cf. p. 87), the decoder receives the previously predicted character as input when generating the next one. This creates a discrepancy between training and testing: during training, the decoder always sees the correct target prediction of the previous character, while during testing, these predictions may sometimes be incorrect. This can cause errors to accumulate along the remainder of the word form, as the decoder can now be in a state that was never seen during training. Bengio et al. (2015) propose *scheduled sampling*—alternating between feeding the correct and the actually predicted output during training—to make the decoder more robust to this scenario. Wiseman and Rush (2016) introduce a training scheme that specifically optimizes a model for beam-search decoding. A further discrepancy is that the model is trained by *locally* optimizing the correct prediction of each character, while the main evaluation measure—word accuracy—considers the *global* accuracy of all predicted characters in a word form. Shen et al. (2016) propose *minimum risk training* to allow a model to be trained directly with a sequence-level loss function such as word accuracy.

In short, many changes and additions to the encoder–decoder architecture for sequence-to-sequence learning have been and still are being suggested, particularly in the context of neural machine translation. All of these could conceivably improve historical normalization using these architectures as well.

11.3 Beyond token-level normalization

A common limitation of all normalization systems evaluated here is that they only operate on a token level, i.e., they normalize single word forms in isolation. This puts an upper limit on

[2] I also briefly experimented with simple convolutional layers in the encoder, though without much success. However, many variants of convolutional architectures exist, and a more thorough investigation is certainly warranted before drawing any conclusions.

[3] I performed preliminary experiments with some of these local and monotonic attention variants, but could not find any improvements over the global attention mechanism. A recent study by Robertson and Goldwater (2018) compared the two approaches for historical normalization, also with mixed results.

the achievable normalization accuracy, as some historical tokens will usually be ambiguous with regard to their correct normalization, depending on context. The analysis in Sec. 3.4 investigated the extent of this issue.

However, few normalization methods have been proposed so far that take word context into account. Jurish (2010b) uses hidden Markov models to select between different normalization candidates depending on word context. Ljubešić et al. (2016b) experiment with "segment-level" normalization, essentially using a string of several historical tokens without explicit word segmentation as the input to the normalizer. This is similar to the fully character-level approach to machine translation by Lee et al. (2017), except that historical text often comes with the additional challenge of how to define the "segments", since sentence boundaries are not always available or easily detectable.

Another option to integrate context is to use a hierarchical architecture that contains both character-level and word-level representations. Hierarchical RNN architectures have been proposed for language modeling (Chung, Ahn, et al., 2016) and document modeling (Lin et al., 2015); in a similar vein, Luong and Manning (2016) propose a hybrid word–character model for neural machine translation to improve the performance on rare words. Essentially, the idea is to treat the output of a character-level encoder (such as the one I used in my experiments) as a representation of the full historical word form, and feed that into another, word-level RNN layer that can transform this representation based on word context. Decoding then only starts after this additional word-level layer.[4]

A simpler option is to integrate context information into the current token-level model as an additional feature. This could be achieved by "encoding" the word context, e.g., by using a separate character-level LSTM, and feeding the resulting context vector into the token-level encoder–decoder network.[5] Horn (2017) suggests to multiply a word embedding vector with the average of its context vectors to obtain a context-dependent representation; this approach could be transferred to the encoder–decoder model presented here by treating the encoder's output as the "word embedding" of the historical input token.

Furthermore, we can also consider NLP tasks for historical texts that go beyond normalization, but still might profit from it. A commonly desired task is POS tagging; many previous studies have shown that normalization can improve the performance of a POS tagger on historical data (e.g., Scheible et al., 2011b; Bollmann, 2013b; Sang, 2016; Yang and Eisenstein, 2016; Ljubešić et al., 2017). Combining these tasks is typically done as a pipeline, where texts are first normalized before POS tagging is performed on the normalized data. However, it is conceivable that both tasks could profit from a closer interaction: automatic normalization could benefit from the knowledge of POS tags—e.g., by ruling out normalization candidates that do not fit the most likely part-of-speech category—and POS tagging could benefit from the knowledge of historical word forms—e.g., when two lexemes are distinguished in historical but not in contemporary spelling. Multi-task learning architectures (e.g., Luong, Le, et al., 2015; Søgaard and Goldberg, 2016; Yang et al., 2016) could conceivably be used to train models on

[4] I performed some experiments with hierarchical structures like these, but could not get the model to learn anything reasonable at all, i.e., accuracy stayed close to 0%. Note that the complexity of the model increases considerably with such an architecture; possibly, the amount of training data I used in my experiments is not enough to train such a network.

[5] I experimented with this approach and found it to work well, but not better than the model without the context.

normalization and POS tagging simultaneously in order to improve the performance on one or both tasks.

In conclusion, there is a lot of potential for future research on natural language processing for historical texts, both within the area of normalization and beyond it. The ongoing digitization efforts, making more and more historical documents available in machine-readable form, will only increase the demand for efficient NLP tools for this data in the future.

Bibliography

Adesam, Yvonne, Ahlberg, Malin, and Bouma, Gerlof (2012). "*bokstaffua, bokstaffwa, bokstafwa, bokstaua, bokstawa...* Towards lexical link-up for a corpus of Old Swedish". In: *Proceedings of the 11th Conference on Natural Language Processing (KONVENS 2012), LThist 2012 workshop*. Vienna, Austria, pp. 365–369. URL: http://www.oegai.at/konvens2012/proceedings/54_adesam12w/ (cited on p. 68).

Aharoni, Roee and Goldberg, Yoav (2017). "Morphological Inflection Generation with Hard Monotonic Attention". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 2004–2015. DOI: 10.18653/v1/P17-1183. URL: http://www.aclweb.org/anthology/P17-1183 (cited on p. 182).

Al Azawi, Mayce, Afzal, Muhammad Zeshan, and Breuel, Thomas M. (2013). "Normalizing Historical Orthography for OCR Historical Documents using LSTM". In: *Proceedings of the 2nd International Workshop on Historical Document Imaging and Processing (HIP '13)*. Washington, DC, pp. 80–85. DOI: 10.1145/2501115.2501131 (cited on p. 70).

Amoia, Marilisa and Martínez, José Manuel (2013). "Using Comparable Collections of Historical Texts for Building a Diachronic Dictionary for Spelling Normalization". In: *Proceedings of the 7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 84–89. URL: http://www.aclweb.org/anthology/W13-2711 (cited on pp. 18, 68).

Archer, Dawn, Kytö, Merja, Baron, Alistair, and Rayson, Paul (2015). "Guidelines for normalising Early Modern English corpora: Decisions and justifications". *ICAME Journal*, 39, pp. 5–24. DOI: 10.1515/icame-2015-0001 (cited on pp. 16, 18, 20, 22, 23).

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". *CoRR*, abs/1409.0473. URL: http://arxiv.org/abs/1409.0473 (cited on pp. xx, 8, 87, 88, 93, 94).

Baldwin, Tyler and Li, Yunyao (2015). "An In-depth Analysis of the Effect of Text Normalization in Social Media". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, pp. 420–429. DOI: 10.3115/v1/N15-1045 (cited on p. 64).

Barnbrook, Geoff (1996). *Language and Computers. A Practical Introduction to the Computer Analysis of Language*. Edinburgh: Edinburgh University Press (cited on p. 67).

Baron, Alistair and Rayson, Paul (2008). "VARD 2: A tool for dealing with spelling variation in historical corpora". In: *Proceedings of the Postgraduate Conference in Corpus Linguistics* (cited on pp. 45, 65, 67).

Baron, Alistair and Rayson, Paul (2009). "Automatic standardization of texts containing spelling variation. How much training data do you need?" In: *Proceedings of the Corpus Linguistics Conference (CL 2009)* (cited on p. 18).

Baron, Alistair, Rayson, Paul, and Archer, Dawn (2009). "Word frequency and key word statistics in corpus linguistics". *Anglistik*, 20(1), pp. 41–67 (cited on p. 14).

Barteld, Fabian, Schröder, Ingrid, and Zinsmeister, Heike (2015). "Unsupervised regularization of historical texts for POS tagging". In: *Proceedings of the Workshop on Corpus-Based Research in the Humanities (CRH)*. Ed. by Francesco Mambrini, Marco Passarotti, and Caroline Sporleder. Warsaw, Poland, pp. 3–12 (cited on pp. 17, 68).

Belz, Malte, Odebrecht, Carolin, Perlitz, Laura, Schnelle, Gohar, and Voigt, Vivian (2017). *Annotationsrichtlinien zu Ridges Herbology Version 6.0*. Tech. rep. Humboldt-Universität zu Berlin. URL: https://www.linguistik.hu-berlin.de/de/institut/professuren/korpuslinguistik/forschung/ridges-projekt/releases-de (cited on pp. 21, 22, 24, 33).

Benavoli, Alessio, Corani, Giorgio, Demšar, Janez, and Zaffalon, Marco (2017). "Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis". *Journal of Machine Learning Research*, 18(77), pp. 1–36. URL: http://jmlr.org/papers/v18/16-305.html (cited on pp. 168, 169).

Benavoli, Alessio, Corani, Giorgio, Mangili, Francesca, Zaffalon, Marco, and Ruggeri, Fabrizio (2014). "A Bayesian Wilcoxon signed-rank test based on the Dirichlet process". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, pp. 1026–1034. URL: http://proceedings.mlr.press/v32/benavoli14.html (cited on p. 169).

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam (2015). "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., pp. 1171–1179. URL: http://papers.nips.cc/paper/5956-scheduled-sampling-for-sequence-prediction-with-recurrent-neural-networks.pdf (cited on pp. 87, 182).

Bengio, Yoshua (2012). "Practical Recommendations for Gradient-Based Training of Deep Architectures". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 437–478. DOI: 10.1007/978-3-642-35289-8_26 (cited on p. 83).

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo (1994). "Learning Long-Term Dependencies with Gradient Descent is Difficult". *IEEE Transactions on Neural Networks*, 5(2), pp. 157–166 (cited on p. 78).

Berg-Kirkpatrick, Taylor and Klein, Dan (2014). "Improved Typesetting Models for Historical OCR". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 118–123. DOI: 10.3115/v1/P14-2020 (cited on p. 15).

Bergstra, James S., Bardenet, Rémi, Bengio, Yoshua, and Kégl, Balázs (2011). "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger. Curran

Associates, Inc., pp. 2546–2554. URL: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf (cited on p. 92).

Bergstra, James, Yamins, Daniel, and Cox, David (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, pp. 115–123. URL: http://proceedings.mlr.press/v28/bergstra13.html (cited on p. 92).

Berry, David M., ed. (2012). *Understanding Digital Humanities*. Basingstoke, UK: Palgrave Macmillan. DOI: 10.1057/9780230371934 (cited on p. 1).

Berry, David M. and Fagerjord, Anders (2017). *Digital Humanities. Knowledge and Critique in a Digital Age*. Cambridge, UK: Polity Press (cited on p. 1).

Biber, Douglas, Finegan, Edward, and Atkinson, Dwight (1994). "ARCHER and its challenges: Compiling and exploring A Representative Corpus of Historical English Registers". In: *Creating and using English language corpora. Papers from the 14th International Conference on English Language Research on Computerized Corpora*. Ed. by Udo Fries, Peter Schneider, and Gunnel Tottie. Amsterdam: Rodopi, pp. 1–13 (cited on p. 1).

Bjarnadóttir, Kristín (2012). "The Database of Modern Icelandic Inflection (Beygingarlýsing íslensks nútímamáls)". In: *Proceedings of the Workshop on Language Technology for Normalisation of Less-Resourced Languages*. Istanbul, Turkey, pp. 13–18 (cited on p. 57).

Bollmann, Marcel (2012). "(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)*. Lisbon, Portugal, pp. 3–12 (cited on pp. xvi, xvii, 9, 45, 65, 68, 71, 137).

– (2013a). "Automatic Normalization for Linguistic Annotation of Historical Language Data". *Bochumer Linguistische Arbeitsberichte*, 13. URL: http://www.linguistics.rub.de/bla/013-bollmann2013.pdf (cited on pp. 40, 71, 140, 160).

– (2013b). "POS Tagging for Historical Texts with Sparse Training Data". In: *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 11–18. URL: http://www.aclweb.org/anthology/W13-2302 (cited on p. 183).

Bollmann, Marcel, Bingel, Joachim, and Søgaard, Anders (2017). "Learning attention for historical text normalization by learning to pronounce". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 332–344. DOI: 10.18653/v1/P17-1031 (cited on pp. xvii, xx, 9, 28, 29, 31, 70, 86, 90, 93, 99, 104, 145, 147, 149, 153, 180).

Bollmann, Marcel, Dipper, Stefanie, Krasselt, Julia, and Petran, Florian (2012). "Manual and semi-automatic normalization of historical spelling – Case studies from Early New High German". In: *Proceedings of the 11th Conference on Natural Language Processing (KONVENS 2012), LThist 2012 workshop*. Vienna, Austria, pp. 342–350. URL: http://www.oegai.at/konvens2012/proceedings/51_bollmann12w/ (cited on p. 71).

Bollmann, Marcel, Dipper, Stefanie, and Petran, Florian (2016). "Evaluating Inter-Annotator Agreement on Historical Spelling Normalization". In: *Proceedings of the 10th Linguistic Annotation Workshop held in conjunction with ACL 2016 (LAW-X 2016)*. Berlin, Germany: Association for Computational Linguistics, pp. 89–98. URL: http://anthology.aclweb.org/W16-1711 (cited on p. 27).

Bollmann, Marcel, Petran, Florian, and Dipper, Stefanie (2011a). "Applying rule-based normalization to different types of historical texts — an evaluation". In: *Proceedings of LTC 2011*. Ed. by Zygmunt Vetulani. Poznan, Poland, pp. 339–344 (cited on pp. 3, 29, 71).

– (2011b). "Rule-Based Normalization of Historical Texts". In: *Proceedings of the International Workshop on Language Technologies for Digital Humanities and Cultural Heritage*. Hissar, Bulgaria, pp. 34–42 (cited on pp. xvi, 7, 43, 66, 67).

Bollmann, Marcel and Søgaard, Anders (2016). "Improving historical spelling normalization with bi-directional LSTMs and multi-task learning". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 131–139. URL: http://www.aclweb.org/anthology/C16-1013 (cited on pp. 9, 29, 31, 43, 70, 145).

Bollmann, Marcel, Søgaard, Anders, and Bingel, Joachim (2018). "Multi-task learning for historical text normalization: Size matters". In: *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP*. Melbourne: Association for Computational Linguistics, pp. 19–24. URL: http://aclweb.org/anthology/W18-3403 (cited on p. 145).

Bowers, Fredson (1989). "Regularization and normalization in modern critical texts". *Studies in Bibliography*, 42, pp. 79–102. URL: http://xtf.lib.virginia.edu/xtf/view?docId=StudiesInBiblio/uvaBook/tei/sibv042.xml (cited on pp. 17, 18).

Bradbury, James, Merity, Stephen, Xiong, Caiming, and Socher, Richard (2016). "Quasi-Recurrent Neural Networks". *CoRR*, abs/1611.01576. URL: http://arxiv.org/abs/1611.01576 (cited on p. 181).

Brill, Eric and Moore, Robert C. (2000). "An Improved Error Model for Noisy Channel Spelling Correction". In: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*. URL: http://www.aclweb.org/anthology/P00-1037 (cited on p. 69).

Callison-Burch, Chris, Osborne, Miles, and Koehn, Philipp (2006). "Re-evaluating the Role of Bleu in Machine Translation Research". In: *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*. URL: http://www.aclweb.org/anthology/E06-1032 (cited on p. 118).

Carbonell, Jaime G. and Tomita, Masaru (1985). "New Approaches to Machine Translation". In: *Proceedings of the Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, pp. 59–74. URL: http://repository.cmu.edu/compsci/355/ (cited on p. 111).

Caruana, Rich (1993). "Multitask Learning: A Knowledge-Based Source of Inductive Bias". In: *Proceedings of the 10th International Conference on Machine Learning (ICML)*, pp. 41–48. URL: http://www.cs.cornell.edu/~caruana/ml93.ps (cited on pp. xviii, 145).

– (1997). "Multitask Learning". *Machine Learning*, 28, pp. 41–75. URL: http://www.cs.cornell.edu/~caruana/mlj97.pdf (cited on p. 145).

Centro de Linguística da Universidade de Lisboa (CLUL), ed. (2014). *P.S. Post Scriptum. Arquivo Digital de Escrita Quotidiana em Portugal e Espanha na Época Moderna*. URL: http://ps.clul.ul.pt (cited on p. 36).

Cho, KyungHyun, Merrienboer, Bart van, Bahdanau, Dzmitry, and Bengio, Yoshua (2014). "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, pp. 103–111. URL: http://www.aclweb.org/anthology/W14-4012 (cited on p. 87).

Cho, Kyunghyun, Merrienboer, Bart van, Gülçehre, Çaglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua (2014). "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)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179 (cited on pp. xviii, 85, 93, 181).

Chollet, François et al. (2015). *Keras*. https://github.com/fchollet/keras (cited on p. 73).

Chollet, François (2017). *Deep Learning with Python*. Shelter Island, NY: Manning. URL: https://www.manning.com/books/deep-learning-with-python (cited on pp. xviii, 9, 73).

Christodouloupoulos, Christos and Steedman, Mark (2015). "A massively parallel corpus: the Bible in 100 languages". *Language Resources and Evaluation*, 49(2), pp. 375–395. DOI: 10.1007/s10579-014-9287-y (cited on p. 57).

Chrupała, Grzegorz (2014). "Normalizing tweets with edit scripts and recurrent neural embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 680–686. DOI: 10.3115/v1/P14-2111 (cited on pp. 8, 70).

Chung, Junyoung, Ahn, Sungjin, and Bengio, Yoshua (2016). "Hierarchical Multiscale Recurrent Neural Networks". *CoRR*, abs/1609.01704. URL: http://arxiv.org/abs/1609.01704 (cited on p. 183).

Chung, Junyoung, Cho, Kyunghyun, and Bengio, Yoshua (2016). "A Character-level Decoder without Explicit Segmentation for Neural Machine Translation". *CoRR*, abs/1603.06147. URL: http://arxiv.org/abs/1603.06147 (cited on pp. 70, 180, 181).

Church, Kenneth W. and Hanks, Patrick (1990). "Word association norms, mutual information, and lexicography". *Computational Linguistics*, 16(1), pp. 22–29 (cited on p. 41).

Cohn, Trevor, Hoang, Cong Duy Vu, Vymolova, Ekaterina, Yao, Kaisheng, Dyer, Chris, and Haffari, Gholamreza (2016). "Incorporating Structural Alignment Biases into an Attentional Neural Translation Model". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 876–885. DOI: 10.18653/v1/N16-1102 (cited on p. 88).

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel (2011). "Natural Language Processing (Almost) from Scratch". *The Journal of*

Machine Learning Research, 12, pp. 2493–2537. URL: http://www.jmlr.org/papers/v12/collobert11a.html (cited on p. 145).

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals and Systems*, 2(4), pp. 303–314. DOI: 10.1007/BF02551274 (cited on p. 75).

Daumé III, Hal (2007). "Frustratingly Easy Domain Adaptation". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, pp. 256–263. URL: http://www.aclweb.org/anthology/P07-1033 (cited on p. 6).

Daumé III, Hal and Marcu, Daniel (2006). "Domain Adaptation for Statistical Classifiers". *Journal of Artificial Intelligence Research (JAIR)*, 26, pp. 101–126. URL: http://hal3.name/docs/#daume06megam (cited on p. 6).

Davies, Mark (2012). "Expanding horizons in historical linguistics with the 400-million word Corpus of Historical American English". *Corpora*, 7(2), pp. 121–157. DOI: 10.3366/cor.2012.0024 (cited on p. 1).

Davis, Mark and Whistler, Ken (2017). *Unicode Normalization Forms*. Unicode Standard Annex #15. Technical report. Version 10.0.0. The Unicode Consortium. URL: http://unicode.org/reports/tr15/ (cited on pp. 39, 40).

Dipper, Stefanie and Schultz-Balluff, Simone (2013). "The Anselm Corpus: Methods and Perspectives of a Parallel Aligned Corpus". In: *Proceedings of the Workshop on Computational Historical Linguistics at NODALIDA 2013*. NEALT Proceedings Series 18/Linköping Electronic Conference Proceedings 87. Oslo, Norway: Linköping University Electronic Press, pp. 27–42. URL: http://www.ep.liu.se/ecp/article.asp?issue=087&article=003 (cited on pp. 15, 16).

Domingo, Miguel and Casacuberta, Francisco (2018). "Spelling Normalization of Historical Documents by Using a Machine Translation Approach". In: *Proceedings of the 21st Annual Conference of the European Association for Machine Translation*. Ed. by Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Miquel Esplà-Gomis, Maja Popović, Celia Rico, André Martins, Joachim van den Bogaert, and Mikel L. Forcada. Alacant, pp. 129–137. URL: http://hdl.handle.net/10045/76035 (cited on p. 70).

Dong, Daxiang, Wu, Hua, He, Wei, Yu, Dianhai, and Wang, Haifeng (2015). "Multi-Task Learning for Multiple Language Translation". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1723–1732. DOI: 10.3115/v1/P15-1166 (cited on p. 145).

Durrett, Greg and DeNero, John (2013). "Supervised Learning of Complete Morphological Paradigms". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, pp. 1185–1195. URL: http://www.aclweb.org/anthology/N13-1138 (cited on p. 65).

Eisenstein, Jacob (2013). "What to do about bad language on the internet". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies. Atlanta, Georgia: Association for Computational Linguistics, pp. 359–369. URL: http://www.aclweb.org/anthology/N13-1037 (cited on p. 64).

Elman, Jeffrey L. (1990). "Finding Structure in Time". *Cognitive Science*, 14(2), pp. 179–211. DOI: 10.1016/0364-0213(90)90002-E (cited on p. 78).

Erjavec, Tomaž (2012). "The goo300k corpus of historical Slovene". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Ed. by Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Istanbul, Turkey: European Language Resources Association (ELRA), pp. 2257–2260. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/445_Paper.pdf (cited on pp. 2, 9, 23, 28, 35).

– (2015). "The IMP historical Slovene language resources". *Language Resources and Evaluation*, 49(3), pp. 753–775. DOI: 10.1007/s10579-015-9294-7 (cited on pp. 19, 24, 36, 55).

Ernst-Gerlach, Andrea and Fuhr, Norbert (2006). "Generating Search Term Variants for Text Collections with Historic Spellings". In: *Proceedings of the 28th European Conference on Information Retrieval Research (ECIR 2006)*. Lecture Notes in Computer Science. Berlin: Springer, pp. 49–60. DOI: 10.1007/11735106 (cited on pp. xvi, 14, 67).

Etxeberria, Izaskun, Alegria, Iñaki, Uria, Larraitz, and Hulden, Mans (2016). "Evaluating the Noisy Channel Model for the Normalization of Historical Texts: Basque, Spanish and Slovene". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Ed. by Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 1064–1069. URL: http://www.lrec-conf.org/proceedings/lrec2016/pdf/147_Paper.pdf (cited on pp. 18, 29, 35, 67, 69).

Faruqui, Manaal, Tsvetkov, Yulia, Neubig, Graham, and Dyer, Chris (2016). "Morphological Inflection Generation Using Character Sequence to Sequence Learning". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 634–643. DOI: 10.18653/v1/N16-1077 (cited on p. 65).

Fiebranz, Rosemarie, Lindberg, Erik, Lindström, Jonas, and Ågren, Maria (2011). "Making verbs count: the research project 'Gender and Work' and its methodology". *Scandinavian Economic History Review*, 59(3), pp. 273–293. DOI: 10.1080/03585522.2011.617576 (cited on pp. 2, 15, 28, 38).

Fix, Hans (1980). "Automatische Normalisierung – Vorarbeit zur Lemmatisierung eines diplomatischen altisländischen Textes". In: *Maschinelle Verarbeitung altdeutscher Texte. Beiträge zum dritten Symposion, Tübingen 17.–19. Februar 1977*. Ed. by Paul Sappler and Erich Straßner. Tübingen: Niemeyer, pp. 92–100 (cited on pp. xv, xvi, 5, 7, 17, 66).

Freund, Yoav and Schapire, Robert E. (1999). "Large Margin Classification Using the Perceptron Algorithm". *Machine Learning*, 37(3), pp. 277–296. DOI: 10.1023/A:1007662407062 (cited on pp. 74, 75).

Froger, Jacques (1970). "La critique des textes et l'ordinateur". *Vigiliae Christianae*, 24(3), pp. 210–217. DOI: 10.2307/1583073 (cited on p. 63).

Gal, Yarin and Ghahramani, Zoubin (2016). "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., pp. 1019–1027. URL: http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf (cited on pp. 83, 99).

Gehring, Jonas, Auli, Michael, Grangier, David, and Dauphin, Yann N. (2016). "A Convolutional Encoder Model for Neural Machine Translation". *CoRR*, abs/1611.02344. URL: http://arxiv.org/abs/1611.02344 (cited on p. 85).

Gehring, Jonas, Auli, Michael, Grangier, David, Yarats, Denis, and Dauphin, Yann N. (2017). "Convolutional Sequence to Sequence Learning". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 1243–1252. URL: http://proceedings.mlr.press/v70/gehring17a.html (cited on p. 182).

Giusti, Rafael, Candido Jr, Arnaldo, Muniz, Marcelo, Cucatto, Lívia, and Aluísio, Sandra (2007). "Automatic Detection of Spelling Variation in Historical Corpus: An Application to Build a Brazilian Portuguese Spelling Variants Dictionary". In: *Proceedings of the Corpus Linguistics Conference (CL2007)*. Ed. by Matthew Davies, Paul Rayson, Susan Hunston, and Pernilla Danielsson. Birmingham, UK. URL: http://ucrel.lancs.ac.uk/publications/cl2007/paper/238_Paper.pdf (cited on pp. 17, 67).

Goldberg, Yoav (2017). *Neural Network Methods for Natural Language Processing*. Synthesis Lectures on Human Language Technologies 37. Morgan & Claypool. DOI: 10.2200/S00762ED1V01Y201703HLT037 (cited on pp. xvii, xviii, 8, 70, 73, 75, 77, 79, 82, 93, 106, 147, 180, 181).

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press (cited on pp. 73, 82, 83).

Graves, Alex and Schmidhuber, Jürgen (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". *Neural Networks*, 18(5), pp. 602–610. DOI: 10.1016/j.neunet.2005.06.042 (cited on p. 80).

Halteren, Hans van and Rem, Margit (2013). "Dealing with orthographic variation in a tagger-lemmatizer for fourteenth century Dutch charters". *Language Resources and Evaluation*, 47(4), pp. 1233–1259. DOI: 10.1007/s10579-013-9236-1 (cited on p. 69).

Hämäläinen, Mika, Säily, Tanja, Rueter, Jack, Tiedemann, Jörg, and Mäkelä, Eetu (2018). "Normalizing Early English Letters to Present-day English Spelling". In: *Proceedings of the Second Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*. Santa Fe, New Mexico: Association for Computational Linguistics, pp. 87–96. URL: http://aclweb.org/anthology/W18-4510 (cited on pp. 70, 140).

Hana, Jirka, Feldman, Anna, and Aharodnik, Katsiaryna (2011). "A low-budget tagger for Old Czech". In: *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH)*. Portland, OR, USA: Association for

Computational Linguistics, pp. 10–18. URL: http://www.aclweb.org/anthology/W11-1502 (cited on p. 6).

Hanson, Stephen J. and Olson, Carl R. (1991). "A Review of: "Neural Networks and Natural Intelligence: Notes from Mudville" by Stephen Grossberg (Ed.)" *Connection Science*, 3(3), pp. 332–335. DOI: 10.1080/09540099108946591 (cited on p. 73).

Hashem, Sherif (1997). "Optimal Linear Combinations of Neural Networks". *Neural Networks*, 10(4), pp. 599–614. DOI: 10.1016/S0893-6080(96)00098-6 (cited on p. 106).

Hauser, Andreas W. and Schulz, Klaus U. (2007). "Unsupervised Learning of Edit Distance Weights for Retrieving Historical Spelling Variations". In: *Proceedings of the First Workshop on Finite-State Techniques and Approximate Search (FSTAS 2007)*. Borovets, Bulgaria, pp. 1–6 (cited on pp. 7, 68).

Hauser, Andreas, Heller, Markus, Leiss, Elisabeth, Schulz, Klaus U., and Wanzeck, Christiane (2007). "Information Access to Historical Documents from the Early New High German Period". In: *Digital Historical Corpora – Architecture, Annotation, and Retrieval*. Ed. by Lou Burnard, Milena Dobreva, Norbert Fuhr, and Anke Lüdeling. Dagstuhl Seminar Proceedings 06491. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL: http://drops.dagstuhl.de/opus/volltexte/2007/1057 (cited on p. 66).

Helgadóttir, Sigrún, Svavarsdóttir, Ásta, Rögnvaldsson, Eiríkur, Bjarnadóttir, Kristín, and Loftsson, Hrafn (2012). "The Tagged Icelandic Corpus (MÍM)". In: *Proceedings of the Workshop on Language Technology for Normalisation of Less-Resourced Languages*. Istanbul, Turkey, pp. 67–72 (cited on p. 57).

Hendrickx, Iris and Marquilhas, Rita (2011). "From old texts to modern spellings: an experiment in automatic normalisation". *Journal for Language Technology and Computational Linguistics (JLCL)*, 26(2), pp. 65–76 (cited on p. 18).

Hoberg, Ursula and Hoberg, Rudolf (1975). "Ïiebe genossen an einer schönen brust. oder: Erfordert die Struktur der deutschen Sprache die Großschreibung?" In: *Sprachsystem und Sprachgebrauch, Teil 2*. Ed. by Paul Grebe and Ulrich Engel. Vol. 34. Sprache der Gegenwart. Düsseldorf: Schwann, pp. 154–171. URL: https://ids-pub.bsz-bw.de/frontdoor/index/index/docId/2108 (cited on p. 26).

Hochreiter, Sepp and Schmidhuber, Jürgen (1997). "Long Short-Term Memory". *Neural Computation*, 9, pp. 1735–1780 (cited on pp. xviii, 78, 181).

Horn, Franziska (2017). "Context encoders as a simple but powerful extension of word2vec". In: *Proceedings of the 2nd Workshop on Representation Learning for NLP*. Vancouver, Canada: Association for Computational Linguistics, pp. 10–14. URL: http://aclweb.org/anthology/W17-2602 (cited on p. 183).

Huang, Zhiheng, Xu, Wei, and Yu, Kai (2015). "Bidirectional LSTM-CRF Models for Sequence Tagging". *CoRR*, abs/1508.01991. URL: http://arxiv.org/abs/1508.01991 (cited on p. 81).

Hundt, Marianne, Denison, David, and Schneider, Gerold (2012). "Retrieving relatives from historical data". *Literary and Linguistic Computing*, 27(1), pp. 3–16. DOI: 10.1093/llc/fqr049 (cited on p. 15).

Jin, Huiming and Kann, Katharina (2017). "Exploring Cross-Lingual Transfer of Morphological Knowledge In Sequence-to-Sequence Models". In: *Proceedings of the First Workshop on Subword and Character Level Models in NLP*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 70–75. URL: http://www.aclweb.org/anthology/W17-4110 (cited on p. 148).

Jurish, Bryan (2010a). "Comparing Canonicalizations of Historical German Text". In: *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*. Uppsala, Sweden: Association for Computational Linguistics, pp. 72–77. URL: http://www.aclweb.org/anthology/W10-2209 (cited on pp. 7, 68).

– (2010b). "More than words: using token context to improve canonicalization of historical German". *Journal for Language Technology and Computational Linguistics*, 25(1), pp. 23–39. URL: http://www.jlcl.org/2010_Heft1/bryan_jurish.pdf (cited on p. 183).

– (2011). "Finite-State Canonicalization Techniques for Historical German". Doctoral dissertation. Potsdam, Germany: University of Potsdam. URL: http://opus.kobv.de/ubp/volltexte/2012/5578/ (cited on p. 18).

Kalchbrenner, Nal and Blunsom, Phil (2013). "Recurrent Continuous Translation Models". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 1700–1709. URL: http://www.aclweb.org/anthology/D13-1176 (cited on p. 85).

Kann, Katharina, Cotterell, Ryan, and Schütze, Hinrich (2017). "One-Shot Neural Cross-Lingual Transfer for Paradigm Completion". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1993–2003. DOI: 10.18653/v1/P17-1182 (cited on p. 148).

Kempken, Sebastian, Luther, Wolfram, and Pilz, Thomas (2006). "Comparison of distance measures for historical spelling variants". In: *Artificial Intelligence in Theory and Practice*. Ed. by Max Bramer. Boston, MA: Springer, pp. 295–304. DOI: 10.1007/978-0-387-34747-9_31 (cited on pp. xvii, 68).

Kestemont, Mike, Daelemans, Walter, and De Pauw, Guy (2010). "Weigh your words—memory-based lemmatization for Middle Dutch". *Literary and Linguistic Computing*, 25(3), pp. 287–301. DOI: 10.1093/llc/fqq011 (cited on p. 68).

Kestemont, Mike, Pauw, Guy de, Nie, Renske van, and Daelemans, Walter (2016). "Lemmatization for variation-rich languages using deep learning". *Digital Scholarship in the Humanities*. DOI: 10.1093/llc/fqw034 (cited on p. 70).

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. (2016). "Character-Aware Neural Language Models". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*. Phoenix, AZ, pp. 2741–2749 (cited on p. 76).

Kingma, Diederik P. and Ba, Jimmy (2014). "Adam: A Method for Stochastic Optimization". *CoRR*, abs/1412.6980. URL: http://arxiv.org/abs/1412.6980 (cited on pp. 82, 99).

Kiperwasser, Eliyahu and Goldberg, Yoav (2016). "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations". *Transactions of the Association of Computational Linguistics*, 4, pp. 313–327. URL: http://www.aclweb.org/anthology/Q16-1023 (cited on p. 81).

Klein, Thomas (1991). "Zur Frage der Korpusbildung und zur computergestützten grammatischen Auswertung mittelhochdeutscher Quellen". In: *Mittelhochdeutsche Grammatik als Aufgabe (Zeitschrift für deutsche Philologie)*. Ed. by Klaus-Peter Wegera. Vol. 110. Berlin: Schmidt, pp. 3–23 (cited on pp. xv, 5).

Klein, Thomas and Dipper, Stefanie (2016). "Handbuch zum Referenzkorpus Mittelhochdeutsch". *Bochumer Linguistische Arbeitsberichte*, 19. URL: https://www.linguistics.rub.de/bla/019-klein-dipper2016.pdf (cited on pp. 2, 5, 16).

Klerke, Sigrid, Goldberg, Yoav, and Søgaard, Anders (2016). "Improving sentence compression by learning to predict gaze". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1528–1533. DOI: 10.18653/v1/N16-1179 (cited on p. 145).

Knight, Kevin and Graehl, Jonathan (1998). "Machine transliteration". *Computational Linguistics*, 24(4), pp. 599–612 (cited on p. 65).

Koehn, Philipp (2005). "Europarl: A Parallel Corpus for Statistical Machine Translation". In: *Proceedings of MT Summit*. URL: http://www.iccs.inf.ed.ac.uk/~pkoehn/publications/europarl-mtsummit05.pdf (cited on p. 56).

Koehn, Philipp, Hoang, Hieu, Birch, Alexandra, Callison-Burch, Chris, Federico, Mercello, Bertoldi, Nicola, Cowan, Brooke, Shen, Wade, Moran, Christine, Zens, Richard, Dyer, Chris, Bojar, Ondřej, Constantin, Alexandra, and Herbst, Evan (2007). "Moses: Open Source Toolkit for Statistical Machine Translation". In: *Proceedings of the ACL 2007 Demo and Poster Sessions*. Prague, Czech Republic, pp. 177–180 (cited on pp. 69, 72).

Koller, Gerhard (1983). "Ein maschinelles Verfahren zur Normalisierung altdeutscher Texte". In: *Germanistik in Erlangen. Hundert Jahre nach der Gründung des Deutschen Seminars*. Ed. by Dietmar Peschel. Vol. 31. Erlanger Forschungen. Erlangen: Universitätsbund Erlangen-Nürnberg, pp. 611–620 (cited on pp. xv, xvi, 5, 7, 17, 66).

Koolen, Marijn, Adriaans, Frans, Kamps, Jaap, and Rijke, Maarten de (2006). "A Cross-Language Approach to Historic Document Retrieval". In: *Proceedings of the 28th European Conference on Information Retrieval Research (ECIR 2006)*. Lecture Notes in Computer Science. Berlin: Springer, pp. 407–419. DOI: 10.1007/11735106 (cited on pp. 7, 67).

Korchagina, Natalia (2017). "Normalizing Medieval German Texts: from rules to deep learning". In: *Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language*. Gothenburg, Sweden: Linköping University Electronic Press, pp. 12–17. URL: http://www.aclweb.org/anthology/W17-0504 (cited on pp. xvii, xx, 9, 10, 70, 71, 180).

Korpushandbuch DDD-Mittelhochdeutsch (2014). *Texterfassung, Lemmatisierung, grammatische Annotierung*. Internal report. Bonn (cited on p. 16).

Krasselt, Julia (2017). "Der Verbalkomplex im Frühneuhochdeutschen. Eine korpuslinguistische Untersuchung zur Serialisierung zwei- und dreigliedriger Verbalkomplexe". *Bochumer Linguistische Arbeitsberichte*, 21. URL: https://www.linguistics.rub.de/bla/021-krasselt2017.pdf (cited on p. 15).

Krasselt, Julia, Bollmann, Marcel, Dipper, Stefanie, and Petran, Florian (2015). "Guidelines für die Normalisierung historischer deutscher Texte / Guidelines for Normalizing Historical German Texts". *Bochumer Linguistische Arbeitsberichte*, 15. URL: https://www.linguistics.rub.de/bla/015-krasselt_etal2015.pdf (cited on pp. 18–24, 32, 33).

Laing, Margaret (1994). "The linguistic analysis of medieval vernacular texts: Two projects at Edinburgh". In: *Corpora Across the Centuries. Proceedings of the First International Colloquium on English Diachronic Corpora*. Ed. by Merja Kytö, Matti Rissanen, and Susan Wright. Amsterdam: Rodopi, pp. 121–142 (cited on pp. xv, 4, 5).

Lample, Guillaume, Ballesteros, Miguel, Subramanian, Sandeep, Kawakami, Kazuya, and Dyer, Chris (2016). "Neural Architectures for Named Entity Recognition". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 260–270. DOI: 10.18653/v1/N16-1030 (cited on pp. 79, 81).

Lee, Jason, Cho, Kyunghyun, and Hofmann, Thomas (2017). "Fully Character-Level Neural Machine Translation without Explicit Segmentation". *Transactions of the Association of Computational Linguistics*, 5, pp. 365–378. URL: http://aclweb.org/anthology/Q17-1026 (cited on pp. 70, 182, 183).

Lenders, Winfried, Lutz, Hans-Dieter, and Römer, Ruth (1973). *Untersuchungen zur automatischen Indizierung mittelhochdeutscher Texte*. 2nd ed. Hamburg: Buske (cited on p. 17).

Levenshtein, Vladimir I. (1966). "Binary codes capable of correcting deletions, insertions, and reversals". *Soviet Physics Doklady*, 10(8), pp. 707–710 (cited on pp. 64, 68).

Lexer, Matthias (1992). *Mittelhochdeutsches Taschenwörterbuch. Mit den Nachträgen von Ulrich Pretzel*. 38th. Stuttgart: S. Hirzel (cited on pp. 17, 23, 33).

Li, Haizhou, Kumaran, A., Pervouchine, Vladimir, and Zhang, Min (2009). "Report of NEWS 2009 Machine Transliteration Shared Task". In: *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*. Suntec, Singapore: Association for Computational Linguistics, pp. 1–18. URL: http://aclweb.org/anthology/W09-3501 (cited on p. 65).

Lin, Rui, Liu, Shujie, Yang, Muyun, Li, Mu, Zhou, Ming, and Li, Sheng (2015). "Hierarchical Recurrent Neural Network for Document Modeling". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 899–907. DOI: 10.18653/v1/D15-1106 (cited on p. 183).

Ling, Wang, Trancoso, Isabel, Dyer, Chris, and Black, Alan W. (2015). "Character-based Neural Machine Translation". *CoRR*, abs/1511.04586. URL: http://arxiv.org/abs/1511.04586 (cited on pp. 8, 69, 70, 93, 180).

Liu, Lemao, Finch, Andrew, Utiyama, Masao, and Sumita, Eiichiro (2016). "Agreement on Target-Bidirectional LSTMs for Sequence-to-Sequence Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*. Phoenix, Arizona, pp. 2630–2637. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/viewPaper/12028 (cited on p. 181).

Liu, Lemao, Utiyama, Masao, Finch, Andrew, and Sumita, Eiichiro (2016). "Neural Machine Translation with Supervised Attention". In: *Proceedings of COLING 2016, the 26th International*

Conference on Computational Linguistics: Technical Papers. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 3093–3102. URL: http://www.aclweb.org/anthology/C16-1291 (cited on p. 182).

Ljubešić, Nikola, Erjavec, Tomaž, and Fišer, Darja (2017). "Adapting a State-of-the-Art Tagger for South Slavic Languages to Non-Standard Text". In: *Proceedings of the 6th Workshop on Balto-Slavic Natural Language Processing*. Valencia, Spain: Association for Computational Linguistics, pp. 60–68. URL: http://aclweb.org/anthology/W17-1410 (cited on p. 183).

Ljubešić, Nikola, Zupan, Katja, Fišer, Darja, and Erjavec, Tomaž (2016a). *Dataset of normalised Slovene text KonvNormSl 1.0*. Slovenian language resource repository CLARIN.SI. URL: http://hdl.handle.net/11356/1068 (cited on p. 36).

– (2016b). "Normalising Slovene data: historical texts vs. user-generated content". In: *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016)*. Vol. 16. Bochumer Linguistische Arbeitsberichte. Bochum, Germany, pp. 146–155. URL: https://www.linguistics.rub.de/konvens16/pub/19_konvensproc.pdf (cited on pp. xvii, 8, 18, 28, 29, 35, 36, 69, 72, 114, 183).

Lüdeling, Anke, Odebrecht, Carolin, and Zeldes, Amir (2017). *RIDGES-Herbology*. Version 6.0. URL: https://korpling.org/ridges/ (cited on p. 33).

Luong, Minh-Thang, Le, Quoc V., Sutskever, Ilya, Vinyals, Oriol, and Kaiser, Lukasz (2015). "Multi-task Sequence to Sequence Learning". *CoRR*, abs/1511.06114. URL: http://arxiv.org/abs/1511.06114 (cited on pp. 145, 146, 183).

Luong, Minh-Thang and Manning, Christopher D. (2016). "Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1054–1063. DOI: 10.18653/v1/P16-1100. URL: http://www.aclweb.org/anthology/P16-1100 (cited on p. 183).

Luong, Thang, Pham, Hieu, and Manning, Christopher D. (2015). "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1412–1421. DOI: 10.18653/v1/D15-1166 (cited on pp. 88, 93, 182).

Lusetti, Massimo, Ruzsics, Tatyana, Göhring, Anne, Samardžić, Tanja, and Stark, Elisabeth (2018). "Encoder-Decoder Methods for Text Normalization". In: *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2018)*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 18–28. URL: http://aclweb.org/anthology/W18-3902 (cited on p. 70).

Marcus, Gary (2018). "Deep Learning: A Critical Appraisal". *CoRR*, abs/1801.00631. URL: http://arxiv.org/abs/1801.00631 (cited on p. 157).

Markus, Manfred (1997). "Normalization of Middle English prose in practice". In: *Corpus-based Studies in English. Papers from the seventeenth International Conference on English Language Research on Computerized Corpora (ICAME 17)*. Ed. by Magnus Ljung. Amsterdam: Rodopi, pp. 211–226 (cited on pp. 17, 18, 21, 30).

– (1999). "Manual of ICAMET (Innsbruck Computer Archive of Machine-Readable English Texts)". *Innsbrucker Beiträge zur Kulturwissenschaft, Anglistische Reihe*, 7 (cited on pp. 28, 30).

Markus, Manfred (2000). "Normalising the Word Forms in the *Ayenbite of Inwit*". In: *Placing Middle English in Context*. Ed. by Irma Taavitsainen, Terttu Nevalainen, Päivi Pahta, and Matti Rissanen. Berlin: Mouton de Gruyter, pp. 181–198 (cited on pp. 13, 14, 17, 18, 30).

Marttila, Ville (2014). "Creating Digital Editions for Corpus Linguistics. The case of *Potage Dyvers*, a family of six Middle English recipe collections". Doctoral dissertation. Helsinki, Finland: University of Helsinki, Department of Modern Languages. URL: http://urn.fi/URN:ISBN:978-951-51-0060-3 (cited on pp. 16, 18, 19, 21, 23).

Müller, Hans-Georg (2016). *Der Majuskelgebrauch im Deutschen. Groß- und Kleinschreibung theoretisch, empirisch, ontogenetisch*. Vol. 305. Reihe Germanistische Linguistik. Berlin/Boston: De Gruyter. URL: https://books.google.de/books?id=-KNlCwAAQBAJ (cited on p. 26).

Nakov, Preslav and Tiedemann, Jörg (2012). "Combining Word-Level and Character-Level Models for Machine Translation Between Closely-Related Languages". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Jeju Island, Korea: Association for Computational Linguistics, pp. 301–305. URL: http://www.aclweb.org/anthology/P12-2059 (cited on p. 69).

Och, Franz Josef and Ney, Hermann (2003). "A Systematic Comparison of Various Statistical Alignment Models". *Computational Linguistics*, 29(1), pp. 19–51 (cited on p. 72).

Odebrecht, Carolin, Belz, Malte, Zeldes, Amir, Lüdeling, Anke, and Krause, Thomas (2016). "RIDGES Herbology: designing a diachronic multi-layer corpus". *Language Resources and Evaluation*, pp. 1–31. DOI: 10.1007/s10579-016-9374-3 (cited on pp. xvi, 5, 16, 19, 22, 28, 29, 33).

Olah, Christopher (2015). *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 11/01/2017) (cited on p. 79).

Oravecz, Csaba, Sass, Bálint, and Simon, Eszter (2010). "Semi-automatic normalization of Old Hungarian Codices". In: *Proceedings of the ECAI 2010 Workshop on Language TEchnology for Cultural Heritage*, pp. 55–59 (cited on pp. 18, 34, 69).

Östling, Robert (2016). "Morphological reinflection with convolutional neural networks". In: *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Berlin, Germany: Association for Computational Linguistics, pp. 23–26. DOI: 10.18653/v1/W16-2003 (cited on p. 182).

Papineni, Kishore, Roukos, Salim, Ward, Todd, and Zhu, Wei-Jing (2002). "BLEU: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. URL: http://www.aclweb.org/anthology/P02-1040 (cited on p. 118).

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, Vanderplas, Jake, Passos, Alexandre, Cournapeau, David, Brucher, Matthieu, Perrot, Matthieu, and Duchesnay, Éduard (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research*, 12, pp. 2825–2830 (cited on p. 133).

Pettersson, Eva (2016). "Spelling Normalisation and Linguistic Analysis of Historical Text for Information Extraction". Doctoral dissertation. Uppsala: Uppsala University, Department

of Linguistics and Philology. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-269753 (cited on pp. 19, 20, 22, 23, 28–30, 34, 35, 38, 57, 69).

Pettersson, Eva, Megyesi, Beáta, and Nivre, Joakim (2012). "Rule-Based Normalisation of Historical Text – A Diachronic Study". In: *Proceedings of the 11th Conference on Natural Language Processing (KONVENS 2012), LThist 2012 workshop*. Vienna, Austria, pp. 333–341. URL: http://www.oegai.at/konvens2012/proceedings/50_pettersson12w/ (cited on p. 38).

– (2013). "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)*. Oslo, Norway: Linköping University Electronic Press, pp. 163–179. URL: http://www.aclweb.org/anthology/W13-5617 (cited on pp. xvii, 7, 18, 68, 133).

– (2014a). "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)*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 32–41. DOI: 10.3115/v1/W14-0605 (cited on pp. 29, 69, 104, 114, 118).

– (2014b). "Verb Phrase Extraction in a Historical Context". In: *The First Swedish National SWE-CLARIN Workshop, Swedish Language Technology Conference (SLTC)*. Uppsala, Sweden. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-239452 (cited on p. 15).

Pettersson, Eva, Megyesi, Beáta, and Tiedemann, Jörg (2013). "An SMT approach to automatic annotation of historical text". In: *Proceedings of the Workshop on Computational Historical Linguistics at NODALIDA 2013*. NEALT Proceedings Series 18/Linköping Electronic Conference Proceedings 87. Oslo, Norway: Linköping University Electronic Press, pp. 54–69. URL: http://www.ep.liu.se/ecp/article.asp?issue=087&article=005 (cited on pp. xvii, 7, 69).

Pilz, Thomas, Ernst-Gerlach, Andrea, Kempken, Sebastian, Rayson, Paul, and Archer, Dawn (2007). "The Identification of Spelling Variants in English and German Historical Texts: Manual or Automatic?" *Literary and Linguistic Computing*, 1(23), pp. 65–72. DOI: 10.1093/llc/fqm044 (cited on p. 64).

Pilz, Thomas, Luther, Wolfram, Fuhr, Norbert, and Ammon, Ulrich (2006). "Rule-based Search in Text Databases with Nonstandard Orthography". *Literary and Linguistic Computing*, 21(2), pp. 179–186. DOI: 10.1093/llc/fql020 (cited on p. 67).

Piotrowski, Michael (2012). *Natural Language Processing for Historical Texts*. Synthesis Lectures on Human Language Technologies 17. Morgan & Claypool. DOI: 10.2200/S00436ED1V01Y201207HLT017 (cited on pp. 3, 6, 15, 63).

Plank, Barbara (2016). "Keystroke dynamics as signal for shallow syntactic parsing". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 609–619. URL: http://www.aclweb.org/anthology/C16-1059 (cited on p. 145).

Plank, Barbara, Søgaard, Anders, and Goldberg, Yoav (2016). "Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 412–418. DOI: 10.18653/v1/P16-2067 (cited on p. 79).

Pollock, Joseph J. (1982). "Spelling error detection and correction by computer: some notes and a bibliography". *Journal of Documentation*, 38(4), pp. 282–291. DOI: 10.1108/eb026733 (cited on pp. 63, 64).

Porta, Jordi, Sancho, José-Luis, and Gómez, Javier (2013). "Edit transducers for spelling variation in Old Spanish". In: *Proceedings of the Workshop on Computational Historical Linguistics at NODALIDA 2013*. NEALT Proceedings Series 18/Linköping Electronic Conference Proceedings 87. Oslo, Norway: Linköping University Electronic Press, pp. 70–79. URL: http://www.ep.liu.se/ecp/article.asp?issue=087&article=006 (cited on p. 67).

Porter, Martin (2001). *Snowball: A language for stemming algorithms*. URL: http://snowballstem.org/texts/introduction.html (cited on p. 122).

Powers, David M. W. (2011). "Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation". *Journal of Machine Learning Technologies*, 2(1), pp. 37–63 (cited on pp. 136, 139).

Rao, Kanishka, Peng, Fuchun, Sak, Haşim, and Beaufays, Françoise (2015). "Grapheme-to-phoneme conversion using Long Short-Term Memory recurrent neural networks". In: *Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Brisbane, Australia, pp. 4225–4229. DOI: 10.1109/ICASSP.2015.7178767 (cited on p. 180).

Rayson, Paul, Archer, Dawn, Baron, Alistair, Culpeper, Jonathan, and Smith, Nicholas (2007). "Tagging the Bard: Evaluating the Accuracy of a Modern POS Tagger on Early Modern English Corpora". In: *Proceedings of the Corpus Linguistics Conference*. URL: http://eprints.lancs.ac.uk/13011/ (cited on pp. xv, 3).

Rayson, Paul, Archer, Dawn, and Smith, Nicholas (2005). "VARD versus Word: A comparison of the UCREL variant detector and modern spell checkers on English historical corpora". In: *Proceedings of the Corpus Linguistics Conference CL2005*. Birmingham, UK: University of Birmingham (cited on pp. xvi, 65).

Reimers, Nils and Gurevych, Iryna (2017). "Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 338–348. URL: http://aclweb.org/anthology/D17-1035 (cited on p. 103).

Reynaert, Martin, Hendrickx, Iris, and Marquilhas, Rita (2012). "Historical spelling normalization. A comparison of two statistical methods: TICCL and VARD2". In: *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities (ACRH-2)*. Lisbon, Portugal, pp. 87–98 (cited on pp. 18, 104).

Rissanen, Matti (2000). "The world of English historical corpora: From Cædmon to the computer age". *Journal of English Linguistics*, 28(1), pp. 7–20. DOI: 10.1177/00754240022004848 (cited on p. 27).

Robertson, Alexander M. and Willett, Peter (1993). "A Comparison of Spelling-Correction Methods for the Identification of Word Forms in Historical Text Databases". *Literary and Linguistic Computing*, 8(3), pp. 143–152. DOI: 10.1093/llc/8.3.143 (cited on pp. xvii, 68).

Robertson, Alexander and Goldwater, Sharon (2018). "Evaluating Historical Text Normalization Systems: How Well Do They Generalize?" In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 720–725. URL: http://aclweb.org/anthology/N18-2113 (cited on pp. 70, 182).

Rogers, Heather J. and Willett, Peter (1991). "Searching for historical word forms in text databases using spelling-correction methods: Reverse error and phonetic coding methods". *Journal of Documentation*, 47(4), pp. 333–353. DOI: 10.1108/eb026883 (cited on p. 14).

Rögnvaldsson, Eiríkur, Ingason, Anton Karl, Sigurðsson, Einar Freyr, and Wallenberg, Joel (2012). "The Icelandic Parsed Historical Corpus (IcePaHC)". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Ed. by Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Istanbul, Turkey: European Language Resources Association (ELRA), pp. 1977–1984 (cited on pp. 19, 28, 35).

Rosenblatt, Frank (1958). "The perceptron: A probabilistic model for information storage and organization in the brain". *Psychological Review*, 65(6), pp. 386–408. DOI: 10.1037/h0042519 (cited on p. 74).

Samardžić, Tanja, Scherrer, Yves, and Glaser, Elvira (2015). "Normalising orthographic and dialectal variants for the automatic processing of Swiss German". In: *Proceedings of the 7th Language and Technology Conference*. Poznań, Poland, pp. 294–298. URL: https://archive-ouverte.unige.ch/unige:82397 (cited on pp. 64, 69).

Sánchez-Marco, Cristina, Boleda, Gemma, and Padró, Lluís (2011). "Extending the tool, or how to annotate historical language varieties". In: *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH)*. Portland, OR, USA: Association for Computational Linguistics, pp. 1–9. URL: http://www.aclweb.org/anthology/W11-1501 (cited on p. 6).

Sánchez-Martínez, Felipe, Martínez-Sempere, Isabel, Ivars-Ribes, Xavier, and Carrasco, Rafael C. (2013). "An open diachronic corpus of historical Spanish: annotation criteria and automatic modernisation of spelling". *CoRR*, abs/1306.3692. URL: http://arxiv.org/abs/1306.3692 (cited on pp. 7, 18, 69).

Sang, Erik Tjong Kim (2016). "Improving Part-of-Speech Tagging of Historical Text by First Translating to Modern Text". In: *Computational History and Data-Driven Humanities*. Ed. by Bojan Bozic, Gavin Mendel-Gleason, Christophe Debruyne, and Declan O'Sullivan. Cham: Springer International Publishing, pp. 54–64. DOI: 10.1007/978-3-319-46224-0_6 (cited on p. 183).

Scheible, Silke, Whitt, Richard J., Durrell, Martin, and Bennett, Paul (2011a). "A Gold Standard Corpus of Early Modern German". In: *Proceedings of the 5th Linguistic Annotation Workshop*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 124–128. URL: http://www.aclweb.org/anthology/W11-0415 (cited on p. 9).

– (2011b). "Evaluating an 'off-the-shelf' POS-tagger on Early Modern German text". In: *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social*

Sciences, and Humanities. Portland, OR, USA: Association for Computational Linguistics, pp. 19–23. URL: http://www.aclweb.org/anthology/W11-1503 (cited on pp. xv, 3, 183).

Scherrer, Yves and Erjavec, Tomaž (2013). "Modernizing historical Slovene words with character-based SMT". In: *Proceedings of the 4th Biennial International Workshop on Balto-Slavic Natural Language Processing*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 58–62. URL: http://www.aclweb.org/anthology/W13-2409 (cited on pp. xvii, 7, 18, 69).

– (2016). "Modernising historical Slovene words". *Natural Language Engineering*, 22(6), pp. 881–905. DOI: 10.1017/S1351324915000236 (cited on pp. 35, 67, 69).

Scherrer, Yves and Ljubešić, Nikola (2016). "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)*. Vol. 16. Bochumer Linguistische Arbeitsberichte. Bochum, Germany, pp. 248–255. URL: https://www.linguistics.rub.de/konvens16/pub/32_konvensproc.pdf (cited on pp. 64, 69).

Schneider, Gerold, Pettersson, Eva, and Percillier, Michael (2017). "Comparing Rule-based and SMT-based Spelling Normalisation for English Historical Texts". In: *Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language*. Gothenburg, Sweden: Linköping University Electronic Press, pp. 40–46. URL: http://www.aclweb.org/anthology/W17-0508 (cited on pp. 8, 18, 69).

Schnober, Carsten, Eger, Steffen, Do Dinh, Erik-Lân, and Gurevych, Iryna (2016). "Still not there? Comparing Traditional Sequence-to-Sequence Models to Encoder-Decoder Neural Networks on Monotone String Translation Tasks". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 1703–1714. URL: http://www.aclweb.org/anthology/C16-1160 (cited on p. 180).

Schultz-Balluff, Simone and Dipper, Stefanie (2013a). "St. Anselmi Fragen an Maria – Schritte zu einer (digitalen) Erschließung, Auswertung und Edition der gesamten deutschsprachigen Überlieferung (14.–16. Jh.)" In: *Medienwandel/Medienwechsel in der Editionswissenschaft. 13. internationale Tagung der Arbeitsgemeinschaft für germanistische Edition 17.–20. Februar 2010, Beihefte zu editio*. Ed. by Anne Bohnenkamp-Renken. Beihefte zu editio 35. Berlin/New York: de Gruyter, pp. 177–196 (cited on pp. xvi, 31).

– (2013b). "St. Anselmi Fragen an Maria – Schritte zu einer (digitalen) Erschließung, Auswertung und Edition der gesamten deutschsprachigen Überlieferung (14.–16. Jh.)" In: *Medienwandel/Medienwechsel in der Editionswissenschaft*. Ed. by Anne Bohnenkamp-Renken. Beihefte zu editio 35. Berlin/Boston: de Gruyter (cited on p. 2).

Schuster, Mike and Paliwal, Kuldip K. (1997). "Bidirectional Recurrent Neural Networks". *IEEE Transactions on Signal Processing*, 45(11), pp. 2673–2681 (cited on p. 80).

Shen, Shiqi, Cheng, Yong, He, Zhongjun, He, Wei, Wu, Hua, Sun, Maosong, and Liu, Yang (2016). "Minimum Risk Training for Neural Machine Translation". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1683–1692. DOI: 10.18653/v1/P16-1159. URL: http://www.aclweb.org/anthology/P16-1159 (cited on p. 182).

Simon, Eszter (2014). "Corpus building from Old Hungarian codices". In: *The Evolution of Functional Left Peripheries in Hungarian Syntax*. Ed. by Katalin É. Kiss. Oxford, UK: Oxford University Press, pp. 224–236 (cited on pp. 5, 9, 15, 16, 20, 22, 23, 28, 34, 40).

Smith, Noah A. (2011). *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies 13. Morgan & Claypool. DOI: 10.2200/S00361ED1V01Y201105HLT013 (cited on p. 133).

Søgaard, Anders and Goldberg, Yoav (2016). "Deep multi-task learning with low level tasks supervised at lower layers". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 231–235. DOI: 10.18653/v1/P16-2038 (cited on p. 183).

Springmann, Uwe and Lüdeling, Anke (2017). "OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus". *Digital Humanities Quarterly*, 11(2). URL: http://www.digitalhumanities.org/dhq/vol/11/2/000288/000288.html (cited on p. 15).

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research*, 15, pp. 1929–1958. URL: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf (cited on p. 83).

Susanto, Hendy Raymond, Chieu, Leong Hai, and Lu, Wei (2016). "Learning to Capitalize with Character-Level Recurrent Neural Networks: An Empirical Study". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2090–2095. URL: http://aclweb.org/anthology/D16-1225 (cited on p. 26).

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. (2014). "Sequence to Sequence Learning with Neural Networks". *CoRR*, abs/1409.3215. URL: http://arxiv.org/abs/1409.3215 (cited on pp. xviii, 79, 85, 86, 93, 179, 180).

Svensson, Patrik (2010). "The Landscape of Digital Humanities". *Digital Humanities*, 4(1). URL: http://digitalhumanities.org/dhq/vol/4/1/000080/000080.html (cited on p. 1).

Tang, Gongbo, Cap, Fabienne, Pettersson, Eva, and Nivre, Joakim (2018). "An Evaluation of Neural Machine Translation Models on Historical Spelling Normalization". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 1320–1331. URL: http://aclweb.org/anthology/C18-1112 (cited on pp. 70, 114).

Tiedemann, Jörg and Nabende, Peter (2009). "Translating Transliterations". *International Journal of Computing and ICT Research*, 3(1) (Special Issue), pp. 33–41 (cited on p. 69).

Tjandra, Andros, Sakti, Sakriani, and Nakamura, Satoshi (2017). "Local Monotonic Attention Mechanism for End-to-End Speech And Language Processing". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, pp. 431–440. URL: http://aclweb.org/anthology/I17-1044 (cited on p. 182).

Vaamonde, Gael (2017). *Userguide for digital edition of texts in P. S. Post Scriptum*. Translated by Clara Pinto. Guidelines. URL: http://ps.clul.ul.pt/files/Manual_PS_english.pdf (cited on pp. 16, 25, 28, 29, 37).

Vaamonde, Gael and Magro, Catarina (2017). *Manual de Edición y Anotación en TEITOK de los Materiales de P.S. Post Scriptum*. Guidelines. URL: http://ps.clul.ul.pt/files/Manual_Mod_Pos_Sin.pdf (cited on pp. 20–22, 24, 37).

Vilar, David, Peter, Jan-Thorsten, and Ney, Hermann (2007). "Can We Translate Letters?" In: *Proceedings of the Second Workshop on Statistical Machine Translation*. Prague, Czech Republic: Association for Computational Linguistics, pp. 33–39. URL: http://www.aclweb.org/anthology/W07-0705 (cited on p. 69).

Vlad Lita, Lucian, Ittycheriah, Abe, Roukos, Salim, and Kambhatla, Nanda (2003). "tRuEcasIng". In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan: Association for Computational Linguistics. URL: http://aclweb.org/anthology/P03-1020 (cited on p. 26).

Vosoughi, Soroush, Vijayaraghavan, Prashanth, and Roy, Deb (2016). "Tweet2Vec: Learning Tweet Embeddings Using Character-level CNN-LSTM Encoder-Decoder". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. Pisa, Italy, pp. 1041–1044. DOI: 10.1145/2911451.2914762 (cited on p. 85).

Wang, Wei, Knight, Kevin, and Marcu, Daniel (2006). "Capitalizing Machine Translation". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York, NY: Association for Computational Linguistics, pp. 1–8. URL: http://aclweb.org/anthology/N06-1001 (cited on p. 26).

Wegera, Klaus-Peter (2014). "Interrogatio Sancti Anselmi de Passione Domini, deutsch. Überlieferung – Edition – Perspektiven der Auswertung". In: *Veröffentlichungen der Nordrhein-Westfälischen Akademie der Wissenschaften und der Künste*. Vol. 445. Paderborn (cited on pp. xvi, 28, 31).

Wieling, Martijn, Prokić, Jelena, and Nerbonne, John (2009). "Evaluating the pairwise string alignment of pronunciations". In: *Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education (LaTeCH – SHELT&R 2009)*. Athens, Greece, pp. 26–34 (cited on p. 40).

Wilson, Ashia C., Roelofs, Rebecca, Stern, Mitchell, Srebro, Nathan, and Recht, Benjamin (2017). "The Marginal Value of Adaptive Gradient Methods in Machine Learning". URL: http://arxiv.org/abs/1705.08292 (cited on p. 82).

Wiseman, Sam and Rush, Alexander M. (2016). "Sequence-to-Sequence Learning as Beam-Search Optimization". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 1296–1306. DOI: 10.18653/v1/D16-1137 (cited on pp. 87, 182).

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, Klingner, Jeff, Shah, Apurva, Johnson, Melvin, Liu, Xiaobing, Kaiser, Lukasz, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Kazawa, Hideto, Stevens, Keith, Kurian, George, Patil, Nishant, Wang, Wei, Young, Cliff, Smith, Jason, Riesa, Jason, Rudnick, Alex, Vinyals, Oriol, Corrado, Greg, Hughes, Macduff,

and Dean, Jeffrey (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". *CoRR*, abs/1609.08144. URL: http://arxiv.org/abs/1609.08144 (cited on pp. xx, 8, 9, 70, 91).

Xie, Ziang, Avati, Anand, Arivazhagan, Naveen, Jurafsky, Dan, and Ng, Andrew Y. (2016). "Neural Language Correction with Character-Based Attention". *CoRR*, abs/1603.09727. URL: http://arxiv.org/abs/1603.09727 (cited on p. 180).

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhudinov, Ruslan, Zemel, Rich, and Bengio, Yoshua (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 2048–2057. URL: http://proceedings.mlr.press/v37/xuc15.html (cited on pp. 88, 89).

Yang, Yi and Eisenstein, Jacob (2016). "Part-of-Speech Tagging for Historical English". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1318–1328. DOI: 10.18653/v1/N16-1157 (cited on p. 183).

Yang, Zhilin, Salakhutdinov, Ruslan, and Cohen, William W. (2016). "Multi-Task Cross-Lingual Sequence Tagging from Scratch". *CoRR*, abs/1603.06270. URL: http://arxiv.org/abs/1603.06270 (cited on p. 183).

Young, Tom, Hazarika, Devamanyu, Poria, Soujanya, and Cambria, Erik (2017). "Recent Trends in Deep Learning Based Natural Language Processing". *CoRR*, abs/1708.02709. URL: http://arxiv.org/abs/1708.02709 (cited on p. 8).

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol (2014). "Recurrent Neural Network Regularization". *CoRR*, abs/1409.2329. URL: http://arxiv.org/abs/1409.2329 (cited on p. 99).

Bildungsgang des Autors

1984	Geboren in Dortmund
2004	Abitur, Reinoldus- und Schiller-Gymnasium, Dortmund
2004–2009	Bachelor of Arts (B.A.) in den Fächern Mathematik und Linguistik (Schwerpunkt Computerlinguistik), Ruhr-Universität Bochum
2009–2012	Master of Arts (M.A.) im Fach Linguistik (Schwerpunkt Computerlinguistik), Ruhr-Universität Bochum
2011–2017	Wissenschaftlicher Mitarbeiter in verschiedenen DFG-geförderten Projekten unter Leitung von Prof. Stefanie Dipper: • St. Anselmi Fragen an Maria. Digitale Erschließung, Auswertung und Edition der gesamten deutschsprachigen Überlieferung (14.–16. Jh.) • Referenzkorpus Frühneuhochdeutsch • Referenzkorpus Mittelhochdeutsch
2018	Promotion im Fach Computerlinguistik, Ruhr-Universität Bochum
seit 2018	Postdoc in der CoAStaL NLP Group unter Prof. Anders Søgaard, Universität Kopenhagen