Statistical and Computational Models for Whole Word Morphology

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig angenommene

Dissertation
zur Erlangung des akademischen Grades
Doctor rerum naturalium
(Dr. rer. nat.)
im Fachgebiet Informatik

vorgelegt von
Maciej Janicki, M. Sc.
geboren am 10.08.1989 in Wrocław, Polen

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Gerhard Heyer (Universität Leipzig)
2. Prof. Dr. Uwe Quasthoff (Universität Leipzig)
3. Dr. Krister Lindén (Universität Helsinki)

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 13. August 2019 mit dem Gesamtprädikat magna cum laude.

Abstract

The purpose of this thesis is to provide an unsupervised machine learning approach for language morphology, in which the latter is modeled as string transformations on whole words, rather than the segmentation of words into smaller structural units.

The choice of a transformation-based morphology model is motivated mainly by the realization that most Natural Language Processing methods treat words as basic units of language, and thus the application of morphological analysis is usually centered at describing *relations* between words, rather than processing subword units. On the other hand, segmentational approaches sometimes require answering questions that are both difficult and irrelevant for most applications, such as whether to segment the German word *fähig* 'able' into *fäh-ig* or the English *suicide* into *sui-cide*. Furthermore, non-concatenative phenomena like German umlaut are not covered by segmentational approaches.

The whole-word-based model of morphology employed in this work is rooted in linguistic theories rejecting the notion of morpheme, particularly Whole Word Morphology. In this theory, it is *words* that are minimal elements of the language combining form and meaning. Morphology is expressed in terms of patterns describing a regular interrelation in form and meaning within pairs of words. Such formulation naturally results in a representation of the lexicon as a graph, in which words are vertices and pairs of words related via a productive, systematic pattern are connected with an edge.

The contribution of the present thesis is twofold. Firstly, I provide a *computational model* for Whole Word Morphology. Its foundation is a formal definition of *morphological rule* as a function mapping strings to sets of strings (as the application of a rule to a word may result in multiple outcomes or no outcome). The rules are formulated in terms of patterns, which consist of alternating constant elements – describing parts of the word that are affected by the rule – and variable elements,

which stand for parts of the word unaffected by the rule. For example, the rule mapping the German word *Haus*$_{\text{N.SG}}$ 'house' to *Häuser*$_{\text{N.PL}}$ 'houses' has the form: $/X_1\text{a}X_2/_{\text{N.SG}} \rightarrow /X_1\text{ä}X_2\text{er}/_{\text{N.PL}}$, where X_1 and X_2 are variables, which can be instantiated with any string. The rules formulated this way can be easily expressed as Finite State Transducers, which enables us to use readily available, performant algorithms for tasks like disjunction of rules or application of a rule to a word or set of words. For the task of rule induction, i.e. extracting rules from pairs of similar words, I propose an approach based on modified versions of algorithms related to edit distance (Fast Similarity Search and the Wagner-Fischer algorithm).

Secondly, I propose a *statistical model* for graphs of word derivations. It is a generative model defining a joint probability distribution over the vertices and edges of the graph. The model is parameterized by a set of rules, as well as a vector of numeric parameters, from which the probability of application of a particular rule to a particular word can be computed. Of the two inference problems for this model – *fitting* consisting of finding optimal values for the numeric parameters and *model selection* corresponding to selecting an optimal set of rules – especially fitting is considered in detail. It is realized by the Monte Carlo Expectation Maximization algorithm, which iteratively maximizes the expected log-likelihood of the graph. In order to approximate expected values over all possible configurations of graph edges, a sampler based on Metropolis-Hastings algorithm is developed.

Once trained, the model can be applied to a variety of tasks. The experiments presented in the thesis include inducing lexemes (sets of word forms corresponding to the same lemma) by means of graph clustering, predicting new words to reduce out-of-vocabulary word rates and predicting part-of-speech tags for unknown words after unsupervised training on a tagged dataset. Although developed for unsupervised training, the model can also be trained in a supervised setting, producing reasonable results in tasks like lemmatization or inflected form generation. However, its main strength lies in a direct description and generalization of regularities amongst words encountered in unlabeled data, without referring to any linguistic notion of 'morphological structure' or 'analysis'.

Contents

1 Introduction **1**
- 1.1 Theories of Morphology . . . 2
  - 1.1.1 Segmentational Morphology . . . 3
  - 1.1.2 Limitations of Segmentational Morphology . . . 6
  - 1.1.3 Word-Based Theories . . . 9
- 1.2 Morphology in Natural Language Processing . . . 16
- 1.3 Goals and Contributions of This Thesis . . . 20

2 A Review of Machine Learning Approaches to Morphology **23**
- 2.1 Recognition of Morph Boundaries . . . 24
- 2.2 Grouping Morphologically Related Words . . . 27
- 2.3 Predicting the Properties of Unknown Words . . . 29
- 2.4 Discussion . . . 31

3 Morphology as a System of String Transformations **33**
- 3.1 Formalization of Morphological Rules . . . 33
- 3.2 Finite State Automata and Transducers . . . 36
  - 3.2.1 Preliminaries . . . 37
  - 3.2.2 Compiling Morphological Rules to FSTs . . . 40
  - 3.2.3 Binary Disjunction . . . 42
  - 3.2.4 Computing the Number of Paths in an Acyclic FST . . . 45
  - 3.2.5 Learning Probabilistic Automata . . . 47
- 3.3 Rule Extraction . . . 51
  - 3.3.1 Finding Pairs of Similar Words . . . 51
  - 3.3.2 Extraction of Rules from Word Pairs . . . 54
  - 3.3.3 Filtering the Graph . . . 61
  - 3.3.4 Supervised and Restricted Variants . . . 61

4 Statistical Modeling and Inference **63**
- 4.1 Model Formulation . . . 63
  - 4.1.1 The Basic Model . . . 64
  - 4.1.2 Distributions on Subsets of a Set . . . 65
  - 4.1.3 Penalties on Tree Height . . . 65
  - 4.1.4 Part-of-Speech Tags . . . 66

4.1.5 Numeric Features . . . 67
4.2 Model Components . . . 68
4.2.1 Root Models . . . 69
4.2.2 Edge Models . . . 70
4.2.3 Tag Models . . . 71
4.2.4 Frequency Models . . . 72
4.2.5 Word Embedding Models . . . 73
4.3 Inference . . . 75
4.3.1 MCMC Methods and the Metropolis-Hastings Algorithm . . . 76
4.3.2 A MCMC Sampler for Morphology Graphs . . . 79
4.3.3 Sampling Negative Examples . . . 83
4.3.4 Fitting the Model Parameters . . . 84
4.3.5 Model Selection . . . 86
4.3.6 Finding Optimal Branchings . . . 87

5 Learning Inflectional Relations 89
5.1 Datasets . . . 90
5.2 Unsupervised Clustering of Inflected Forms . . . 91
5.3 Unsupervised Lemmatization . . . 95
5.4 Supervised Lemmatization . . . 96
5.5 Supervised Inflected Form Generation . . . 99

6 Semi-Supervised Learning of POS Tagging 103
6.1 A General Idea . . . 104
6.1.1 Intrinsic and Extrinsic Tag Guessing . . . 104
6.1.2 Applying Tagged Rules to Untagged Words . . . 105
6.2 The Method . . . 106
6.2.1 The Forward-Backward Algorithm for Trees . . . 108
6.2.2 Modifications to the Sampling Algorithm . . . 111
6.2.3 Extending an HMM with New Vocabulary . . . 114
6.3 Evaluation . . . 115
6.3.1 Experiment Setup . . . 115
6.3.2 Evaluation Measures . . . 116
6.3.3 Results . . . 117
6.3.4 Remarks . . . 120

7 Unsupervised Vocabulary Expansion 123
7.1 Related Work . . . 124
7.2 The Method . . . 124
7.2.1 Predicting Word Frequency . . . 125
7.2.2 Computing Word Costs from Edge Probabilities . . . 126
7.3 Evaluation . . . 128
7.3.1 Experiment Setup . . . 128
7.3.2 Results . . . 129

8 Conclusion **137**

Bibliography **141**

List of Figures

1.1 An example analysis of *relied* in two-level morphology. 17
1.2 A typical segmentational analysis of the German word *beziehungsunfähiger*. The information contained in the inner nodes (shown in gray) is lost in the segmentation. 18

3.1 A scheme for converting morphological rules into FSTs. 41
3.2 The transducer corresponding to rule (3.3). 41
3.3 The transducer corresponding to rule (3.9). 41
3.4 A comparison of running times for different disjunction strategies. . 44
3.5 The transducer S_1 for generating a deletion neighborhood. 52
3.6 The filter S_2 ensuring, that no more than the half of a word is deleted. 53
3.7 Matrices D, Ξ, X, Y and aligned sequences u, v after executing Algorithm 3.8 on the word pair (*trifft*, *getroffen*). The shaded cells correspond to the optimal alignment, which is extracted by Algorithm 3.9. 56
3.8 Rules extracted from the pair (*trifft*, *getroffen*) by Algorithm 3.10 and their corresponding masks, sorted by generality. 59

4.1 An example tree of word derivations. 64
4.2 A 'bad' tree caused by the lack of a constraint on tree height. . . . 66
4.3 NN architecture for the edge model. 71
4.4 NN architecture for the root tag model. 72
4.5 The difference in log-frequencies between words on the left and right side of the rule /Xen/ → /Xes/ in German. The dashed line is the best-fitting Gaussian density. 73
4.6 NN architecture for the root vector model. 74
4.7 NN architecture for the edge vector model. 75
4.8 The two variants of the 'flip' move. The deleted edges are dashed and the newly added edges dotted. The goal of the operation is to make possible adding an edge from v_1 to v_2 without creating a cycle. 80
4.9 The trajectory of the graph sampler expressed as the cost (negative log-likelihood) of the graph at a given iteration. The top plot shows all 10 million sampling iterations, while the bottom plot shows iterations from 4 million onward. The acceptance rate in this run was 0.12. 82

4.10 Convergence of MCEM fitting. The cost at the first iteration is far beyond the scale (around $1.6 \cdot 10^6$). 86
4.11 The number of rules and possible edges during model selection. . . 88

6.1 Two possible morphology graphs corresponding to the words *machen*, *mache*, *macht*. What does each of them tell us about the possible tags of those words according to (6.1)? 106
6.2 The Forward-Backward computation for a linear sequence in an HMM. $\alpha_{v_6,t} = P(v_1, \ldots, v_6, T_6 = t)$, whereas $\beta_{v_6,t} = P(v_7, v_8, v_9 | T_6 = t)$. 108
6.3 The Forward-Backward computation for a tree. Also here, $\alpha_{v_6,t} = P(v_1, \ldots, v_6, T_6 = t)$ and $\beta_{v_6,t} = P(v_7, v_8, v_9 | T_6 = t)$. 109
6.4 Adding or removing the edge (v, v', r). 112
6.5 Exchanging an edge to another with the same target node. The change can take place within one tree (a) or involve two separate trees (b). 113
6.6 In case of a 'flip' move, the smallest subtree containing all changes is the one rooted in v_3. The deleted edges are dashed, while the newly added edges are dotted. In order to obtain the new β_{v_3}, we recompute the backward probabilities in the whole subtree. α_{v_3} is not affected by the changes. (The node labels are consistent with the definition in Fig. 4.8.) 113
6.7 A special case of the 'flip' move, which changes the root of the tree. (a) – the tree before the move (the edge to delete is dashed, while the edge to be added is dotted); (b) – the tree after the move. The node labels agree with the ones in Fig. 4.8, with $v_4 = v_2$, $v_5 = v_1$ and v_3 not existing. The difference between the two variants of 'flip' is neutralized in this case. 113

7.1 Predicted log-frequency of the hypothetical word **jedsten*, as derived from *jedes* via the rule $/X\text{es}/ \rightarrow /X\text{sten}/$. The frequency model predicts the mean log-frequency $\mu = 4.24$ (corresponding to the frequency around 69) with standard deviation $\sigma = 0.457$. The probability of the word not occurring in the corpus (i.e. frequency < 1) corresponds to the area under the curve on the interval $(-\infty, 0)$, which is approximately 10^{-20}. 125
7.2 Predicted log-frequency of *aufwändigsten*, as derived from *aufwändiges* via the rule $/X\text{es}/ \rightarrow /X\text{sten}/$. The probability of log-frequency being negative (i.e. the frequency being < 1) is here around 0.863 ($\mu = -0.5, \sigma = 0.457$). 126

List of Tables

3.1 Commonly used semirings. $\oplus_{\log}$ is defined by: $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$. 38
3.2 Example rules extracted from the German Wikipedia. 60

5.1 Results of unsupervised clustering. 94
5.2 Results of unsupervised lemmatization. 97
5.3 Size (number of types) of the datasets used for supervised lemmatization. 97
5.4 Results of supervised lemmatization. 100
5.5 Results of supervised inflected form generation. 101

6.1 Different setups of the HMM tagger used in the tagging experiment. 116
6.2 Lexicon evaluation with coarse-grained tags. 118
6.3 Lexicon evaluation with fine-grained tags. 118
6.4 Tagging evaluation with coarse-grained tags. 119
6.5 Tagging evaluation with fine-grained tags. 120

7.1 The datasets used for evaluation. 129
7.2 Token-based OOV reduction rates for various numbers of generated words. 130
7.3 Type-based OOV reduction rates for various numbers of generated words. 131
7.4 Type-based OOV reduction as the percentage of the maximal reduction at a given size. 134

List of Algorithms

3.1 Binary disjunction. 43
3.2 Computing the number of paths in an acyclic DFST. 46
3.3 STOCHASTIC-MERGE. 48
3.4 STOCHASTIC-FOLD. 49
3.5 ALERGIA-TEST. 49
3.6 ALERGIA-COMPATIBLE. 50
3.7 ALERGIA. 50
3.8 Aligning similar words with minimum number of edit operations. . . 55
3.9 EXTRACT-ALIGNMENT. 55
3.10 Extracting rule candidates from pairs of similar words. 58

4.1 A single iteration of the Metropolis-Hastings algorithm. 79
4.2 Proposing the next branching in the sample. 81
4.3 Sampling negative examples of edges. 84

Acknowledgments

As is the case with every dissertation, also this one would not have been possible without support and good will of many people. First and foremost, I would like to thank my supervisors, Prof. Uwe Quasthoff and Prof. Gerhard Heyer. While they always supported me with advice and feedback, they also gave me a tremendous amount of freedom and trust in pursuing my own ideas. Furthermore, they secured the financial resources, including a three-year scholarship, which gave me a plenty of time to work on this topic. I also thank my colleagues, especially Dr. Ingmar Schuster, Max Bryan and Dr. Christoph Teichmann, for valuable discussions and introducing me to some topics, notably neural networks and Markov Chain Monte Carlo. I thank Dr. Giuseppe G. A. Celano for the consultation on Latin and Ancient Greek, as well as Christopher Schröder for proofreading. Finally, I thank all the colleagues that have worked in the office P818 over the years for contributing to a friendly and productive work environment.

This thesis was completed entirely using free software. The methods described here were implemented in Python. All the work was done under GNU/Linux systems and the text was typeset using LaTeX. I could not imagine efficient work without tools like the various tiling window managers or the Vim text editor. I owe much to the free software community. As to the more specialized software, I would like to thank the developers of OpenFST, HFST and Keras for their excellent work.

I also thank Michał Śliwiński, a teacher from my high school, for introducing me to the International Linguistics Olympiad and awakening my interest in theoretical and computational linguistics, which has shaped my whole professional career. Further thanks to the rock band Linie 7ieben, of which I was a member for the past 4 years, for providing me with pleasant distraction and a tempting career alternative.

The period of work on this thesis saw a wedding, as well as an abrupt breakup of the marriage two years later. I am deeply grateful to all the people who supported

me through 2018, which was the most difficult year of my life. I particularly want to mention my parents Anna and Andrzej, my brother Marek, and my friends from Leipzig: Agata Barcik, Basia Kubaińska and Cyprian Zajt. Needless to say, they were great company in better times as well.

Chapter 1

Introduction

> Every good work of software starts by
> scratching a developer's personal itch.
> Eric Raymond, *The Cathedral and the Bazaar*

This thesis is concerned with the topic of machine learning of language morphology. The focus lies on *unsupervised learning*, i.e. learning from raw, observable language data, without any additional labeling. This contrasts with *supervised learning*, i.e. learning from data labeled by human experts, as well as *rule-based approaches*, in which the language processing relies on manually constructed resources, like dictionaries and grammars. While the early history of Natural Language Processing (NLP) was dominated by rule-based approaches grounded in formal linguistic theories, the recent two or three decades have seen a nearly complete shift towards machine learning methods.

Despite their relatively low reliability (as compared to the two other approaches), unsupervised language learning methods have been a topic of active research for the entire history of NLP. Their roots lie in the linguistic structuralism of the early 20th century, which placed emphasis on deriving grammatical notions directly from observable language data. Concrete algorithms were presented as far back as 1950s [Harris 1955]. More recently, the rise of shared tasks like *Multilingual Parsing from Raw Text to Universal Dependencies* [Hajič and Zeman 2017] attests the increasing practical significance of unsupervised methods and high expectations placed on them by the research community.

In addition to the practical benefits of unsupervised learning, like rapid adaptation to new languages and domains, or the possibility of processing resource-scarce languages, for which human experts are unavailable, the research on unsupervised methods also poses interesting theoretical questions. An unsupervised

learning algorithm is expected to derive a knowledge representation from unannotated data, without any clue from a human annotator about what kind of representation is expected – apart from the clues contained in the design of the algorithm itself, in isolation from the data. Therefore, unsupervised learning requires an especially careful consideration of *what* should be learnt and *how* it can be derived from observable data. It thus stimulates the research on finding plausible representations of linguistic knowledge which are as near to electronically available, machine-readable language data as possible.[1] Furthermore, it might also shed light on the process of first language acquisition by humans, which is similar (although not entirely analogical) to unsupervised learning.

Correspondingly, the thesis begins by addressing the question of what representation should be learnt and how it relates to observable data. In Section 1.1, I review different approaches to morphology found in theoretical linguistics. Section 1.2 describes how those theories are applied to derive representations of morphology used in NLP and what kinds of morphological analysis are expected in practical applications. It turns out that the widely adopted segmentational view suffers from discrepancies: both between its strictly concatenative model of morphology and the reality of morphological phenomena, and between its objective and the use cases for morphological analysis in NLP. Those considerations lead to the motivation for the current thesis, which is adapting for the purposes of NLP an apparently underexplored and promising theory which takes a relational (rather than structural) view on morphology. This goal is elaborated in Sec. 1.3.

1.1 Theories of Morphology

Morphology is the part of language grammar concerned with individual words. As explained by an introductory text (emphasis original):

> In linguistics *morphology* refers to the mental system involved in **word** formation or to the branch of linguistics that deals with words, their internal structure and how they are formed.
>
> [Aronoff and Fudeman 2004, p. 2]

Morphology is usually divided into *inflection*, *derivation* and *compounding*, which are treated differently by many theories. However, it is difficult to formulate

[1] For example, the research on unsupervised parsing, as well as machine learning of parsing in general, recently contributed to the wide popularity of dependency grammar in the NLP community.

a clear definition distinguishing inflection from derivation. In a commonly used understanding, inflection is responsible for providing the word with features required by the syntactic context (like e.g. case or number), whereas derivation forms new words with new meanings. The term *compounding* is used for the process of creating a new word from more than one base word (like *lighthouse* ← *light* + *house*).

The above quote of Aronoff and Fudeman mentions two subjects: the internal *structure* of words and the *process* of word formation. A view shared by a large majority (but not all) of grammar theorists is that a theory of morphology is supposed to describe both of them and that the process of word formation is tightly related to the internal word structure. The latter is usually described in terms of *morphemes*, which constitute the basic building blocks for words. Some of the most influential theories based on this assumption are described in Sec. 1.1.1.

Although the assumption of internal word structure seems intuitive, there are certain phenomena in morphology which are very difficult to describe this way. Extending the basic theory to account for such cases results in a very abstract and vaguely defined notion of morpheme, which is only remotely related to the original idea. In Sec. 1.1.2, we look into details of such problems and how they are typically handled.

Finally, in Sec. 1.1.3, we turn to different theories of morphology, which make little or no assumptions about the internal structure of words and instead focus entirely on the process of word formation or structural similarities between words. Such theories provide the motivation and linguistic foundation for the current thesis.

1.1.1 Segmentational Morphology

The concept of a word being composed out of smaller, meaning-bearing units dates back to the ancient Indian grammar tradition, especially to Pāṇini's grammar of Sanskrit [Gillon 2007]. In contrast, the older European grammar tradition concentrated on the study of words and paradigms, treating words as basic units of meaning. The notion of *morpheme* as a minimal meaningful unit of language was popularized by the American structuralist school in the first half of the 20th century in the following formulation:

> A linguistic form which bears no partial phonetic-semantic resemblance to any other form, is a *simple* form or *morpheme*.
>
> [Bloomfield 1933, p. 161]

This definition is part of the *structuralist* approach to language. The structuralist school aimed to derive linguistic descriptions strictly from the structural properties of language. The most important criterion was the *distribution* of various elements of the language and its influence on the meaning. Contrary to older traditions, the structuralist school put emphasis on the spoken language, rather than written, as the subject of linguistic analysis [Bloomfield 1933, p. 21]. The structuralist approach to morphology was further elaborated in works such as [Nida 1949] and [Harris 1951].

Later interpretations of the above definition state that morphemes are minimal *signs* of the language. The notion of sign, introduced by de Saussure [1916], is essential in structuralist linguistics. A sign is an entity consisting of *form* and *meaning*, in which the connection between the two is arbitrary, i.e. one cannot be predicted from the other. Language is thus a system of signs, as every spoken utterance has a meaning, which is related to its phonetic form only by means of the convention of a particular language.

The usefulness of morphemes as means of predicting the relationship between form and meaning of words can be illustrated by the following example:

Example 1.1. Finnish

	lentokoneissanikin				
lento	kone	i	ssa	ni	kin
flight	machine	PL	INESS	1SG	too

'in my aeroplanes, too'

It is not very likely for a Finnish speaker to ever have heard exactly this word.[2] After all, hardly anybody possesses multiple aeroplanes. However, understanding it will not pose any problems to the speaker, because it contains parts known from many other words, like *-i-* (plural), *-ssa* 'in', *-ni* 'my' (possessive suffix) and *-kin* 'too'. Even the word *lentokone* 'aeroplane', which is probably known to all Finnish speakers, is composed in a transparent way, so that one could understand it without having heard it before.

It can be seen from Example 1.1 that Finnish morphology is (mostly) *agglutinative*: each unit of meaning or grammatical function corresponds to one morpheme. The morphemes are concatenated to form words. On the other hand, we speak of *fusional* morphology if multiple grammatical functions are fused in a single morpheme. For example, the Polish equivalent of 'in planes' is *w samolotach*,

[2] A Web search with DuckDuckGo finds no results for this word. (accessed on Jan 16, 2019)

where the suffix *-ach* contains both 'plural' and 'locative' features, which cannot be isolated. Languages with fusional morphology typically display fewer morphs per word, more homophony of functional morphemes (called *syncretism*) and more non-concatenative phenomena.

The two fundamental models of morphology that emerged in the mid-20th century were the *Item-and-Arrangement* (IA) and the *Item-and-Process* (IP) model [Hockett 1954]. Especially the former is tightly related to the notion of morpheme: words (and ultimately, all linguistic constructions) are arrangements of morphemes. The *arrangement* encompasses more than just the linear ordering: also a kind of constituent structure is postulated. Morphology in this formulation is thus somewhat similar to syntax, but operating below the word level. This analogy has been explored in more detail by, among others, Selkirk [1982].

In a simplified view, the IA model could be characterized as follows:

$$w = \mu_1 \mu_2 \dots \mu_k \tag{1.1}$$

In this formulation, a word w is simply a sequence of *morphemes* μ_i. The morphemes are realized by strings of phonemes called *morphs* and the concatenation operation is realized by grammar-wide *morphophonological rules*, which can change the shape of morphs depending on their context. In order to account for the hierarchical structure, one could additionally introduce a bracketing on the sequence of morphemes. Note that, in general, morphs are not allowed to contain any additional phonological information besides the string of phonemes (e.g. 'alternation markers').

The IA model is a static description: it concentrates on identifying and describing the *internal structure* of words. The regular similarity in form and meaning of words can then be explained by the common elements of the internal structure.

On the other hand, the Item-and-Process model describes the structure of words in dynamic terms: words are *derived* from other words (or from more abstract lexical items, like *stems* or *roots*) by means of *processes*. This model may, but does not have to, involve the notion of morpheme. However, there is a widespread agreement that a process typically encompasses adding some phonological material, called *process markers*. The naming of some processes, like *affixation* (i.e. the addition of an affix) indicates though, that morphemes are implicitly present in most IP descriptions as well.

A schematic formulation of the IP model, analogously to (1.1), would thus

be the following:

$$w = \pi_k(\pi_{k-1}(\dots \pi_1(r) \dots)) \tag{1.2}$$

A word w is derived from a root r through a series of processes π_i, each of which takes the output of the previous process as its input. In contrast to the IA model, in which everything was a morpheme, the roots and the processes in the IP model are fundamentally different entities.

The main problem with the IP model is that it is very difficult to define what exactly a process may and may not do. Allowing arbitrary rewrite rules is clearly too strong, while restricting the processes to addition of phonological material is clearly too weak. Besides, it is not clear what exactly is the input of a process. In addition to complete existing words, 'incomplete words' (stems) are normally allowed and it is difficult to define such entities precisely, as well as prove that language speakers possess an internal representation of them.

1.1.2 Limitations of Segmentational Morphology

While the intuition that stands behind the notion of morpheme is clearly seen in a case like Example 1.1, there are many kinds of similarities between words that cannot be easily analyzed using morphemes. This usually leads to workarounds and extensions to the basic theory: in case of the IA model, either the morphs are allowed to contain abstract, transformation-related information (which, as [Hockett 1954, p. 224] points out, undermines the whole idea of Item-and-Arrangement), or too much burden is put on morphophonological rules, which are expected to 'fix' the discrepancy between the model and the reality. In case of the IP model, the flexible notion of 'process' is used to encompass more and more sophisticated transformations. In both cases, the result is a, in my opinion, vague theory with unclearly defined concepts.

In the following, I present a few fairly common phenomena, which pose problems for segmentational morphology, as well as for NLP applications built upon a segmentational theory of morphology.

Stem alternation. Stem alternation is probably the most common non-concatenative phenomenon in morphology: a certain sound inside the stem (often the stressed vowel) undergoes a regular transformation. Example 1.2 shows German

plural formation which exhibits the well-known *umlaut*.

Example 1.2. German

a.	Haus	/haʊ̯s/	'house.NOM.SG'
	Häuser	/hɔɤ̯zɐ/	'house.NOM.PL'
b.	Buch	/buːx/	'book.NOM.SG'
	Bücher	/byːçɐ/	'book.NOM.PL'
c.	Vater	/fɑːtɐ/	'father.NOM.SG'
	Väter	/fɛːtɐ/	'father.NOM.PL'

Sound changes at morph boundaries. Some affixes trigger sound changes affecting the nearby phonemes. It is unclear whether the resulting alternation should be modeled as part of the affix or as stem alternation. In Example 1.3, *-a* is the nominative suffix of many Polish feminine nouns. The locative suffix is something like *-ʲe*, where the first element triggers a regular consonant change. (Note that it is not even exactly a palatalization marker in the phonological sense, since the sequences /kʲe/ and /gʲe/ are possible in Polish.) It is difficult to mark morph boundaries here, as, strictly speaking, a part of the stem and a part of the suffix melt into a single phoneme.

Example 1.3. Polish

a.	droga	/drɔga/	'way-NOM.SG'
	drodze	/drɔd͡zɛ/	'way-LOC.SG'
b.	ręka	/rɛŋka/	'hand-NOM.SG'
	ręce	/rɛnt͡sɛ/	'hand-LOC.SG'
c.	strata	/strata/	'loss-NOM.SG'
	stracie	/strat͡ɕe/	'loss-LOC.SG'
d.	woda	/vɔda/	'water-NOM.SG'
	wodzie	/vɔd͡ʑe/	'water-LOC.SG'
e.	zupa	/zupa/	'soup-NOM.SG'
	zupie	/zupʲe/	'soup-LOC.SG'
f.	żaba	/ʒaba/	'frog-NOM.SG'
	żabie	/ʒabʲe/	'frog-LOC.SG'

Zero morphs. It is often convenient to tie some grammatical categories, like e.g. plural, to a particular morpheme that occurs in the respective form. In Example 1.4b, the 'plural' feature is associated with the suffix *-s*, while the stem *house* does not carry any number feature. In case of 1.4a, the word consists of only

one morpheme, which is supposed to be the same stem as in 1.4b. However, the word also contains the feature 'singular', which would have to be located in this only morpheme. There are two possible conclusions for the segmentational analysis: either the stems in 1.4a and 1.4b are homophonic, but distinct morphemes, one with the singular feature and one without (which seems absurd), or the widely accepted statement that the word from 1.4a contains another morpheme: a 'singular suffix' with null phonological realization.[3] Many analyses tend to the latter option, thus allowing phonologically null morphs.

Example 1.4. English

a. house 'house.SG'
b. house-s 'house-PL'

Cranberry morphs. The term 'cranberry morph' was coined by Aronoff [1976]. It describes a morph that occurs only in one word, like the first part of words in Example 1.5a. The rationale behind isolating such morphs is dubious, because no well-defined meaning can be attributed to them – their meaning is inseparably tied to the meaning of the word they occur in. For example, the only statement that can be made about the meaning of *cran-* is that it turns a generic *berry* into a *cranberry*. However, *berry* is clearly a morph, as can be seen from the words listed in 1.5b.

Example 1.5. English

a. cranberry boysenberry huckleberry
b. strawberry blueberry blackberry gooseberry

[Aronoff 1976, p. 10]

Case studies like that of Rainer [2003], who analyzes in detail the Spanish derivational suffix *-azo*, show that attributing meanings to derivational affixes is difficult as well. Instead, he suggests to treat meaning as a property of word. The meaning of newly coined words is attributed by analogy to one or more already existing words derived with the same affix. However, the general meaning of an affix does not have to be specified and once the words are coined, their meaning may evolve in an idiosyncratic way.

[3] In fact, there are more possible explanations: for example that the plural feature overrides the singular. While it seems to be a plausible explanation of the example, it is very difficult to convert it into a general principle: what is overridden by what, why and when? Is the singular feature part of the stem or is it assigned as default if no number is explicitly marked? As each of such explanations raises further questions, the notion of 'zero morph' is widespread.

Words of foreign origin. A frequent source of borderline cases for segmentation are borrowings, in which some morphological structure of the source language is still visible. This is well exemplified by words of Greek and Latin origin in many European languages, especially English. Example 1.6a shows words that must have been borrowed into English already as compounds, as their first part is not a proper English morpheme. 1.6b lists words that correspond to valid Latin compounds, but might be English word formations as well. Finally, the words listed in 1.6c are clearly English formations, demonstrating that the suffix *-cide* is productive in English.[4] Any consistent segmentation of all three groups is going to be disputable.

Example 1.6. English

a.	—	deicide
	—	regicide
	—	suicide
b.	bacterium	bactericide
	herb	herbicide
	infant	infanticide
c.	computer	computercide
	robot	roboticide
	squirrel	squirrelcide
	weed	weedicide

The examples presented in this section are only a modest selection of 'problematic' phenomena. In my opinion, those pose the most problems for automatic processing, at least of European languages, which are the focus of this thesis. Further such phenomena include e.g. subtraction, reduplication, transfixation or morphologically conditioned stress and tone alternations. They are covered in detail in linguistic literature, especially introductory works like [Aronoff and Fudeman 2004] or monographies critical of the notion of morpheme reviewed in the next section.

1.1.3 Word-Based Theories

The problems with the notion of morpheme are well-known in theoretical linguistics. To my knowledge, the first thorough critique of morpheme as a 'minimal meaningful unit', combined with a theory of morphology based on rewrite rules,

[4] For references to sources attesting the words *computercide*, *roboticide* and *squirrelcide*, see the Wiktionary pages for those words (accessed on Jan 11, 2019).

was proposed by Aronoff [1976]. It is rooted in the tradition of Generative Grammar and builds upon the treatment of morphology of, among others, Chomsky [1970] and Halle [1973].

After reviewing problems with the notion of morpheme as a minimal sign, especially 'cranberry morphs', Aronoff postulates a *word-based* theory of morphology by formulating a very important principle, which will apply to other theories as well:

> Hypothesis: All regular word-formation processes are word-based. A new word is formed by applying a regular rule to a single already existing word. [...]
>
> [Aronoff 1976, p. 21]

Thus, rules are always applied to *already existing words*. However, Aronoff's theory is only concerned with word formation (i.e. derivation), as it views inflection to be a part of syntax. Correspondingly, what is referred to as 'word' is in fact 'word minus inflectional markers'. Moreover, despite the criticism, Aronoff upholds the concept of morpheme as motivated, albeit not essential for a theory of morphology and under a different, extremely vague, definition:

> A morpheme is a phonetic string which can be connected to a linguistic entity outside that string.
>
> [Aronoff 1976, p. 15]

Aronoff [2007] reiterates his criticism of the decomposition of morphologically complex words and provides arguments showing that complex words are usually lexicalized and their meaning cannot be predicted from their postulated structure.

Another theory rejecting the notion of morpheme which is rooted in Generative Grammar is *A-morphous Morphology* [Anderson 1992]. It views lexicon as consisting of *stems*, which are again defined as 'word minus inflectional material'. Inflection and derivation are treated as fundamentally different phenomena: while inflection is thought of as part of syntax, derivation operates inside the lexicon. Those phenomena are realized by two kinds of rewrite rules: *Word Formation Rules* generate words (inflected forms) from stems and constraints on morphosyntactic features provided by the syntactic structure, whereas *Stem Formation Rules* generate stems from other stems and exchange no information with the syntax. The notion of morpheme is explicitly rejected. Moreover, it is emphasized that both

kinds of rules create *no internal word structure* which would be available to other parts of grammar. However, in the analysis of compounding, Anderson makes some concessions towards segmentational morphology (which are reviewed and criticized in detail by Starosta [2003]).

The notion of morpheme and internal word structure continues to be a topic of debate in more recent research. Hay and Baayen [2005] provide a comprehensive overview of the discussion. They take an interesting intermediary position: while internal word structure is motivated, it is best seen as a gradient (i.e. fuzzy) structure, rather than discrete. This in turn fits into the larger-scale idea of incorporating the notions of probability, uncertainty and fuzziness into grammar theory, which is comprehensively presented by [Bod et al. 2003].

Hudson [1984] proposes a theory named *Word Grammar*, which accounts for all major levels of linguistic representation (i.e. phonology, morphology, syntax and semantics). Although it takes a concatenative, Item-and-Arrangement view on morphology, it is interesting here for a different reason. Firstly, it displays an approach that came to be called *pan-lexicalism*: all information relevant for producing valid utterances of the language is stored in the lexicon. There is no ontological distinction between *lexical entries* and *rules*: rules (e.g. of syntax) are also modeled as lexical entries. Secondly, the lexicon (together with all the grammatical information) is modeled as a network (i.e. graph), which, according to Hudson, is part of an even larger 'cognitive network'. Both ideas – pan-lexicalism and the representation of lexicon as a graph – are fundamental to the theories of morphology presented further below and to the model of morphology adopted by this thesis.

Whole Word Morphology mentioned in the title of this thesis is a theory proposed by Ford et al. [1997]. It regards *words* as minimal meaning-bearing units of language. By words, it means whole, inflected word forms just as they are uttered in the language, rather than abstract lexical entities (like 'word without inflection'). No distinction is made between inflection and derivation.

Systematic similarities in form and meaning between pairs of words are expressed by rules (*Morphological Strategies* in WWM's original formulation) of the form:

$$/X/_{\alpha} \leftrightarrow /X'/_{\beta} \tag{1.3}$$

In the above representation, X and X' are both words whereas α and β are mor-

phosyntactic categories ('part-of-speech tags' in the language of NLP). The bidirectional arrow expresses that if a word fitting to the pattern on the left side of the rule exists, then the existence of a counterpart matching the right side is postulated, *and vice versa*. The rule has no privileged direction and none of the words is said to be 'derived' from the other or morphologically 'more complex' than the other.

An example rule reflecting the relationship between French masculine and feminine nouns like (*chanteur*, *chanteuse*) is expressed as follows:[5]

$$/X\text{œr}/_{\text{N.MASC}} \leftrightarrow /X\text{øz}/_{\text{N.FEM}} \tag{1.4}$$

Thus, WWM provides a different answer to the fundamental question 'what is morphology?':

> Morphology is the study of formal **relationships** amongst words.
>
> [Ford et al. 1997, p. 1]

The internal structure of words is not a subject of study here.

A theory very similar to WWM was proposed as part of a larger theory of dependency grammar called *Lexicase* [Starosta 1988]. The similarity was noted by the authors of both theories, which resulted in a jointly edited volume [Singh and Starosta 2003], in which the common theory was renamed to *Seamless Morphology*. Starosta frequently cites Hudson and pan-lexicalism as an inspiration for his theory, which can also be seen as pan-lexicalist. In Lexicase syntax, there are no rules of syntax separate from the lexicon. Instead, the information about how words can be combined to form utterances is contained in the lexical entries: each word contains a frame of possible dependents (in the sense of dependency grammar).[6] Rules like 'every noun can have an adjective dependent on its left' emerge only as a method of 'compression': to store redundant information efficiently, so that it does not have to be memorized explicitly in every entry. The rules are thus not necessary for the functioning of the grammar and what exactly is covered by rules might be left unclear. Starosta argues that the pan-lexicalist formulation is a more accurate model of human language processing than the traditional generative grammar:

[5] The presentation given here follows exactly the one of [Ford et al. 1997]. In my opinion, it is slightly misleading: X seems to have different meanings in (1.3) and in (1.4). Nevertheless, the intended meaning is apprehensible, at least from the example. My own version of the definition of rule is given in Sec. 3.1.

[6] This can be seen as an extreme counterpart of generative Phrase Structure Grammar, in which the lexicon is contained in the grammar in form of rules generating terminals from non-terminals.

> [Psychological experiments] indicate that memory is far more important in actual speech processing than most generative grammarians had hitherto believed possible and the lexicon bears a much greater burden in this respect than the rules which many of us have dedicated our professional lives to constructing.
>
> [Starosta 1988, p. 41]

The same approach is applied to morphology: although every word could be stored in the lexicon as a separate entry, it is more efficient to capture regularities in form of rules, which generate words from other words. The rules are formulated very similarly to those in WWM.

Compounding

Although compounding might seem to be the most articulate example of the concatenative character of morphology, the authors of WWM make important points there as well. An analysis of compounding as putting two words together to form at third one, like (1.5), is rejected:

$$/X/ + /Y/ \leftrightarrow /XY/ \tag{1.5}$$

Instead, the authors note that compounds come in clusters, in which at least one of the parts is known from multiple compounds:

> Facts of 'compounding' from various languages indicate quite clearly that 'compounds' come in sets, each set anchored in some specific constant.
>
> [Singh and Dasgupta 2003, p. 78]

A word that is not generally known as 'compound-forming' is unlikely to form a compound, even if there seems to be no reason to prevent it. This phenomenon

can be illustrated by the following example:[7]

Example 1.7. English

a.	oldtown	*oldcity
b.	downtown	*downcity
c.	boomtown	*boomcity
d.	Chinatown	*Chinacity
e.	Koreatown	*Koreacity

Although *town* and *city* are generally synonyms (certainly in this context), only *town* is allowed in the listed compounds. This is difficult to explain using a general model of compounding like (1.5). The analyses presented by both Singh and Dasgupta [2003] and Starosta [2003] would instead postulate that those words are related to a *single* word through a rule like:

$$/X/_{\text{N/Adj}} \leftrightarrow /X\text{town}/_{\text{N}} \tag{1.6}$$

-town is thus a morphological constant, or what would have been called an 'affix' in other theories. The productivity of compounding with *-town*, together with the impossibility of replacing it with *-city*, can be explained with the assumption that no similar rule for *-city* is active in the morphology.

There remains an open question: how to account for the evident relatedness between the word *town* and the rule (1.6). As neither of the two articles cited above provides an explicit answer to this question, I propose my own extension to WWM: *second-order rules*. Those are rules relating a *word* to a *rule*. The general formulation of the second-order rule responsible for the pair (*town*, $/X/_{\text{N/Adj}} \leftrightarrow /X\text{town}/_{\text{N}}$) would then be:

$$/Y/_{\text{N}} \leftrightarrow (/X/_{\text{N/Adj}} \leftrightarrow /XY/_{\text{N}}) \tag{1.7}$$

Semantically, this forms a hyponym or meronym of Y which is additionally specified by X. Some further instances of this pattern include (*berry*, $/X/_{\text{N/Adj}} \leftrightarrow /X\text{berry}/_{\text{N}}$) (*blackberry*, *gooseberry*) or (*light*, $/X/_{\text{N/Adj}} \leftrightarrow /X\text{light}/_{\text{N}}$) (*gaslight*, *backlight*).

The difference between (1.5) and (1.7) is that (1.7) divides the compounding process into two stages: deriving an 'affixing rule' from one of the parts and then applying this rule to the second part. If no rule arises, no compounds are produced.

[7] This particular example is my invention, but it follows the argumentation of Singh and Dasgupta [2003].

If a rule is created, it typically becomes productive. This accounts for the 'many or none' character of compounds.

Mathematical associations

Graphs. The relational model of morphology provided by WWM is naturally expressed in terms of graph theory: words are vertices and morphologically related words are linked by an edge. This fits well into the wider tendency of modeling language structures using graphs, which is most commonly displayed in dependency grammar. Hudson [1984] even modeled the complete language grammar, from phonology all the way up to semantics, using graphs (albeit adopting a concatenative model of morphology). From the point of view of mathematics and computer science, graphs are well-understood data structures. This greatly simplifies the formalization of WWM and developing algorithms for it. The research on graph-based NLP offers some well-developed generic methods (like Chinese Whispers clustering) [Biemann 2007a,b] which can be transferred to morphology thanks to adopting a theory like WWM.

Pan-lexicalism and Minimum Description Length. The idea of 'pan-lexicalism', prominently displayed by Starosta [1988], but attributed by him to Hudson [1984], shows a highly interesting parallel to the mathematical and philosophical idea of Minimum Description Length [Rissanen 1978, 2005; Grünwald 2007], which in turn has been used as a theoretical foundation for several unsupervised learning algorithms reviewed in Chap. 2. According to the pan-lexicalists, all information needed to form valid utterances of the language is stored in the lexicon. This includes especially syntax, expressed as constraints on dependency relations of a word. The 'grammar', rather than being a component additional and external to the lexicon, is merely a method of compressing the information stored in the lexicon by capturing regularities and eliminating redundancies. In the language of information theory, the lexicon is the 'data' and the grammar is the 'code' used to encode the data efficiently. The MDL principle provides a formal criterion for selecting 'good' codes by analyzing the tradeoff between the length of the encoded data and the complexity of the code. The capability of the grammar to predict new data arises naturally from its compression capability.

This view on the relationship between lexicon and grammar also explains the uncertainty about which information is stored explicitly in the lexicon and which generated by rules. The traditional assumption that the lexicon is minimal and the

grammar captures *all* regularities seems not to be psychological reality, as noted by the Starosta quote above. In the pan-lexicalist formulation, the answer to this question is less categorical: all information is stored in the lexicon, but different speakers might use different near-optimal 'codes' (i.e. grammars) in order to encode their lexicon efficiently. Such codes typically do not capture *all* redundancies. The single optimal grammar (in the MDL sense) is probably not used by anyone and might even not exist.[8]

With respect to morphology, this reasoning leads to a lexicon being a list of existing words (known to a certain speaker or occurring in a certain corpus) and a grammar being a 'code' for encoding such lists with little redundancy. This is the underlying concept of the probabilistic model presented in Chapter 4. I believe that the combination of pan-lexicalism and MDL is a fascinating topic for further research as a solid theoretical foundation for unsupervised language learning algorithms. The possibilities provided by this approach are barely scratched on the surface with the current thesis.

1.2 Morphology in Natural Language Processing

The morphological analysis in Natural Language Processing almost always follows the Item-and-Arrangement model. Arguably the most important reason for this fact is that such formulation provides a clear task definition: segmenting a word into some smaller units, which are thought to be well defined and understood. Part of the explanation might also be that computational approaches to morphology were developed especially for the goal of processing highly agglutinating languages, like Finnish or Turkish, which seem to fit the idealized Item-and-Arrangement model very well. On the other hand, in the processing of English, or even languages with a more sophisticated fusional morphology, like German or French, morphology is often not taken into account, because the number of possible forms is manageable

[8]More specifically, a notion like '*the* optimal grammar of English' might be ill-defined, as every speaker has their own lexicon, corresponding to words and utterances that they know, with frequency information reflecting the sample of the language that they have encountered. The grammar learnt by an individual speaker reflects the compression they perform to manage the large amount of information contained in the language sample known to them. The information-theoretic optimality of the grammar is thus local to the representation of the language as known by a certain speaker, rather than a global property of the language. Of course, as the knowledge of large portions of language data is shared among many speakers, so are many grammatical generalizations, so it is perfectly reasonable to speak of 'the grammar of English' reflecting the widely shared language competence. However, such grammar is not expected to be optimal to any single speaker in the information-theoretic sense.

Lexical representation:	r	e	l	y	+	e	d
Surface representation:	r	e	l	i	ɛ	e	d

Figure 1.1: An example analysis of *relied* in two-level morphology.

even if every word form is treated separately. In processing Germanic languages, like German or Dutch, the only part of morphology that cannot be tackled by listing all forms is compounding, which is perfectly concatenative, so most effort is usually invested to address this phenomenon.

The segmentational morphological analysis fits well into a more general framework of NLP: segmenting the text into structural units on various levels (syllables, morphemes, words, phrases, sentences) and labeling the units with some additional information (tagging). Furthermore, the distribution of each structural unit in the context can be analyzed. This kind of analysis is highly inspired by the structuralist school of linguistics. Bordag and Heyer [2007] presented a formalized framework, in which various linguistic notions are related either by means of *concatenation* or *abstraction*. In this formulation, morphemes constitute a layer between phonemes/letters and word forms.

Two-level morphology. Computational linguists have noticed early that, even in an agglutinative language like Finnish, morphological phenomena do not strictly follow the idealized concatenative model. Especially phonologically or orthographically conditioned alternations at morpheme boundaries are a frequent phenomenon that had to be accounted for by a computational model. The solution that has set the standards for a long time is *two-level morphology* [Koskenniemi 1983; Karttunen et al. 1992; Beesley and Karttunen 2003]. It views words on two levels of representation: the *surface level*, which is exactly how the word is spelled, and the *lexical level*, at which the word is represented as a concatenation of morphs (Fig. 1.1). The mapping between the two levels is usually more complicated than the obvious deletion of morph boundary symbols. It is described by a special kind of rules, which can be compiled into a finite-state transducer. Examples of mature, open-source two-level morphological analyzers include Omorfi for Finnish [Pirinen 2015], Morphisto for German [Zielinski and Simon 2008] and TRMorph for Turkish [Çöltekin 2010].

Although two-level morphology was designed specifically with agglutinating morphology in mind, attempts have also been made to extend it to non-concatenative morphology. In particular, Kiraz [2001] adapted the formalism to the Semitic root-and-pattern morphology by using transducers with multiple tapes.

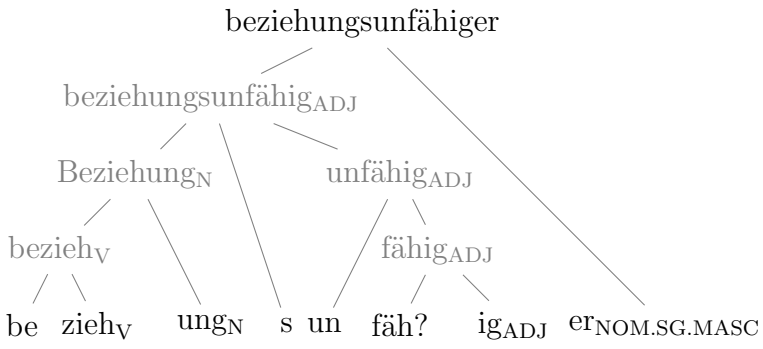

Figure 1.2: A typical segmentational analysis of the German word *beziehungsunfähiger*. The information contained in the inner nodes (shown in gray) is lost in the segmentation.

Figure 1.2 shows an example of morphological analysis based on a segmentational theory. The German word *beziehungsunfähiger* is an inflected form of the adjective *beziehungsunfähig* 'incapable of relationships'. This in turn is a compound of *Beziehung* 'relationship' and *unfähig* 'incapable'. The former is derived from the verb stem *bezieh-* 'to relate', which is a concatenation of the root *zieh-* 'to pull, drag' and a derivational prefix *be-*, the meaning of which is obscure. The adjective *unfähig* is derived from *fähig* 'capable' with the negative prefix *un-*. The analysis of *fähig* is disputable: it clearly contains a common adjectival derivational suffix *-ig*, but the corresponding root, **fäh-*, is missing from modern German.[9] This is another example of a 'cranberry morph'. In the NLP context, such cases, which are not as rare as they might seem, require arbitrary decisions by human annotators. Those influence noticeably the reference datasets and evaluation results of automatic segmentation algorithms.

A complete morphological analysis of the word *beziehungsunfähiger* consists of the whole tree depicted in Fig. 1.2. However, the output of an automatic morphological analyzer (e.g. based on two-level morphology) is most often going to be the sequence of leaf nodes of the tree: morphs labeled with some grammatical symbols. It is very difficult to reconstruct any meaning out of such sequence: *zieh-* means 'to pull', the meaning of *be-* is unspecified, *un-* is negation, but it is not clear what is negated. The inner nodes of the tree (shown in gray) contain information on relationships between the analyzed word and other existing words. This infor-

[9]Diachronically, it was derived from the verb whose modern form is *fangen* 'to catch'. Originally being similar to the paradigm *gehen-ging-gegangen*, the paradigm of *fangen* was rebuilt based on past forms [Kluge 1989].

mation is crucial for the proper understanding of the analyzed word. However, it is lost if the task of morphological analysis is defined as segmentation into morphs.[10]

In contrast to the focus of morphological research, which is mostly on segmentation, other areas of NLP generally view words as minimal units of language and are hardly ever interested in any information below the level of word forms. This is especially true for statistical language models nowadays employed throughout the field of NLP, from POS tagging and parsing [Manning and Schütze 1999; Kübler et al. 2009], through machine translation and speech recognition [Koehn 2010; Jelinek 1997], to topic modeling and text mining [Feldman and Sanger 2007; Blei 2012]. In result, most tools developed for morphological analysis are difficult to plug into larger pipelines and make use of.

From an application point of view, morphology is often understood in a very reduced way: for example as compound splitting or lemmatization and inflectional analysis. Experiments with modeling linguistically motivated subword units, like [Creutz et al. 2007; Virpioja et al. 2007; Botha and Blunsom 2014], are rare, and even such models often ultimately aim at predicting the occurrence of *words*. The inclusion of morphology in a task like statistical machine translation usually has the character of pre- or post-processing and amounts to inflectional analysis and/or compound splitting [Nießen and Ney 2000; Amtrup 2005; Popović et al. 2006; Dyer 2007]. Attempts to include productive derivational morphology in the lexicon component, like [Cartoni 2009], are very rare.

Turning back to Figure 1.2, the most important information is contained in the top part of the tree, i.e. in the branching of the root node into the lemma and the inflectional suffix, and then the splitting of the compound. Further branchings become less and less relevant the deeper we descend in the tree. Arguably, this prioritization should be reflected in the design of computational approaches to morphology, as well as their evaluation metrics.

Spoken vs. written language. While discussing theoretical linguistic foundations for natural language processing, it is important to keep in mind a significant

[10] Analyzers based on two-level morphology sometimes cope with this problem by including complex stems into their lexica. For example, Morphisto [Zielinski and Simon 2008] lists multiple segmentation alternatives for *beziehungsunfähiger*, some of them containing words like *Beziehung* or *unfähig* as single lexicon items, and some segmenting them further into morphs. This approach helps retain relationships between words, but it introduces new problems: it leads to a combinatorial explosion of possible analyses and the segmentation items are in general no longer morphs.

difference between the two disciplines: theoretical linguistics aims to describe the language as it is spoken, while NLP methods almost always work with written representations. This leads to another discrepancy: the annotation of gold standards and training data often tries to follow guidelines based on linguistic theory, which originally describe the spoken form, but the structures have to be reflected in the written form – in case of machine learning approaches, often by marking morph boundaries directly in the surface forms of words.

For example, it is a common case that simple concatenative phenomena appear non-concatenative in writing. For example, the English pairs (*embed*, *embed**ded***) or (*sit*, *sit**ting***) exhibit 'gemination', while pairs like English (*rel**y***, *rel**ie**s*) or Dutch (*l**e**zen*, *l**ee**s*) 'vowel alternation', that are solely artifacts of orthography. Thus, because we are dealing with written language, non-concatenative morphology might be an even more common and significant issue than the traditional grammar would predict. Furthermore, in cases like Example 1.3, the written forms motivate a different segmentation than the spoken forms (Locative suffix *-ie* in most cases) due to phonological issues. It is unclear which segmentation should be considered correct (e.g. *zup-ie* or *zupi-e*). Even in a case like English (*house*, *houses*), the spelling suggests the segmentation *house-s*, while the pronunciation (/haʊ̯s/, /haʊ̯zɪz/) would rather speak for *hous-es*.

Regarding evaluation and gold standard annotation, this problem is addressed by metrics such as EMMA [Spiegler and Monson 2010], which do not evaluate the segmentation of the surface form, but rather the sequence of morphs attributed to the word, regardless of their labeling.

1.3 Goals and Contributions of This Thesis

The primary goal of this thesis is the application of a word-based theory of morphology (inspired mostly by Ford et al.'s Whole Word Morphology) in the field of Natural Language Processing, with emphasis on unsupervised learning of morphology.

In Chapter 3, I formalize the string transformations constituting morphological rules in WWM and integrate them into the Finite State Transducer calculus. Furthermore, I propose efficient algorithms for discovering morphological relations in raw lists of words. In Chapter 4, I present a generative probabilistic model for trees of word derivations. This is, to my knowledge, the first probabilistic model

explicitly mentioning Whole Word Morphology as a linguistic foundation.[11] The model is formulated as a framework with several replaceable components. In addition to the model, I propose an inference method based on Markov Chain Monte Carlo sampling of branchings (directed spanning trees) of the full morphology graph. The model is not designed with a particular task in mind (like e.g. lemmatization). Instead, it is a general model of 'morphological competence' in the WWM formulation. It can be applied to various tasks and trained under various degrees of supervision. Its minimal input data are a list of words, but in case further information is available, like e.g. POS tags or word embeddings, it can be optionally used.

As I started working on this thesis, the state-of-the-art in machine learning of morphology was dominated by segmentation. Therefore, one of the goals was to look for alternative tasks for evaluating morphology learning, which would be suitable for segmentational and non-segmentational approaches alike and ideally be designed with concrete applications in mind. This situation has changed in the recent years: tasks like vocabulary expansion or learning tree representations are now present in mainstream literature independently of my modest effort. Nevertheless, in Chapters 5, 6 and 7, I propose various methods of evaluating 'morphological competence' without referring to the internal structure of words. In Chap. 5, I describe experiments related to the learning of relational descriptions of inflection, like lemmatization or clustering of words belonging to the same lemma. Chapter 6 describes a method for guessing POS tags of words unseen in a corpus and using it to improve part-of-speech tagging. Finally, Chap. 7 evaluates the model on the task of vocabulary expansion.

Non-goals. It should be emphasized that, despite the somewhat polemic presentation in Sec. 1.1, the goal of this thesis is not to argue for the 'right' theory of morphology. The choice of WWM is dictated by the fact that it has received little attention, while seemingly being able to solve some well-known problems of the more common approaches. In addition, the minimalism of WWM in postulating structures and entities and the strict, uncompromising approach of describing *only* the directly observable, fits well to the unsupervised learning paradigm.

The goal of the experiments in Chapters 5-7 is not to solve every single task

[11] The only computational experiment based on WWM that I am aware of is Neuvel and Fulop [2002], which consists of only rule discovery. However, an increasing number of recent papers (like e.g. Luo et al. [2017]) adopt a similar view on morphology without mentioning any linguistic theory explicitly.

with the highest possible accuracy, but rather to assess how well they can be solved by a unified, whole-word-based model of 'morphological competence'. Models specialized on a single task, like supervised lemmatization, might well achieve higher evaluation scores, but are less interesting from a wider theoretical point of view.

Own previous work. A part of the material included in this thesis has been published as conference papers [Janicki 2015; Sumalvico 2017]. The basic idea of basing the unsupervised learning of morphology on Whole Word Morphology was pursued already in my MSc thesis [Janicki 2013, 2014].

Chapter 2

A Review of Machine Learning Approaches to Morphology

In this chapter, I provide a brief review of the literature on machine learning of morphology. 'Machine learning' is understood here in a broad sense, meaning all methods that learn a description from data. This includes both the typical machine learning methods, like classification or probabilistic models, and approaches based on problem-specific heuristics. Because the present thesis is mainly concerned with the development of unsupervised learning methods, so will also be the focus of the presentation in this chapter. However, many unsupervised approaches can easily be reused in a supervised or semi-supervised setting, so for the sake of comparison, a small amount of space will also be given to a review of purely supervised methods.

An exhaustive survey on the unsupervised learning methods has been given by Hammarström and Borin [2011]. It is not my purpose here to replicate this work, as this would be clearly redundant. Hence, in the following overview, I will restrict myself to highlighting some research directions which are especially relevant for this thesis, as well as completing the picture with the more recent important developments.

The positions in the following presentation are grouped by the key idea on what it means to 'learn morphology'. The three major task formulations considered here are: recognition of morph boundaries (Sec. 2.1), grouping morphologically related words (Sec. 2.2) and predicting properties of unknown words (Sec. 2.3). This classification is somewhat arbitrary: especially the grouping of related words is often used as a preprocessing step for segmentation. In such cases, the decision will be made according to whether the grouping or the segmentation method is the most important contribution of the respective algorithm. The chapter concludes

with a short discussion (Sec. 2.4), including a summary of the recent developments and trends.

2.1 Recognition of Morph Boundaries

Recognition of morph boundaries is the oldest and most common formulation of the morphology learning task. It consists of marking morph boundaries in surface forms of words and is usually evaluated by reporting the precision and recall of morph boundary detection.

Harris [1955, 1967] is usually cited as the first attempt to learn morphology from a corpus using statistical methods. It is directly inspired by the structuralist grammar: the underlying assumption is that morphemes can be discovered from the statistical peculiarities in the distribution of phonemes in words or utterances. The measure employed for this purpose is *Letter Successor Variety (LSV)*: the number of different letters that can follow a letter at a given position. It is expected to be low inside morphemes, but high at morpheme boundaries, as the different possibilities for the next morpheme make the next letter more unpredictable. This approach has inspired a lot of further research: the measure has been refined several times and tested on various languages in addition to English. Those methods are referenced and reviewed thoroughly by [Hammarström and Borin 2011, sec. 3.2.1].

Linguistica [Goldsmith 2006; Lee and Goldsmith 2016] is a toolbox for unsupervised learning of language structures, with special emphasis on morphology. The morphology of a language is encoded in two parts: a set of *signatures* (sets of suffixes that can be attached to a given stem) and a pairing between stems and signatures. As a criterion for selecting the best description, the authors employ the Minimum Description Length principle for the stem-signature coding. The search for the best hypothesis begins by suggesting splits into stem and suffix using LSV and proceeds by trying to improve the description length with various heuristics. The method takes a strongly simplified view on morphology: a word is assumed to be a concatenation of a stem and a single (possibly null) suffix.

Bordag [2006, 2007, 2008] presents a line of research based on a refined version of the LSV measure. The score for a given word is computed not on entire corpus, but only on a set of context-similar words. A combination of various weightings is proposed to adjust the LSV measure to various criteria (like the general letter and

letter n-gram frequency). Furthermore, the results of this procedure are used as training data for a supervised learning method (PATRICIA tree classifier), which generalizes the learned splitting schemes. Finally, an additional compound splitting algorithm is applied in the last publication.

Morfessor [Creutz and Lagus 2005a,b; Virpioja et al. 2013] is a tool for unsupervised segmentation that has been considered state-of-the-art for many years and still serves as a point of reference for other methods. The basic idea is to treat words as sequences of morphs chosen independently from a lexicon. The lexicon consists of a list of morphs and a probability for each morph. In addition to the corpus probability (the joint probability of all words given the lexicon), a prior probability of the lexicon is also taken into account. The latter is based on a character-level unigram model. The model is trained by means of Maximum A-Posteriori Likelihood estimation, combining the likelihood of the data and the prior probability of the lexicon. Simple morphotactics was subsequently added, consisting of 'prefix', 'stem' and 'suffix' morph types modeled by a Hidden Markov Model [Creutz and Lagus 2005b; Grönroos et al. 2014]. Semi-supervised training with a small amount of labeled training data is also possible [Kohonen et al. 2010].

Morfessor in its different varieties has been widely used as a baseline reference for morphological segmentation, as well as a source of unsupervised morph segmentation in further applications, like [Botha and Blunsom 2014; Varjokallio and Klakow 2016]. It has also been applied in speech recognition [Creutz et al. 2007] and machine translation [Virpioja et al. 2007].

Poon et al. [2009] propose a log-linear model for computing the probabilities of word segmentations. An important point of the model is the inclusion of morph context features: the left and right neighbors of a morph are taken into account while computing the probability of a concatenation. Priors on the number of morphs in the lexicon and the number of morphs per word are used to control the learning process. The inference is done using a method called Contrastive Estimation [Smith and Eisner 2005]: the objective is to shift as much of the probability mass as possible from an artificially generated neighborhood to the observed examples. A combination of Gibbs sampling and deterministic annealing is used for an efficient search in the space of possible segmentations. The learning algorithm can be easily adjusted to the task of supervised or semi-supervised learning: the training segmentations are simply held fixed, instead of being sampled.

Can [2011] uses a Dirichlet process to model the distributions of stems and suffixes in a simplified model of morphology, in which words consist of a stem and a single suffix. The learnt segmentations are subsequently generalized by employing a probabilistic hierarchical clustering algorithm to induce sets of related suffixes, which correspond to morphological paradigms.

Spiegler [2011] presents a generative probabilistic model of words and their segmentations, where the latter are treated as a latent variable. The model can be trained either in the supervised setting using Maximum Likelihood estimation, or in the unsupervised using the Expectation Maximization algorithm. Its main assumption is the dependence between the presence or absence of a morph boundary and character transitions at various points in a word. It thus models morph boundaries without modeling morphs explicitly – an approach which is designed especially to deal with small training data and extensive data sparsity. Those problems occur for the language that is the main point of interest of Spiegler's thesis – Zulu. The model also achieved very good results in Arabic, scoring first in MorphoChallenge 2009.

Botha and Blunsom [2013] and [Botha 2014, chap. 5] propose a segmentation method based on learning mildly context-sensitive grammars. Although the basic objective of the model is segmentation into morphs, it is specifically designed to capture non-concatenative phenomena (as discontinuous morphs), targeting especially Semitic root-and-pattern morphology. In order to establish the learning method, the Bayesian framework of Adaptor Grammars, originally designed for context-free grammars, is extended to handle mild context-sensitivity. The inference method utilizes a sampler based on the Metropolis-Hastings algorithm. The model is evaluated on Arabic and Hebrew data.

MorphoChallenge [Kurimo et al. 2010] was a yearly competition taking place between 2005 and 2010, which provided standardized tasks, datasets and evaluation measures for unsupervised morphological segmentation. In addition to recognizing morph boundaries, the task involved also the labeling of morphs, which required a correct identification of allomorphs and differentiation of homonymic morphs. For unsupervised learning algorithms, this step is actually much more challenging than the segmentation itself and many approaches omit it entirely.

Supervised and semi-supervised morphological segmentation has also been approached using general-purpose classifiers, like Memory-Based Learning [van den Bosch and Daelemans 1999] or Conditional Random Fields [Ruokolainen et al. 2013; Würzner and Jurish 2015]. The usual setup is to classify each letter or gap between letters in a word. The features needed for classification are extracted from neighboring letters. The set of classes might be binary (morph boundary present/absent) or more sophisticated, distinguishing different types of morph boundaries. On the other hand, Cotterell et al. [2015] present a method for joint supervised learning of morphological segmentation, morph labeling and morphotactics using a semi-Markov Conditional Random Field.

2.2 Grouping Morphologically Related Words

An alternative formulation of the morphology learning task is to aim at a relational description which groups related words together without making claims about their internal structure. The concept of morphological relatedness usually encompasses inflected forms of the same lemma, but may be also extended to include whole derivational families. The algorithms used for such tasks often involve distance or similarity measures on words, which typically combine different kinds of similarity (e.g. orthographical and contextual) into one measure. The evaluation of such approaches is problematic because of a lack of standardized datasets and evaluation procedures, as well as an unclear definition of 'morphological relatedness'. Some authors apply their method as a step towards learning segmentation and use the latter task for evaluation.

Schone and Jurafsky [2000] propose an algorithm for learning the stem-suffix segmentation involving Latent Semantic Analysis. Candidates for suffixes and pairs of related words are extracted by inserting the words into a trie and looking for nodes, at which a branching occurs. Additionally, LSA is used to compute a vector of distributional features for each word. By means of statistical analysis of the cosine similarities of the vectors, the candidate pairs can be scored according to distributional similarity. Only pairs with a statistically significant similarity are kept. The evaluation is done by converting the results into sets of related words and comparing them to sets extracted from a gold standard segmentation (CELEX) by means of a clustering evaluation measure.

Yarowsky and Wicentowski [2000]; Wicentowski [2002] propose a model for unsupervised alignment between inflected forms and lemmas. The model uses a combination of similarity metrics, involving orthographic and distributional similarity, word frequency ratio and a probabilistic model of word transformations. The evaluation is particularly focused on handling irregular, non-concatenative patterns, like English irregular past tense formation. A further contribution of this work is a supervised, trie-based model, which learns inflection patterns from the aligned pairs.

Baroni et al. [2002] attempt to find pairs of morphologically related words using a combination of orthographic similarity (expressed in terms of edit distance) and distributional similarity (measured by the mutual information of word occurrence). The authors mention that the approach does not rely on the concatenative model of morphology and is able to identify pairs related by non-concatenative operations.

Neuvel and Fulop [2002] present an approach directly inspired by Whole Word Morphology. They attempt to learn word formation strategies by extracting alignments from word pairs and generalizing them to patterns. The method is non-statistical: all extracted patterns with frequency larger than some fixed threshold are considered to be morphological strategies. The extracted patterns are used to suggest new words. In the evaluation section, the authors report only the precision of the word generation experiment, stating that a plausible recall metric is impossible to compute.

[Chan 2008, chap. 5] presents an algorithm for identifying transformations between bases and inflected forms of words. The algorithm consists of grouping morphologically related words and identifying the base word, from which all other forms are most easily derived. A simple stem+suffix model of morphology is assumed. The grouping of similar words is done by means of stripping word final character n-grams of various sizes and comparing the resulting hypothetical stems.

Kirschenbaum et al. [2012]; Kirschenbaum [2013, 2015] employs a combination of orthographic and distributional similarity to identify groups of potentially related words. Such groups are then segmented using a multiple sequence alignment algorithm borrowed from bioinformatics. The segmentation model trained this way is subsequently used to segment infrequent words, for which the method

based on distributional similarity does not work because of the scarcity of statistically significant co-occurrences.

Narasimhan et al. [2015] and Luo et al. [2017] assume a process-based model of morphology operating on whole existing words. Related words are grouped into derivational chains [Narasimhan et al. 2015] or trees [Luo et al. 2017]. The edges in such graphs correspond to transformational rules turning one word into another. A log-linear model combining orthographical and distributional features is used to predict edge probabilities. The distributional features involve word embeddings obtained from the `word2vec` tool [Mikolov et al. 2013a]. Similarly to Poon et al. [2009], Contrastive Estimation is used for inference. The induced graphs are not evaluated directly: they are rather used as input for further tasks, especially segmentation into morphs. The latter relies on the assumption that the morphological rules operate by adding or removing affixes, so the strings of phonemes that are affected by a rule can be used directly to mark morph boundaries.

Supervised methods. The supervised equivalent of grouping morphologically related words is mainly the learning of lemmatization. Chrupała et al. [2008] demonstrate how lemmatization can be formulated as a classification task and employ a Maximum Entropy classifier to solve it. An alternative to classification is presented by Clark [2002]. It consists of learning the mapping between lemmas and inflected forms with a stochastic transducer trained with the Expectation Maximization algorithm. Another transducer-based model for learning mappings between lemmas and inflected forms was proposed by Lindén [2008, 2009]; Lindén and Tuovila [2009].

2.3 Predicting the Properties of Unknown Words

This section presents approaches that utilize some representation of morphology in order to solve tasks related to unknown (out-of-vocabulary) words. This represents a shift in focus compared to the approaches presented in previous sections: the representation of morphological knowledge is now only a means, rather than a goal in itself. In consequence, the only selection criterion for a morphology model is how well it captures regularities in the given dataset, regardless of its linguistic plausibility.

The methods presented here might be seen as either supervised or unsupervised. They are supervised in that the training data are labeled with additional

information (e.g. POS tag or word embedding) and the goal of a trained model is to predict the same kind of information for further data. However, the representation of morphology used for prediction is learnt in an unsupervised way: the training data contain no information about the morphological relationships between words.

Mikheev [1997] approaches the task of predicting POS-tags for out-of-lexicon words. He proposes a method for learning word-based prefix and suffix substitution rules that capture systematic tag correspondences between words differing with an affix (e.g. present and past tense verbs or infinitives and gerunds). The rules are induced from the lexicon of a POS tagger. The probability of a rule generating a word included in the lexicon, given the conditions in which the rule can apply, is taken as a measure of reliability of the rule. An additional statistical confidence analysis is carried out to take account of the frequency of the rules (infrequent rules are less trustworthy). The induced rules are subsequently applied to propose possible tags for out-of-vocabulary words. The evaluation shows an improvement in tagging performance in cases where this method is used.

Botha and Blunsom [2014] and [Botha 2014, chap. 4] present a method for factorizing word vectors in a continuous space language model to obtain vectors for single morphs. They utilize word segmentations provided by Morfessor-CatMAP. The experiments conducted by the authors show that modeling sub-word units contributes to lower language model perplexity and improvements in both machine translation and word similarity tasks. The factorization method can also be applied to other word decompositions than morph segmentation (e.g. lemma+inflection).

Soricut and Och [2015] observe that the regularities in the space of word embeddings can be used to learn morphological rules. For example, the difference vector between *car* and *cars* is very similar to the difference between *dog* and *dogs*. The rules are learnt by extracting candidate prefix and suffix substitutions and scoring them according to how well they predict the vector of the target word. The discovered sets of morphologically related words are presented in form of graphs. The evaluation is done by using the method to predict vectors for low-frequency and OOV words and computing the correlation of word similarities obtained from such vectors with the human-annotated semantic similarity. Thus, the focus of this work lies on learning to predict the *meaning* of rare words, represented by a word embedding, by applying morphological transformations leading to known words.

Faruqui et al. [2016] employ a graph-based label propagation algorithm in order to predict part-of-speech and inflectional tags for unknown words based on systematic word similarities. The morphological patterns are represented as a graph, in which words are nodes and related words are linked with an edge. The authors apply a broad understanding of morphological relatedness: e.g. pairs of words sharing a common affix are also linked with an edge.

2.4 Discussion

General conclusions. The above overview of the state-of-the-art in unsupervised learning of morphology enables us to draw the following conclusions:

1. The research on unsupervised learning of morphological segmentation started with heuristics like LSV, but gradually shifted to probabilistic models. The heuristic-based approaches seem to be no longer developed.

2. Although generative models (especially Morfessor) have been successful, the recent work tends to discriminative models, which are more flexible with respect to the features that they can incorporate.

3. Many approaches tried to learn morphology only from the string form of words. It turns out that such data do not provide enough information (e.g., as Soricut and Och [2015] point out, there is no way to distinguish pairs like *bold-boldly* from *on-only*).

4. Because of 3., it became common to incorporate distributional information into the training data. Recently, word embeddings became popular as a concise description of the word context distribution. Such information is useful for learning morphology and helps overcome the limitations of learning from string forms only.

5. Segmentation into morphs remains the main evaluation task, even for methods that do not target segmentation as the primary goal in learning morphology (like Luo et al. [2017]). A widely accepted method of evaluation for non-segmentational approaches is still missing, as are standardized datasets.

Influences on the present work. Inspired by the success of Morfessor and related approaches, Chapter 4 of this thesis presents a generative model for vocabularies, which involves morphology to account for structural similarities between

words. However, instead of focusing on morph segmentation, the model presented here is going to learn whole-word transformations. Considering the recent research, a shift in this direction is observable. The importance of incorporating distributional information, especially in form of word embeddings, will be accounted for. Additionally, the model is supposed to optionally make use of POS tags and if trained this way, be able to suggest POS tags and lemmas for unknown words. Finally, as is the case with many probabilistic models, also semi-supervised and supervised training will be possible in case the plausible training data are available.

Chapter 3

Morphology as a System of String Transformations

We begin the computational adaptation of word-based morphology by formulating a formal definition of morphological rule as a string transformation (Sec. 3.1). Furthermore, I present an implementation of such rules as Finite State Transducers, which provides us with a solid algorithmic basis for dealing with such transformations (Sec. 3.2). Finally, in Sec. 3.3, algorithms for discovering morphological rules in raw text are introduced, which constitutes the first step towards learning transformation-based morphology from data.

3.1 Formalization of Morphological Rules

The Morphological Strategies of WWM are defined as patterns that capture structural similarities between pairs of words. An appropriate formal realization of this idea would be an undirected binary relation on words. However, with the generative probabilistic model of Chapter 4 in mind, we aim at a dynamic description of words 'being derived from' other words, rather than a static description of words 'being related to' other words. For this reason, we are going to use directed rules. As rules might have multiple outcomes, or no outcome, each rule is a function $r : \Sigma^+ \mapsto 2^{\Sigma^+}$ mapping non-empty strings onto sets of non-empty strings.[1]

Obviously, morphological rules may not be arbitrary functions. Just like Morphological Strategies in WWM, they have to be expressed in terms of *patterns*, consisting of constant elements, which have to be matched exactly, and variable

[1] As usual in the automata and string-related literature, Σ denotes a finite alphabet.

elements (wildcards), which stand for a string preserved by the morphological rule, but varying from pair to pair.

More specifically, a *morphological rule with n variables* will be expressed as follows:

$$/a_0X_1a_1X_2a_2 \ldots X_na_n/ \mapsto /b_0X_1b_1X_2b_2 \ldots X_nb_n/ \tag{3.1}$$

The elements a_i and b_i are constants (literal strings), which usually represent the differing parts of words on the left-hand and right-hand side of the rule.[2] The elements X_i are variables (wildcards), which represent the part that is preserved by the rule, but varies from pair to pair. Additionally, the following conditions must be satisfied:

1. The variables must be retained in the same order on both sides of the rule.
2. For $0 < i < n$, either a_i or b_i has to be non-empty.

Example 3.1. The transformation between German word pairs like (*singen*, *gesungen*), (*ringen*, *gerungen*), (*trinken*, *getrunken*) or (*winken*, *gewunken*) can be expressed by the following rule:

$$/X_1\text{i}X_2/ \mapsto /\text{ge}X_1\text{u}X_2/ \tag{3.2}$$

The rule could also contain more constant elements to express the necessary conditions for its application:

$$/X_1\text{in}X_2\text{en}/ \mapsto /\text{ge}X_1\text{un}X_2\text{en}/ \tag{3.3}$$

Example 3.2. The following pattern:

$$/X_1\text{a}X_2/ \mapsto /X_2\text{a}X_1/ \tag{3.4}$$

does not constitute a valid morphological rule, because the variables are not retained in the same order on both sides of the rule. The corresponding string transformation – swapping two parts of a word separated by an 'a' – is unlikely to be part of morphology of any natural language.

[2]However, the constants a_i, b_i for a given position i do not have to differ. By being equal or containing a common part, they might also represent the context necessary for the rule to apply. For example, in the rule $/X\text{ate}/ \mapsto /X\text{ation}/$, both constants contain the common prefix 'at'. Formulating this rule as $/X\text{e}/ \mapsto /X\text{ion}/$ would correspond to the same string transformation, but would extend its coverage to a few further cases, like (*deplete*, *depletion*).

Example 3.3. The following pattern:

$$/X_1\text{i}X_2X_3/ \mapsto /\text{ge}X_1\text{u}X_2X_3/ \tag{3.5}$$

does not constitute a valid morphological rule, because both a_2 and b_2 are empty (and $n = 3$). The corresponding transformation can be expressed more concisely as $/X_1\text{i}X_2/ \mapsto /\text{ge}X_1\text{u}X_2/$.

As a conclusion from the notation (3.1), we can represent a rule with n variables as a vector of $2n + 2$ strings: $\langle a_0, a_1, \ldots, a_n, b_0, b_1, \ldots, b_n \rangle$. The variables are implicit in this representation. In practice, we will almost always use rules with one or two variables. A case of rule with zero variables, having form: $/a_0/ \to /b_0/$, is also possible. This corresponds to a whole-word substitution, which is a handy way of handling irregular correspondences and suppletion in the same formalism (although it violates the precondition that a rule expresses a *systematic* pattern).

With each rule, we can associate a function r, which transforms a word fitting to the left-hand side of the rule into a set of corresponding words fitting to the right-hand side:

$$r(v) = \{b_0X_1b_1 \ldots X_nb_n : X_1, \ldots, X_n \in \Sigma^+ \wedge v = a_0X_1a_1 \ldots X_na_n\} \tag{3.6}$$

Note that the outcome of the rule application is a *set* of words, rather than a single word. In general, the rule application might result in multiple different words, because there might be different ways of splitting the word into the sequence $a_0X_1a_1 \ldots X_na_n$. For example, the application of the rule $/X_1\text{a}X_2/ \to /X_1\text{ä}X_2\text{e}/$ to the German word *Kanal* results in the set: $\{$*Känale*, *Kanäle*$\}$. In case the word does not fit to the left-hand side of the rule, the rightmost condition is never fulfilled and the result is an empty set. Thus, the function r is defined on the whole of Σ^+.

Definition 3.4. Given a rule r and a set of words V, we define the *derivation relation* of r on V as follows:

$$\text{der}_r(V) = \{(v, v') : v \in V \wedge v' \in r(v)\} \tag{3.7}$$

Following Ford et al. [1997] and Neuvel and Fulop [2002], we observe that syntactic features (like the part-of-speech) are important information in determining whether a rule can apply to a given word. Thus, we extend our definition of a morphological rule to optionally include the part-of-speech information wherever

it is available. This is expressed in the following notation (cf. 3.1):

$$/a_0X_1a_1X_2a_2 \dots X_na_n/_\alpha \mapsto /b_0X_1b_1X_2b_2 \dots X_nb_n/_\beta \tag{3.8}$$

In this notation, α and β are labels expressing part-of-speech, inflectional information, or other syntactic features relevant for morphology. They are called 'tags' and are sequences of symbols from a predefined, finite set of tags $\mathcal{T}$. For example, a tag like N.GEN.PL, meaning 'noun genitive plural', is composed of three symbols: N, GEN and PL. In the above rule notation, α and β are fully specified tags: no wildcards are allowed.[3]

Example 3.5. If the word pairs given in Example 3.1 are labeled with part-of-speech information, e.g. (*singen*$_{\text{V.INF}}$, *gesungen*$_{\text{V.PP}}$), we can reformulate the rule (3.2) accordingly:

$$/X_1\text{i}X_2/_{\text{V.INF}} \mapsto /\text{ge}X_1\text{u}X_2/_{\text{V.PP}} \tag{3.9}$$

Because we are dealing with written language, some peculiarities of the orthography might also be relevant for morphology. An example would be the German verb nominalization: in a pair like (*machen*$_{\text{V.INF}}$, *Machen*$_{\text{N.NOM.SG}}$), the noun is homophonic with the verb infinitive, the only difference being the capitalization in writing. A rule like $/\text{m}X_1/_{\text{V.INF}} \mapsto /\text{M}X_1/_{\text{N.NOM.SG}}$ does not reflect this phenomenon properly: a separate rule would be needed for every initial letter. In such cases, it is beneficial to isolate the capitalization feature and treat it as a separate symbol of the alphabet. Thus, instead of *Machen*$_{\text{N.NOM.SG}}$, we will write $\{CAP\}$*machen*$_{\text{N.NOM.SG}}$, where the multi-character symbol $\{CAP\}$ reflects the capitalization of the next letter and is itself treated as a letter and thus part of the string. In general, we will denote multi-character symbols in string representations by writing them inside curly brackets. A plausible rule can then be formulated as follows:

$$/X_1/_{\text{V.INF}} \mapsto /\{\text{CAP}\}X_1/_{\text{N.NOM.SG}} \tag{3.10}$$

3.2 Finite State Automata and Transducers

Finite State Automata (FSAs) and Finite State Transducers (FSTs) are widely known theoretical devices for describing algorithms that manipulate strings. FSAs

[3]Allowing for wildcards in the tag transformation could allow for more powerful generalizations, e.g. employing the same rule for the genitive plural formation of nouns and adjectives. However, it would also make the computational model, as well as the machine learning algorithms that utilize it, more complex. Therefore, this possibility is not explored here.

constitute the theoretical foundation for the common solutions of tasks like regular expression matching or efficient text search. For a thorough introduction to automata theory and its applications, see for example Hopcroft et al. [2006].

FSTs are an extension of FSAs: in addition to the input tape, they also contain an output tape. Thus, they reflect a *relation* on pairs of strings. Thanks to a well-developed calculus of standard operations, FSTs are *the* tool for modeling, as well as efficient and modular implementation of complex string transformations. In the area of language processing, they have been applied to, among others, morphology (two-level morphology mentioned in Sec. 1.2), computational phonology [Kaplan and Kay 1994; Carson-Berndsen 1998], speech processing [Mohri et al. 1996, 2002; Rybach 2014], statistical machine translation [Vogel 2005] and spelling correction [Pirinen 2014; Silfverberg et al. 2016].

There exist multiple software packages and libraries for manipulating automata and transducers. The typical functionality involves compiling regular expressions to automata, algebraic operations (disjunction, intersection, composition etc.), optimization (determinization, minimization, ϵ-removal etc.) and lookup. The Xerox Finite State Tools (XFST) [Beesley and Karttunen 2003] was a widely used software package including a command language for transducer manipulation, a compiler of lexicon descriptions (`lexc`) and a compiler for two-level rules (`twolc`). The Stuttgart Finite State Transducer Tools (SFST) [Schmid 2005] is another toolkit, which includes a compiler for a language for building transducers, which is especially well suited for developing morphological analyzers. OpenFST [Allauzen et al. 2007] is a C++ library and a set of command-line tools dedicated to the manipulation of weighted transducers. Finally, HFST [Lindén et al. 2011] is a software library and set of command-line tools providing a unified interface to different transducer libraries, including SFST and OpenFST. An own transducer format is also provided: an immutable transducer optimized for the lookup operation [Silfverberg and Lindén 2009]. Furthermore, HFST is released as free software, provides official and well-developed Python bindings and includes a re-implementation of the whole XFST functionality, extending it further to weighted transducers. All of the automata processing described in this thesis was done using the HFST Python API, either with OpenFST backend or using HFST's optimized lookup format.

3.2.1 Preliminaries

This section introduces the concept of a weighted finite-state transducer, along with the common algebraic operations. The following presentation is based mainly

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+ \cup \{+\infty\}$	$+$	$\times$	0	1
Log	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Table 3.1: Commonly used semirings. $\oplus_{\log}$ is defined by: $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

on Mohri [2009] and may contain literal quotes thereof.

The set of weights of a weighted transducer must have the algebraic structure of a *semiring*, which is defined as follows:

Definition 3.6. A system $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is a *semiring* if $(S, \oplus, \bar{0})$ is a commutative monoid with identity element $\bar{0}$, $(S, \otimes, \bar{1})$ is a monoid with identity element $\bar{1}$, $\otimes$ distributes over $\oplus$, and $\bar{0}$ is an annihilator for $\otimes$.

The $\otimes$ operation is used to compute the weight of a path from the weights of single transitions along this path. $\oplus$ is used to combine the weights of different possible paths into a weight of a string mapping.

Table 3.1 lists the semirings commonly used as transducer weights. In the Boolean semiring, the weight simply denotes the existence of a transition or path. The probability semiring is used to combine probabilities, while the log semiring is suitable for computation involving minus log-probabilities. The tropical semiring is a variant of the log semiring applying the *Viterbi approximation*: the sum of path weights is approximated by the weight of the best path.

Definition 3.7. A *weighted transducer* T over a semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is a 7-tuple $T = (\Sigma, \Delta, Q, q_0, F, E, \rho)$, where Σ is a finite input alphabet, Δ a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ the set of final states, E a finite set of transitions, which are elements of $Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times S \times Q$, and $\rho : F \mapsto S$ is a final weight function.[4]

In the above definition, ϵ denotes the *empty symbol*, which is the identity element of string concatenation.

A path π of a transducer T is an element of E^* with consecutive transitions (i.e. the target state of each transition in the sequence equals the source state of

[4]The definition given here differs from Mohri [2009] in two points. Firstly, the initial state is assumed to be unique and there is no initial weight associated with it (or the initial weight is assumed to be $\bar{1}$). Secondly, no multiple transitions between the same pair of states with the same symbol pair are allowed. As Mohri [2009] states, every transducer not fulfilling those assumptions can be converted to an equivalent one that does. The definition in the form given here is also encountered e.g. in Jurish [2010] and is the basis for the transducer class implemented by HFST.

the following transition). We denote by $p[\pi]$ the source state and by $n[\pi]$ the target state of the path. A path π is called a *cycle* if $p[\pi] = n[\pi]$. We also define:

- $P_T(Q_1, Q_2)$ as the set of all paths π with $p[\pi] \in Q_1 \wedge n[\pi] \in Q_2$,
- $P_T(Q_1, x, Q_2)$ as the subset of $P_T(Q_1, Q_2)$ with input label x,
- $P_T(Q_1, x, y, Q_2)$ as the subset of $P_T(Q_1, x, Q_2)$ with output label y.

The *weight* $T[\pi]$ of a path π is defined as the $\otimes$-product of the weights of its transitions. The weight of a string pair (x, y) is defined as follows:

$$T(x, y) = \bigoplus_{\pi \in P(\{q_0\}, x, y, F)} T[\pi] \otimes \rho(n[\pi]) \tag{3.11}$$

Two transducers are called *equivalent* if they attribute the same weights to each pair of strings. Note that if no path in T corresponds to a pair (x, y), then $T(x, y) = \bar{0}$.

An unweighted transducer can be regarded as a weighted transducer over the Boolean semiring, in which every transition and final state has weight $\bar{1}$. A weighted automaton can be regarded as a weighted transducer, in which the input and output symbol of every transition are equal. The recognition of a language is then equivalent to mapping this language onto itself.

An *ϵ-transition* is a transition with ϵ as both input and output symbol. A transducer is called *ϵ-free* if it contains no ϵ-transitions. Every transducer can be converted into an equivalent ϵ-free transducer by an operation called *ϵ-removal*.[5]

A transducer is *deterministic* if it contains no two transitions with equal source state, input and output symbol. Every transducer can be converted into an equivalent deterministic transducer by the *determinization* operation. As the worst-case complexity of the determinization algorithm is exponential in the number of states [Hopcroft et al. 2006], the computational cost of this operation may be prohibitive for some transducers. However, most transducers used in practice are possible to determinize within reasonable time and space.

A transducer is called *minimal* if no equivalent transducer with a smaller number of states exists. Every transducer can be converted into a minimal transducer by the *minimization* algorithm.

[5] For a description of algorithms realizing the operations mentioned here, see Mohri [2009]. All described operations are implemented in the HFST library and its backend libraries.

The result of a *disjunction* of transducers T_1 and T_2 is a transducer $T_1 \cup T_2$ with following weights:

$$(T_1 \cup T_2)(x, y) = T_1(x, y) \oplus T_2(x, y) \tag{3.12}$$

The *concatenation* $T_1 \cdot T_2$ of two transducers is defined as follows:

$$(T_1 \cdot T_2)(x, y) = \bigoplus_{\substack{x=x_1x_2 \\ y=y_1y_2}} [T_1(x_1, y_1) \otimes T_2(x_2, y_2)] \tag{3.13}$$

The *composition* $T_1 \circ T_2$ of two transducers is obtained by taking the output of T_1 as input for T_2. The resulting transducer is described as follows:

$$(T_1 \circ T_2)(x, y) = \bigoplus_{z} [T_1(x, z) \otimes T_2(z, y)] \tag{3.14}$$

The *inversion* of a transducer is obtained by swapping input and output symbols of each transition. The resulting transducer is defined as follows:

$$T^{-1}(x, y) = T(y, x) \tag{3.15}$$

The *input/output projection* of a transducer T, denoted with $\pi_{in}(T)$ and $\pi_{out}(T)$, respectively, yields an automaton recognizing the input or output strings from the language recognized by T:

$$(\pi_{in}(T))(x, x) = \bigoplus_{y} T(x, y) \tag{3.16}$$

$$(\pi_{out}(T))(y, y) = \bigoplus_{x} T(x, y) \tag{3.17}$$

Finally, if T_1 is a weighted automaton, T_2 an unweighted automaton and $L(T_2)$ denotes the language recognized by T_2, then we define the *difference* of T_1 and T_2 as:

$$(T_1 \setminus T_2)(x, x) = \begin{cases} T_1(x, x) & \text{if } x \notin L(T_2) \\ \bar{0} & \text{otherwise} \end{cases} \tag{3.18}$$

3.2.2 Compiling Morphological Rules to FSTs

A morphological rule represented in the notation (3.1) can be easily converted to an FST. A general scheme of the resulting transducer is shown in Figure 3.1. Each block in this scheme represents a smaller transducer and the arrows repre-

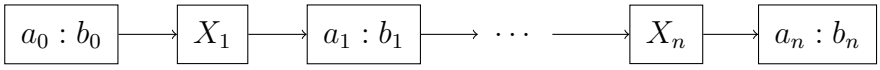

Figure 3.1: A scheme for converting morphological rules into FSTs.

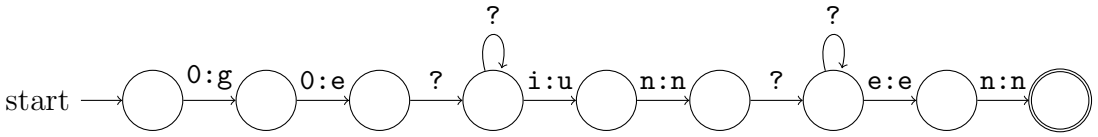

Figure 3.2: The transducer corresponding to rule (3.3).

sent concatenation. The smaller transducers are of two different kinds: transducers mapping the corresponding constants (like $a_0 : b_0$) or transducers representing the variables.

Each pair of corresponding constants is converted to a one-to-one alignment by padding the shorter constant with epsilon symbols. With an efficient lookup in mind, we follow the principle of matching the input early and giving the output late: therefore, the input sequence is padded at the end and the output sequence at the beginning.

The transducers representing the variables correspond to the parts of the words which are unspecified by the rule and remain unchanged by the rule application. Thus, they are simple universal language acceptors, mapping every character onto itself and containing a loop. It should be noted however, that those transducers must not match the empty sequence: therefore, in addition to the loop, they also contain one non-loop transition.

Figure 3.2 shows the transducer corresponding to the rule (3.3). Following the notation of Beesley and Karttunen [2003], `0` denotes the epsilon symbol and `?` denotes mapping an arbitrary symbol onto itself.

In case tags are used, they are attached at the end of the string representations. Figure 3.3 shows an FST corresponding to the rule (3.9). Note that the identity symbol (`?`) should *not* match any tag symbols.

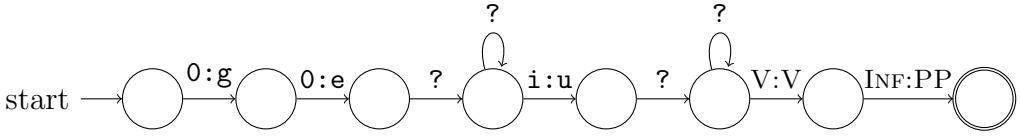

Figure 3.3: The transducer corresponding to rule (3.9).

3.2.3 Binary Disjunction

In further applications, we will often use a transducer corresponding to the disjunction of all known morphological rules. While building large transducers using disjunction, we have to be careful about when to minimize. On one hand, it is desirable to operate on minimized transducers as often as possible. On the other hand, minimization can be a costly operation – especially for large transducers due to its high theoretical complexity[6] – so we should try to keep the number of times it has to be called low and the size of the transducers it is called on small. Hence, the following two straightforward strategies for disjuncting a large number of transducers are both inefficient:

1. Applying all disjunctions first and minimizing the resulting transducer.
2. Disjuncting with one transducer at a time and applying minimization after each disjunction.

Instead, I propose a strategy inspired by the well-known algorithm of *binary exponentiation*. The idea is to always apply minimization on a disjunction of two minimized transducers of equal size in order to obtain a minimal transducer twice that size. By 'size', I mean the number of elementary transducers (from the input data) contained in the given transducer.

Algorithm 3.1 presents the binary disjunction procedure. It utilizes two stacks: S_1 for storing intermediary transducers, and S_2 for storing their corresponding sizes. Note that the sizes are always powers of two, because a disjunction of two transducers of equal size yields a transducer twice as large. At each step of the iteration, we check whether the top of the stack contains two transducers of equal size (we assume that the functions First and Second return the top and next-to-top element without removing it from the stack). If so, we remove them from the stack and replace them with their minimized disjunction (lines 6-7). If not, we push an additional transducer from the input data to the stack with size 1 (lines 10-12). In case there are no more transducers left, the loop terminates.

After executing the loop in lines 4-17, S_1 contains transducers, the sizes of which are different powers of two (together they make up the binary representation of n). The resulting transducer is a disjunction of all that are left on the stack. This disjunction is computed in lines 19-21. As it typically involves only several transducers, it is sufficient to minimize only once after performing all disjunctions.

[6] By minimization, I mean here the operation that yields a minimal *deterministic* transducer equivalent to the given one. Determinization is thus implicitly part of minimization. The worst-case complexity of determinization is exponential.

Algorithm 3.1: Binary disjunction.

Input: Transducers $T_1, T_2, \ldots, T_n$ to disjunct.
Output: $T = \bigcup_i T_i$

```
1  S_1 ← Stack() ;
2  S_2 ← Stack() ;
3  i ← 1 ;
4  while True do
5  |  if Size(S_2) ⩾ 2 ∧ First(S_2) = Second(S_2) then
6  |  |  Push(S_1, Minimize(Pop(S_1) ∪ Pop(S_1))) ;
7  |  |  Push(S_2, Pop(S_2) + Pop(S_2)) ;
8  |  else
9  |  |  if i ⩽ n then
10 |  |  |  Push(S_1, T_i) ;
11 |  |  |  Push(S_2, 1) ;
12 |  |  |  i ← i + 1 ;
13 |  |  else
14 |  |  |  break ;
15 |  |  end
16 |  end
17 end
18 T ← Pop(S_1) ;
19 while ¬Empty(S_1) do
20 |  T ← T ∪ Pop(S_1) ;
21 end
22 T ← Minimize(T) ;
23 return T
```

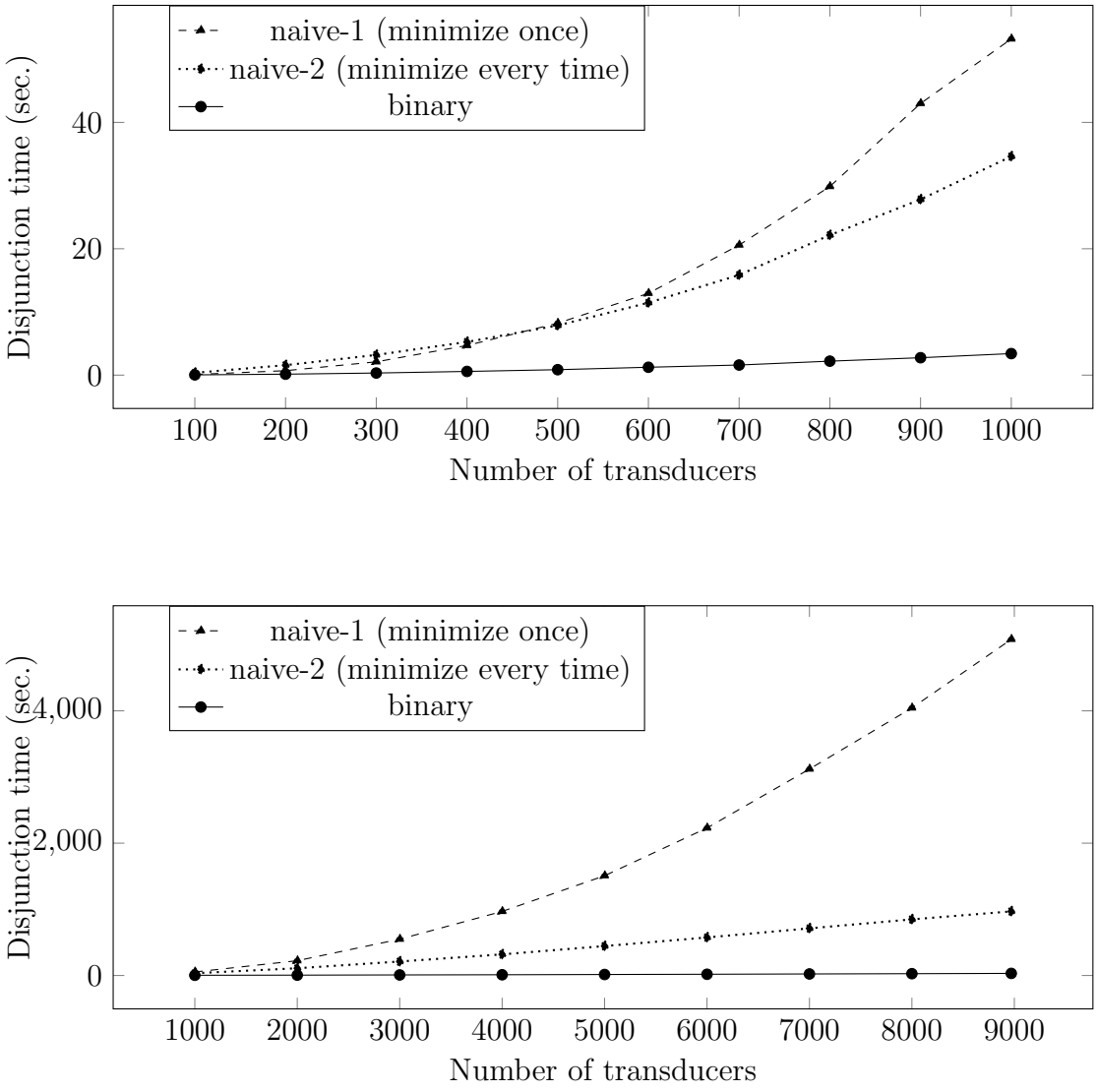

Figure 3.4: A comparison of running times for different disjunction strategies.

Figure 3.4 presents computation times of the two naïve strategies and the binary disjunction for different numbers of transducers. The difference is very clear: for the largest input, which consisted of 8970 transducers (each corresponding to a morphological rule), binary disjunction needed only 31.8 seconds, compared to 5080 sec. and 968 sec. for the two naïve strategies.

3.2.4 Computing the Number of Paths in an Acyclic FST

For the purpose of the probabilistic model developed in Sec. 4.1, we will need to be able to compute the *number* of distinct paths in an acyclic[7] transducer. As this task is not covered in the standard libraries and extracting all paths in order to count them is clearly wasteful, we will develop a dynamic programming algorithm for this purpose.

Let $s_{i,q}$ denote the number of distinct paths of length i ending in state q. This number can be computed using the following recursive formula:

$$s_{0,q} = \begin{cases} 1 & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

$$s_{(i+1),q} = \sum_{\substack{(q',a,b,w,q'')\in E \\ q''=q}} s_{i,q'} \tag{3.20}$$

The number of paths of length $i+1$ ending in state q is thus the number of paths of length i leading to states, from which q can be reached by one transition, multiplied by the number of transitions leading from such states to q. The number of paths n_{paths} can be obtained by summing $s_{i,q}$ for all final states:

$$n_{\text{paths}} = \sum_i \sum_{q\in F} s_{i,q} \tag{3.21}$$

In practice, we do not need to keep all the values $s_{i,q}$. At each iteration, we only need the values from the previous iteration. We thus denote the value for state q from the previous iteration with s_q and the value computed at the current iteration with s'_q.

Algorithm 3.2 presents the complete pseudocode. As we do not know the maximum path length, the iteration proceeds until $s'_q = 0$ for every state. We denote the set of transitions leaving state q of the transducer T with $\text{Tr}_T(q)$ and the target state of a transition t with $\text{tgt}(t)$.

[7]This question is not relevant for cyclic transducers, as they have an infinite number of paths.

Algorithm 3.2: Computing the number of paths in an acyclic DFST.

Input: An acyclic DFST $T = (\Sigma, \Delta, Q, q_0, F, E, \rho)$.
Output: $n_{\text{paths}} = |P_T(\{q_0\}, F)|$.

$n_{\text{paths}} \leftarrow 0$;
$s_{q_0} \leftarrow 1$;
`changed` $\leftarrow$ **true** ;
$s_q \leftarrow 0$ **for each** $q \in Q \setminus \{q_0\}$;
while `changed` **do**
  `changed` $\leftarrow$ **false** ;
  $s'_q \leftarrow 0$ **for each** $q \in Q$;
  for $q \in Q$ **do**
    if $s_q > 0$ **then**
      for $t \in \mathrm{Tr}_T(q)$ **do**
        $z \leftarrow \mathrm{tgt}(t)$;
        $s'_z \leftarrow s'_z + s_q$;
        `changed` $\leftarrow$ **true** ;
      end
    end
  end
  for $q \in F$ **do**
    $n_{\text{paths}} \leftarrow n_{\text{paths}} + s'_q$;
  end
  $s_q \leftarrow s'_q$ **for each** $q \in Q$;
end
return n_{paths}

The time complexity of the algorithm is $O(nm)$, where n is the number of states and m is the maximum path length. This holds under the assumption that the function $\mathrm{Tr}(\cdot)$ operates in constant time, which can be achieved by hashing. The space complexity is $O(n)$, as we store the values s_q and s'_q for every state.

3.2.5 Learning Probabilistic Automata

Probabilistic automata, i.e. weighted automata over the probability semiring, define probability distributions over regular languages. The *learning* of such automata is a statistical inference problem, in which the automaton is reconstructed from a sample from its distribution. This section presents one particular algorithm suited for this task, called ALERGIA [de la Higuera and Thollard 2000; de la Higuera 2010].

ALERGIA is a representative of the group of *state-merging algorithms*: it begins by constructing an automaton exactly reflecting the sample distribution, the so-called *prefix tree acceptor*. Then, pairs of similar states are identified and merged, which results in a generalization of the distribution. In order to identify whether the states can be merged, the so-called *ALERGIA test* (Algorithm 3.5) is used, which checks whether the differences in transition probabilities are statistically significant. Provided that the sample is indeed generated from a probabilistic automaton, ALERGIA is proven to identify the 'true' automaton in the limit (i.e. for arbitrarily large sample sizes) with probability one [de la Higuera and Thollard 2000].

The following presentation of the ALERGIA algorithm is based on de la Higuera [2010]. We will start by defining a frequency automaton, which can be used to reflect a multiset of strings:

Definition 3.8. A *deterministic frequency finite automaton* (DFFA) is a tuple $\mathcal{A} = (\Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr})$ where Σ is the alphabet, Q is a finite set of states, $\mathbb{I}_{fr} : Q \mapsto \mathbb{N}$ is the initial state frequency function, $\mathbb{F}_{fr} : Q \mapsto \mathbb{N}$ is the final state frequency function, and $\delta_{fr} : Q \times \Sigma \times Q \mapsto \mathbb{N}$ is the transition frequency function.

A frequency automaton is similar to a weighted automaton in that each transition and final state is labeled with a weight, in this case a natural number representing frequency. The following property requires that the sum of frequencies entering a state equals the sum of frequencies leaving it, which is necessary if the weights are supposed to reflect frequency:

Definition 3.9. A DFFA $\mathcal{A} = (\Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr})$ is said to be *well defined* if $\forall q \in Q : \mathbb{I}_{fr}(q) + \sum_{\substack{q' \in Q \\ a \in \Sigma}} \delta_{fr}(q', a, q) = \mathbb{F}_{fr}(q) + \sum_{\substack{q' \in Q \\ a \in \Sigma}} \delta_{fr}(q, a, q')$.

A DFFA can be easily converted into a probabilistic automaton by dividing the frequency of each transition or final state by the sum of frequencies leaving the state (including its final frequency). The latter value will be called *state frequency* and written as $FREQ(q)$.

The sample, from which the distribution will be learned, will be represented by a special kind of automaton, called *frequency prefix tree acceptor*, which is defined as follows:

Definition 3.10. Let S be a multiset of strings from Σ^* and $\textsc{Pref}(S)$ denote the set of prefixes of strings from S. The *frequency prefix tree acceptor* FPTA(S) is the DFFA $(\Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr})$ where:

- $Q = \{q_u : u \in \textsc{Pref}(S)\}$,
- $\mathbb{I}_{fr}(q_\epsilon) = |S|$,
- $\forall ua \in \textsc{Pref}(S), \delta_{fr}(q_u, a, q_{ua}) = |S|_{ua\Sigma^*}$,
- $\forall u \in \textsc{Pref}(S), \mathbb{F}_{fr}(q_u) = |S|_u$.

The key element of a state-merging algorithm is merging a pair of states which are deemed to be indistinguishable. This is realized by the STOCHASTIC-MERGE operation (Algorithm 3.3). The state q' is merged into q by transferring the incoming transition, together with its frequency, to q. Then, the subtree rooted in q' has to be merged with the one rooted in q. This is realized by the recursive operation STOCHASTIC-FOLD (Algorithm 3.4).

Algorithm 3.3: STOCHASTIC-MERGE.

Input: a DFFA $\mathcal{A}$, 2 states q, q' to be merged
Output: $\mathcal{A}$ updated, with q and q' merged
Let (q_f, a) be such that $\delta_{\mathcal{A}}(q_f, a) = q'$;
$n \leftarrow \delta_{fr}(q_f, a, q')$;
$\delta_{\mathcal{A}}(q_f, a) \leftarrow q$;
$\delta_{fr}(q_f, a, q) \leftarrow n$;
$\delta_{fr}(q_f, a, q') \leftarrow 0$;
return STOCHASTIC-FOLD$(\mathcal{A}, q, q')$

We use the algorithm ALERGIA-COMPATIBLE (Algorithm 3.6) to determine whether two states can be merged. The algorithm calls ALERGIA-TEST (Algorithm 3.5) on the final frequencies of the states and on the frequencies of

Algorithm 3.4: STOCHASTIC-FOLD.

Input: a DFFA $\mathcal{A}$, 2 states q and q'
Output: $\mathcal{A}$ updated, where subtree in q' is folded into q
$\mathbb{F}_{fr}(q) \leftarrow \mathbb{F}_{fr}(q) + \mathbb{F}_{fr}(q')$;
for $a \in \Sigma$ *such that* $\delta_{\mathcal{A}}(q', a)$ *is defined* **do**
  if $\delta_{\mathcal{A}}(q, a)$ *is defined* **then**
    $\delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)) \leftarrow \delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)) + \delta_{fr}(q', a, \delta_{\mathcal{A}}(q', a))$;
    $\mathcal{A} \leftarrow$ STOCHASTIC-FOLD$(\mathcal{A}, \delta_{\mathcal{A}}(q, a), \delta_{\mathcal{A}}(q', a))$;
  else
    $\delta_{\mathcal{A}}(q, a) \leftarrow \delta_{\mathcal{A}}(q', a)$;
    $\delta_{fr}(q, a, \delta_{\mathcal{A}}(q, a)) \leftarrow \delta_{fr}(q', a, \delta_{\mathcal{A}}(q', a))$
  end
end
return $\mathcal{A}$

each outgoing transition. The test states whether the difference between frequencies is statistically insignificant. The theoretical foundation for the formula is the *Hoeffding bound* [de la Higuera 2010, p. 199]: if the frequencies f_1, f_2 are generated from the same Binomial distribution, the test is passed with probability at least $(1-\alpha)^2$. Two states can be merged if the test is passed for all frequencies. Smaller values of α mean laxer bounds on the difference between frequencies generated from the same distribution, and thus easier merging.

Algorithm 3.5: ALERGIA-TEST.

Input: $f_1, n_1, f_2, n_2, \alpha > 0$
Output: a Boolean indicating whether $\frac{f_1}{n_1}$ and $\frac{f_2}{n_2}$ are sufficiently close
$\gamma \leftarrow \left|\frac{f_1}{n_1} - \frac{f_2}{n_2}\right|$;
return $\left[\gamma < \left(\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}\right) \cdot \sqrt{\frac{1}{2} \ln \frac{2}{\alpha}}\right]$

We can now turn to the description of the ALERGIA algorithm (Algorithm 3.7). Given a sample S of strings, it starts by building a prefix tree acceptor for this sample. Subsequently, certain states are marked as RED or BLUE. The RED states are the ones which are certainly going to be part of the resulting automaton. The BLUE states are candidates for merging with a RED state. Initially, the set of RED states consists only of the initial state, corresponding to the empty prefix, while the set of BLUE states consists of all the children of the initial state.

The algorithm proceeds by iteratively choosing a BLUE state q_b and attempting to merge it with one of the RED states. If no merge is possible, q_b is 'promoted', i.e. it becomes itself a RED state. At the end of each iteration, the set of BLUE

Algorithm 3.6: ALERGIA-COMPATIBLE.

Input: an FFA $\mathcal{A}$, two states $q_u, q_v, \alpha > 0$
Output: q_u and q_v compatible?
Correct $\leftarrow$ **true**;
if $\neg$ALERGIA-TEST$(\mathbb{F}_{\mathbb{P}\mathcal{A}}(q_u), \text{FREQ}_{\mathcal{A}}(q_u), \mathbb{F}_{\mathbb{P}\mathcal{A}}(q_v), \text{FREQ}_{\mathcal{A}}(q_v), \alpha)$ **then**
  Correct $\leftarrow$ **false**;
end
for $a \in \Sigma$ **do**
  if $\neg$ALERGIA-TEST$(\delta_{fr}(q_u, a), \text{FREQ}_{\mathcal{A}}(q_u), \delta_{fr}(q_v, a), \text{FREQ}_{\mathcal{A}}(q_v), \alpha)$ **then**
    Correct $\leftarrow$ **false**;
  end
end
return Correct

states is updated to include all children of RED states, which are not RED states themselves. The loop terminates if there are no BLUE states left.

The parameter t_0 denotes the minimum frequency of a state to be considered for merging. For states with low frequency, we assume to have too little information to make any statistically informed decision on whether to merge them. In order to avoid overgeneralization, we leave such states out.

Algorithm 3.7: ALERGIA.

Input: a sample S, $\alpha > 0, t_0$
Output: an FFA $\mathcal{A}$
$\mathcal{A} \leftarrow \text{FPTA}(S)$;
RED $\leftarrow \{q_\epsilon\}$;
BLUE $\leftarrow \{q_a : a \in \Sigma \cap \text{PREF}(S)\}$;
while CHOOSE q_b *from* BLUE *such that* $\text{FREQ}(q_b) \geqslant t_0$ **do**
  if $\exists q_r \in$ RED : ALERGIA-COMPATIBLE$(\mathcal{A}, q_r, q_b, \alpha)$ **then**
    $\mathcal{A} \leftarrow$ STOCHASTIC-MERGE$(\mathcal{A}, q_r, q_b)$
  else
    RED $\leftarrow$ RED $\cup \{q_b\}$
  end
  BLUE $\leftarrow \{q_{ua} : ua \in \text{PREF}(S) \wedge q_u \in \text{RED}\} \setminus$ RED
end
return $\mathcal{A}$

3.3 Rule Extraction

The first step to learning systematic structural correspondences between words is the extraction of candidate patterns from pairs of words which are likely to be morphologically related, i.e. sufficiently similar in terms of string edit distance. This task will be achieved in two stages: finding pairs of similar words (Sec. 3.3.1) and extraction of rules from word pairs (Sec. 3.3.2). The result of those two steps combined is a directed graph, in which words are vertices and the postulated morphological relations edges. A filtering of the graph according to some simple criteria (Sec. 3.3.3) removes a lot of noise and greatly diminishes the density of the graph. Placing further restrictions on admissible graph edges can introduce a certain degree of supervision, up to fully supervised training (Sec. 3.3.4).

3.3.1 Finding Pairs of Similar Words

For the task of finding pairs of similar words, we employ a slightly modified version of the FastSS algorithm [Bocek et al. 2007]. In its original version, the algorithm finds all pairs of words with Levenshtein distance[8] at most k in a given word list. It works by creating a deletion neighborhood for each word, consisting of all substrings obtainable by performing up to k deletions on the given word. The words are subsequently grouped according to the substrings (e.g. by sorting or hashing). Finally, the Levenshtein distance is computed for each pair of words sharing a substring. For small k ($\leqslant 3$), the algorithm has been shown to outperform several alternative approaches, like Neighborhood Generation or n-gram cosine similarity, among others.

For the purpose of discovering potential morphological rules, it is reasonable to modify the notion of edit distance. Firstly, morphological rules usually operate on groups of consecutive letters, rather than single letters independently, so deletion or substitution of a segment of consecutive letters should yield higher similarity than deletion or substitution of the same number of non-consecutive letters. Secondly, although we are going to permit word-internal alternations, more change should be permitted at the beginning and at the end of words, since that is where most morphological rules operate. Bearing in mind the representation (3.1), let l_{affix} denote the maximum length of a morphological constant at the beginning or the

[8]The *Levenshtein distance*, a.k.a. *string edit distance* [Levenshtein 1966; Wagner and Fischer 1974] between two strings is the minimum number of elementary edit operations, i.e. deletions, insertions or substitutions of a single symbol, needed to transform one of the strings into the other.

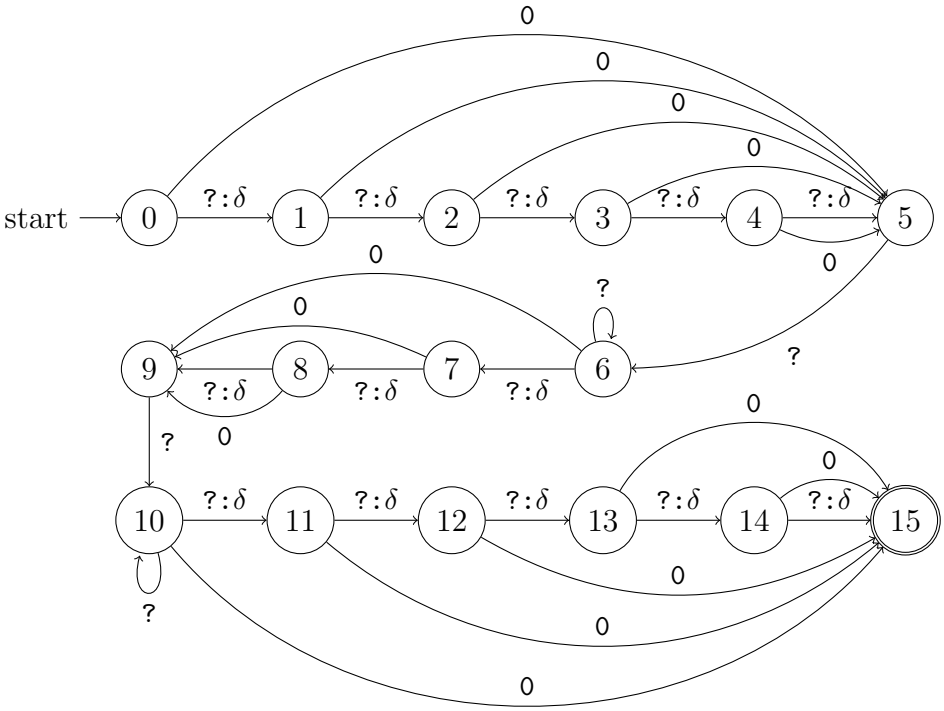

Figure 3.5: The transducer S_1 for generating a deletion neighborhood.

end of a word (a_0, b_0, a_n, b_n in (3.1)), l_{infix} the maximum length of a morphological constant inside the word (a_i, b_i for $0 < i < n$ in (3.1)) and $k_{\max}$ the maximum number of variables. In order to generate pairs which are related by a rule satisfying this constraint, we obtain the following constraints on a deletion environment: deleting up to l_{affix} consecutive letters at the beginning and end of the word, and up to l_{infix} consecutive letters in at most $k_{\max} - 1$ slots inside the word. The usual setting for those parameters, which covers a vast majority of morphological rules encountered in practice, is $l_{\text{affix}} = 5, l_{\text{infix}} = 3, k_{\max} = 2$.

Such settings allow for deletion of up to 13 letters in total, so that even for middle-length words it would consider all pairs to be similar. In order to prevent this, we introduce an additional constraint: the total amount of deleted characters must be smaller than half of the word's length. In this way, we can consider long affixes, but only if enough of the word is still left to form a recognizable stem.

With all those constraints, computing a deletion neighborhood of a word becomes a complex operation. It is therefore helpful to visualize and implement it using transducers. We will construct the transducer S mapping words to their deletion

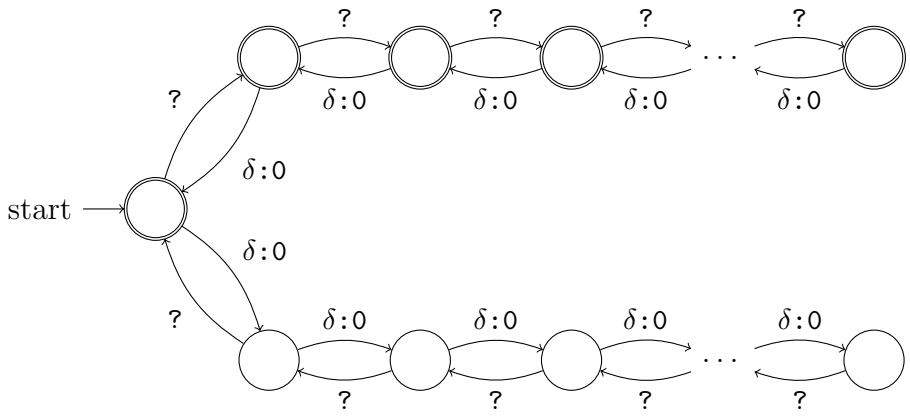

Figure 3.6: The filter S_2 ensuring, that no more than the half of a word is deleted.

neighborhoods as a composition of two simpler transducers: $S = S_1 \circ S_2$. The transducer S_1 (Fig. 3.5) performs the deletions, substituting a special symbol δ for each deleted character. The transducer consists of segments, corresponding to the deleted sequences: states 0-5 represent the prefix, 10-15 the suffix and 7-9 the infix. Between each pair of segments, an arbitrary number of identity mappings is performed (state sequences 5-6 and 9-10). The epsilon transitions, for example from states 0-4 to 5, correspond to a less-than-maximum number of deletions in a given slot. It can easily be seen that changing e.g. the parameter l_{affix} simply corresponds to altering the length of the top and bottom chains, just as l_{infix} correspond to the length of the middle chain and $k_{\max} - 1$ to the number of such middle chains.

The transducer S_2 (Fig. 3.6) takes the output of S_1 and checks whether the number of deletions is smaller than the number of remaining characters. As the general formulation of this problem cannot be solved by a finite-state machine, it requires a bound on word length. In my implementation, I restrict the maximum word length to 20 characters, but it is easy to change this parameter. The states of S_2 correspond to the difference between the number of letters and the number of deletions seen so far. The states above the initial state correspond to positive, and the ones below to negative values. Furthermore, S_2 removes the deletion symbols and returns the substring consisting of the remaining letters.

The composition $S_1 \circ S_2$ and a subsequent determinization and minimization yield a significantly more complex transducer S, with 129 states and 240 transitions. Thanks to the FST algebra, we do not have to understand this one.

We can now generate all pairs of similar words from a lexicon automaton L by performing the following composition:

$$P = (L \circ S) \circ (L \circ S)^{-1} \tag{3.22}$$

There are various ways to implement this in practice. Computing the composition directly is usually not feasible because of high memory complexity. One possibility is to use S for substring generation, but otherwise proceed as in the original FastSS algorithm: store the words and substrings in an index structure, either on disk or in memory, then retrieve words for each substring. Another possibility is to use S to generate substrings for a given word and then look the substrings up in the transducer $(L \circ S)^{-1}$ to obtain similar words. The latter composition can be computed statically. While the second approach is significantly slower, it has an advantage in providing a way to retrieve *all* words w' similar to a given word w at once. It is thus better suited for parallelization, especially in case the pairs (w, w') are subject to further processing.

3.3.2 Extraction of Rules from Word Pairs

Given a pair (w, w') of string-similar words, we want to extract morphological rules modeling the difference between those words. For this purpose, we first align the words on character-to-character basis and then attribute each character mapping either to a morphological constant or a variable.

Algorithm 3.8 is used to compute the optimal alignment between two words. It is a variant of the well-known dynamic programming algorithm for computing edit distance [Wagner and Fischer 1974]. In addition to the distance matrix D, three further matrices are used. Ξ remembers the last edit operation at each point. It can be used to reconstruct the optimal alignment after having computed the edit distance. Beginning in the bottom right corner, the arrows in Ξ show the optimal path. The matrices X and Y contain the character that is consumed respectively from word w or w' by the current edit operation.

After building the matrices D, Ξ, X, Y, the function Extract-alignment (Algorithm 3.9) walks the path from the bottom right to the top left corner by following the arrows in matrix Ξ. At each step, a pair of characters from the corresponding cells in matrices X and Y is picked as an element of the alignment.

The workings of Algorithm 3.8 are illustrated in Figure 3.7. In an actual implementation, there is no need to store the matrices Ξ, X, Y explicitly, as the

Algorithm 3.8: Aligning similar words with minimum number of edit operations.

Input: w, w' – string-similar words
Output: u, v – sequences of characters representing the optimal alignment
$d_{0,0} \leftarrow 0$; $x_{0,0} \leftarrow \epsilon$; $y_{0,0} \leftarrow \epsilon$; $\xi_{0,0} \leftarrow$ '?' ;
for $i \leftarrow 1$ **to** $|w| + 1$ **do**
  $d_{0,i} \leftarrow d_{0,(i-1)} + 1$; $x_{j,i} \leftarrow w_i$; $y_{j,i} \leftarrow \epsilon$; $\xi_{j,i} \leftarrow$ ' $\leftarrow$ ' ;
end
for $j \leftarrow 1$ **to** $|w'| + 1$ **do**
  $d_{j,0} \leftarrow d_{(j-1),0} + 1$; $x_{j,i} \leftarrow \epsilon$; $y_{j,i} \leftarrow w'_j$; $\xi_{j,i} \leftarrow$ ' $\uparrow$ ' ;
  for $i \leftarrow 1$ **to** $|w| + 1$ **do**
    $d_{j,i} \leftarrow \min\left[d_{(j-1),i} + 1, d_{j,(i-1)} + 1, d_{(j-1),(i-1)} + \delta(w_i, w'_j)\right]$;
    if $d_{j,i} = d_{(j-1),(i-1)} + \delta(w_i, w'_j)$ **then**
      $x_{j,i} \leftarrow w_i$; $y_{j,i} \leftarrow w'_j$; $\xi_{j,i} \leftarrow$ ' $\nwarrow$ ' ;
    else if $d_{j,i} = d_{j,(i-1)} + 1$ **then**
      $x_{j,i} \leftarrow w_i$; $y_{j,i} \leftarrow \epsilon$; $\xi_{j,i} \leftarrow$ ' $\leftarrow$ ' ;
    else
      $x_{j,i} \leftarrow \epsilon$; $y_{j,i} \leftarrow w'_j$; $\xi_{j,i} \leftarrow$ ' $\uparrow$ ' ;
    end
  end
end
$u, v \leftarrow$ EXTRACT-ALIGNMENT$(X, Y, \Xi, |w|, |w'|)$;
return u, v

Algorithm 3.9: EXTRACT-ALIGNMENT.

Input: X, Y, Ξ, n, m
Output: u, v – alignment
$i \leftarrow n$; $j \leftarrow m$; $k \leftarrow 0$;
while $i > 0$ *or* $j > 0$ **do**
  $u_k \leftarrow x_{j,i}$; $v_k \leftarrow y_{j,i}$; $k \leftarrow k + 1$; $i' \leftarrow i$;
  if $\xi_{j,i} \in \{\nwarrow, \leftarrow\}$ **then**
    $i \leftarrow i - 1$;
  end
  if $\xi_{j,i'} \in \{\nwarrow, \uparrow\}$ **then**
    $j \leftarrow j - 1$;
  end
end
return REVERSE(u), REVERSE(v)

$D =$

		t	r	i	f	f	t
g	0	1	2	3	4	5	6
e	1	1	2	3	4	5	6
t	2	2	2	3	4	5	6
r	3	2	3	3	4	5	5
o	4	3	2	3	4	5	6
f	5	4	3	3	4	5	6
f	6	5	4	4	3	4	5
e	7	6	5	5	4	3	6
n	8	7	6	6	5	4	4
	9	8	7	7	6	5	5

$\Xi =$

		t	r	i	f	f	t
g	?	←	←	←	←	←	←
e	↑	↖	↖	↖	↖	↖	↖
t	↑	↖	↖	↖	↖	↖	↖
r	↑	↑	↖	←	↖	↖	↖
o	↑	↑	↑	↖	↖	↖	↖
f	↑	↑	↑	↖	↖	↖	←
f	↑	↑	↑	↖	↖	↖	←
e	↑	↑	↑	↑	↑	↑	↖
n	↑	↑	↑	↑	↑	↑	↖
	↑	↑	↑	↑	↑	↑	↖

$X =$

		t	r	i	f	f	t
g	ϵ	t	r	i	f	f	t
e	ϵ	t	r	i	f	f	t
t	ϵ	r	r	i	f	f	t
r	ϵ	t	r	i	f	f	t
o	ϵ	ϵ	ϵ	i	f	f	t
f	ϵ	ϵ	ϵ	i	f	f	t
f	ϵ	ϵ	ϵ	i	f	f	t
e	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	t
n	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	t

$Y =$

		t	r	i	f	f	t
g	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
e	g	g	g	g	g	g	g
t	e	e	e	e	e	e	e
r	t	t	t	t	t	t	t
o	r	r	r	ϵ	r	r	r
f	o	o	o	o	o	o	o
f	f	f	f	f	f	f	ϵ
e	f	f	f	f	f	f	ϵ
n	e	e	e	e	e	e	e
	n	n	n	n	n	n	n

$$u = (\epsilon \quad \epsilon \quad \text{t} \quad \text{r} \quad \text{i} \quad \text{f} \quad \text{f} \quad \epsilon \quad \text{t})$$
$$v = (\text{g} \quad \text{e} \quad \text{t} \quad \text{r} \quad \text{o} \quad \text{f} \quad \text{f} \quad \text{e} \quad \text{n})$$

Figure 3.7: Matrices D, Ξ, X, Y and aligned sequences u, v after executing Algorithm 3.8 on the word pair (*trifft*, *getroffen*). The shaded cells correspond to the optimal alignment, which is extracted by Algorithm 3.9.

values on the optimal path can be easily reconstructed while walking through D. However, they are shown here for the purpose of presentation and simplifying the pseudocode.

Given the alignment, we construct rules by turning sequences of identity mappings into variables. As identity mappings may also be parts of constants (describing the context in which a rule applies), many different rules can be extracted from a single pair. In order to compute them, we will use an auxiliary object called *mask*. A mask is a sequence of dashes and asterisks, like `--**-**--`. Applied to an alignment, it expresses that mappings at a position marked with a dash are parts of a constant, while mappings at a position marked with an asterisk are parts of a variable. Thus, applied to the mapping (u, v) from Figure 3.7, the above mask yields the rule $/X_1\text{i}X_2\text{t}/ \rightarrow /\text{ge}X_1\text{o}X_2\text{en}/$.

Algorithm 3.10 extracts the rules from a pair of words using the alignment algorithm described above. As converting a mask to a rule, realized by the function Make-Rule-From-Mask, is straightforward, the description of this step is skipped. Thus, the algorithm describes computing the different masks from a given alignment. The parameters $k_{\max}, l_{\text{affix}}, l_{\text{infix}}$ are used with the same meaning as in previous section.

The masks are computed incrementally on prefixes of the alignment. A priority queue is used to store the unfinished masks. Each queue item is a sextuple $\langle s, i, k, l_x, l_y, z\rangle$, with s being the current mask prefix, i the position in the alignment to be considered next, k the current number of variables (consecutive sequences of asterisks), l_x and l_y the lengths of the currently constructed constants (respectively on the left and right side) and z a Boolean variable stating, whether the currently constructed constant is a suffix (i.e. if $z =$ True, no more asterisks are allowed).

Initially, the queue contains one item, corresponding to an empty mask (line 3). In each iteration, one element is taken from the queue and considered. If $i > |u|$, the whole alignment has been processed and s contains a complete mask. In this case, the resulting rule is added to the result set (line 7). Otherwise, the next character pair in the alignment (u_i, v_i) is considered and either a dash or an asterisk is appended to the mask. As there might be multiple possible results of this step, each possibility is added to the queue for further processing. The various cases arise from the requirement on the mask to comply with the restrictions $k_{\max}, l_{\text{affix}}, l_{\text{infix}}$.

First, let us consider the case where $u_i = v_i$ (lines 9-27). The pair (u_i, v_i) can then be added either to a constant or to a variable. The first two subcases consider appending the pair to either a prefix (lines 10-12) or a suffix (lines 13-15).

Algorithm 3.10: Extracting rule candidates from pairs of similar words.

Input: w, w' – string-similar words; $k_{\max}$ – max. number of variables; $l_{\text{infix}}, l_{\text{affix}}$ – max. length of infixes/affixes.

Output: R – a set of rules extracted from (w, w')

```
1  u, v ← Align(w, w');
2  q ← Queue(); R ← ∅;
3  Enqueue(q, ⟨ϵ, 1, 0, 0, 0, False⟩);
4  while not Empty(q) do
5  |  ⟨s, i, k, l_x, l_y, z⟩ ← Dequeue(q);
6  |  if i > |u| then
7  |  |  R ← R ∪ {Make-Rule-From-Mask(u, v, s)};
8  |  else
9  |  |  if u_i = v_i then
10 |  |  |  if k = 0 ∧ max{l_x, l_y} < l_affix then
11 |  |  |  |  Enqueue(q, ⟨s · '-', i + 1, k, l_x + 1, l_y + 1, z⟩);
12 |  |  |  end
13 |  |  |  if k > 0 ∧ max{l_x, l_y} < l_affix then
14 |  |  |  |  Enqueue(q, ⟨s · '-', i + 1, k, l_x + 1, l_y + 1, True⟩);
15 |  |  |  end
16 |  |  |  if not z then
17 |  |  |  |  if k > 0 ∧ max{l_x, l_y} < l_infix then
18 |  |  |  |  |  Enqueue(q, ⟨s · '-', i + 1, k, l_x + 1, l_y + 1, z⟩);
19 |  |  |  |  end
20 |  |  |  |  if s_{i−1} ≠ '*' then
21 |  |  |  |  |  if k < k_max then
22 |  |  |  |  |  |  Enqueue(q, ⟨s · '*', i + 1, k + 1, 0, 0, z⟩);
23 |  |  |  |  |  end
24 |  |  |  |  else
25 |  |  |  |  |  Enqueue(q, ⟨s · '*', i + 1, k, 0, 0, z⟩);
26 |  |  |  |  end
27 |  |  |  end
28 |  |  else
29 |  |  |  if (k = 0 ∧ max{l_x, l_y} < l_affix) ∨ max{l_x, l_y} < l_infix then
30 |  |  |  |  Enqueue(q, ⟨s · '-', i + 1, k, l_x + 1 − δ(u_i, ϵ), l_y + 1 − δ(v_i, ϵ), z⟩);
31 |  |  |  else if max{l_x, l_y} < l_affix then
32 |  |  |  |  Enqueue(q, ⟨s · '-', i + 1, k, l_x + 1 − δ(u_i, ϵ), l_y + 1 −
   |  |  |  |   δ(v_i, ϵ), True⟩);
33 |  |  end
34 |  end
35 end
36 return R
```

u	ϵ	ϵ	t	r	i	f	f	ϵ	t
v	g	e	t	r	o	f	f	e	n
$/X_1\text{i}X_2\text{t}/ \rightarrow /\text{ge}X_1\text{o}X_2\text{en}/$	–	–	*	*	–	*	*	–	–
$/X_1\text{if}X_2\text{t}/ \rightarrow /\text{ge}X_1\text{of}X_2\text{en}/$	–	–	*	*	–	–	*	–	–
$/X_1\text{i}X_2\text{ft}/ \rightarrow /\text{ge}X_1\text{o}X_2\text{fen}/$	–	–	*	*	–	*	–	–	–
$/X_1\text{ri}X_2\text{t}/ \rightarrow /\text{ge}X_1\text{ro}X_2\text{en}/$	–	–	*	–	–	*	*	–	–
$/\text{t}X_1\text{i}X_2\text{t}/ \rightarrow /\text{get}X_1\text{o}X_2\text{en}/$	–	–	–	*	–	*	*	–	–
$/\text{t}X_1\text{i}X_2\text{ft}/ \rightarrow /\text{get}X_1\text{o}X_2\text{fen}/$	–	–	–	*	–	*	–	–	–
$/X_1\text{ri}X_2\text{ft}/ \rightarrow /\text{ge}X_1\text{ro}X_2\text{fen}/$	–	–	*	–	–	*	–	–	–
$/X_1\text{rif}X_2\text{t}/ \rightarrow /\text{ge}X_1\text{rof}X_2\text{en}/$	–	–	*	–	–	–	*	–	–
$/X\text{ifft}/ \rightarrow /\text{ge}X\text{offen}/$	–	–	*	*	–	–	–	–	–
$/\text{tri}X\text{t}/ \rightarrow /\text{getro}X\text{en}/$	–	–	–	–	–	*	*	–	–
$/\text{t}X_1\text{if}X_2\text{t}/ \rightarrow /\text{get}X_1\text{of}X_2\text{en}/$	–	–	–	*	–	–	*	–	–
$/\text{tri}X\text{ft}/ \rightarrow /\text{getro}X\text{fen}/$	–	–	–	–	–	*	–	–	–
$/\text{t}X\text{ifft}/ \rightarrow /\text{get}X\text{offen}/$	–	–	–	*	–	–	–	–	–

Figure 3.8: Rules extracted from the pair (*trifft*, *getroffen*) by Algorithm 3.10 and their corresponding masks, sorted by generality.

Furthermore, it can also be added to an infix (lines 17-19) or to a variable (lines 20-27). The latter two subcases are only possible if z is false, as an infix must be followed by a variable (otherwise it would be a suffix) and the last subcase itself extends a variable, i.e. adds an asterisk to the mask. In the last subcase, we also distinguish whether the variable has to be newly created (lines 21-23), or an already existing variable is extended (lines 24-26).

The second major case is $u_i \neq v_i$ (lines 28-34). Then, the pair (u_i, v_i) can only be added to a constant. The two subcases ensure that the currently constructed constant does not exceed the maximum length. The first subcase (lines 29-30) accounts for a prefix or an infix and the second (lines 31-32) for a suffix.

In lines 30 and 32, $\delta(\cdot, \cdot)$ denotes the Kronecker delta, which is 1 if both arguments are equal and 0 otherwise. This is due to the fact that adding an epsilon to a constant does not increase its length (as nothing is added). Those terms can be skipped in the case $u_i = v_i$, because the alignment must not contain pairs (ϵ, ϵ).

As the case $u_i = v_i$ has two possible outcomes (adding either a dash or an asterisk), concern may arise whether the algorithm has exponential complexity. However, the limitation of at most $k_{\max}$ sequences of consecutive asterisks and at most l_{infix} consecutive dashes between asterisks greatly constrain the number of possibilities, especially as those parameters are typically set to very small values ($k_{\max} = 2, l_{\text{infix}} = 3$). With such settings, no runtime problems arise.

Figure 3.8 illustrates the results of Algorithm 3.10 for the pair (*trifft*, *getrof-*

rank	rule	frequency	example
1	$/X\text{n}/ \rightarrow /X/$	8555	Epoche → Epochen
2	$/X/ \rightarrow /X\text{n}/$	8555	Epochen → Epoche
3	$/X\text{en}/ \rightarrow /X\text{e}/$	7465	aufgehenden → aufgehende
4	$/X\text{e}/ \rightarrow /X\text{en}/$	7465	aufgehende → aufgehenden
5	$/X\text{en}/ \rightarrow /X/$	6030	Abkürzungen → Abkürzung
6	$/X/ \rightarrow /X\text{en}/$	6030	Abkürzung → Abkürzungen
7	$/X\text{e}/ \rightarrow /X/$	5640	niedrige → niedrig
8	$/X/ \rightarrow /X\text{e}/$	5640	niedrig → niedrige
9	$/X\text{s}/ \rightarrow /X/$	4917	Erdbebens → Erdbeben
10	$/X/ \rightarrow /X\text{s}/$	4917	Erdbeben → Erdbebens
	...		
53	$/X\text{en}/ \rightarrow /X\text{t}/$	1194	nutzen → nutzt
54	$/X\text{en}/ \rightarrow /X\text{es}/$	1194	einfachen → einfaches
	...		
746	$/X_1\text{a}X_2/ \rightarrow /X_1\text{o}X_2/$	196	unterbrachen → unterbrochen
747	$/X_1\text{t}X_2/ \rightarrow /X_1\text{m}X_2/$	195	warten → warmen
748	$/X_1\text{n}X_2/ \rightarrow /X_1\text{l}X_2/$	195	Zähnen → Zählen
749	$/X_1\text{m}X_2/ \rightarrow /X_1\text{t}X_2/$	195	warmen → warten
750	$/X_1\text{l}X_2/ \rightarrow /X_1\text{n}X_2/$	195	Zählen → Zähnen
751	$/X_1\text{ge}X_2\text{t}/ \rightarrow /X_1X_2\text{en}/$	195	zugefügt → zufügen
752	$/\{\text{CAP}\}X_1X_2/ \rightarrow /X_1\text{sch}X_2/$	195	Allergie → allergische
753	$/X_1\text{sch}X_2/ \rightarrow /\{\text{CAP}\}X_1X_2/$	195	allergische → Allergie
754	$/X_1\text{ä}X_2\text{er}/ \rightarrow /X_1\text{a}X_2/$	195	Häuser → Haus
755	$/X_1\text{a}X_2/ \rightarrow /X_1\text{ä}X_2\text{er}/$	195	Haus → Häuser
756	$/\text{g}X/ \rightarrow /\text{ausg}X/$	194	gegraben → ausgegraben
757	$/\text{ausg}X/ \rightarrow /\text{g}X/$	194	ausgegraben → gegraben

Table 3.2: Example rules extracted from the German Wikipedia.

fen). The parameter values are set to $l_{\text{affix}} = 5, l_{\text{infix}} = 3, k_{\max} = 2$. All rules satisfying those requirements are extracted. The rules describe the transformation within the word pair with various degrees of generality: the more asterisks a mask contains, the more general is the resulting rule.

Table 3.2 shows example rules extracted from a corpus of 1 million sentences from German Wikipedia, where only word types with frequency of at least 8 were considered. It can be seen in the bottom of the table that patterns consisting of a single letter substitution and corresponding to accidental word similarity can be fairly frequent and get mixed with correct and important morphological rules, like the one deriving *Häuser* from *Haus*. For this reason, frequency alone is not a good criterion for determining whether a pattern is truly morphological. This is the motivation for the probabilistic model that will be presented in the next chapter.

3.3.3 Filtering the Graph

The graph extracted by the above two steps contains a lot of noise. Infrequent patterns either correspond to accidental string similarity, or are very specific, like the last rule in Fig. 3.8. For learning morphology however, only highly regular, general and frequent patterns are of interest. In order to filter out most of the noise, the criteria listed below are applied.

Minimum rule frequency. In order to be considered candidate for a morphological rule, a pattern has to be supported by a sufficient number of examples. Especially patterns occurring only once are poor candidates. In my experiments, I usually set this threshold to a value between 3 and 10.

Maximum number of rules. In many practical cases, capturing *all* rules is not really important. The number of rules (or, equivalently, edges) may however affect the runtime and resource usage of statistical inference algorithms used in further processing. For this reason, it is beneficial to keep the number of rules bounded. The rules are selected according to frequency: restricting the maximum number to n results in keeping only the n most frequent rules. This parameter is usually set to $10,000$, which results in acceptable processing times, while the likelihood of missing an important morphological rule is still very low. However, the plausible value might depend on the size of the training data.

Maximum number of edges per word pair. As Figure 3.8 illustrates, one word pair can yield many rules, most of them being too specific and thus useless. In case the overspecific rules are frequent, they may survive the two previous filtering steps and even have harmful effects on further processing, by taking irrelevant details into account. In order to prevent that, we allow only the top-n rules for each word pair, ordered by frequency. This parameter is usually set to between 3 and 5. For supervised learning, slightly higher values may be appropriate in order to capture irregularities and exceptions. On the other hand, higher values for unsupervised learning result in harmful random overfitting.

3.3.4 Supervised and Restricted Variants

Supervised training. The approach for extracting candidate edges described so far can be easily modified for the task of supervised learning. In this variant, instead of a plain list of words, we obtain a list of word pairs (w, w'), where w' is

supposed to be derived from w. We then simply skip the FastSS step and apply Algorithm 3.10 directly to the pairs provided by the training data. The result is a list of triples (w, w', r), with (w, w') belonging to the training data and r being a rule.

Left and right restrictions. The supervised training approach described above is equivalent to placing a restriction on admissible edges. Only edges (w, w', r), for which (w, w') belong to the training data, are allowed.

A similar, but more lax restriction can also be formulated: let V_L denote the set of words which are allowed on the left side of rules, and V_R the set of words which are allowed on the right side. Then, we can filter the extracted edges by allowing only edges (w, w', r) satisfying: $w \in V_L \wedge w' \in V_R$. A canonical example for this variant is learning lemmatization from an unlemmatized corpus, when an additional list of lemmas is provided, as it is done in Sec. 5.3. In this case, V_L is the list of lemmas and V_R are the word forms from the corpus that are not contained in V_L.

Chapter 4

Statistical Modeling and Inference

The purpose of this chapter is to define a generative probabilistic model of vocabularies (sets of words) together with their morphological analysis, much like it is done by Morfessor [Creutz and Lagus 2005a]. However, instead of segmenting words, the morphological relationships will be expressed as an application of a whole-word rule. In consequence, the morphological analysis takes the form of a directed graph – more specifically, a *branching*, i.e. a directed forest. The model is inspired by the Minimum Description Length[1] principle: high probability is attributed to analyses that make use of morphological rules to eliminate redundancies among related words.

In Sec. 4.1 I provide a high-level description of the model. For some of its components, multiple functions are possible – proposals for them are described in Sec. 4.2. Finally, in Sec. 4.3, I describe inference methods used to fit the model parameters to the data.

4.1 Model Formulation

Our present aim is to introduce a probabilistic model defining a probability distribution over graphs such as the one depicted in Figure 4.1, which describe the derivation relation between words. The vertices of the graph are words, while the edges are triplets consisting of a source word, a target word and a rule. The model thus defines a joint probability of vertices and edges.

The graph illustrates a view on the structure of the lexicon, in which only

[1]However, the use of continuous parameter spaces makes it difficult to apply MDL directly, so the actual inference is based on the somewhat related methods of Maximum Likelihood and Maximum A-Posteriori likelihood.

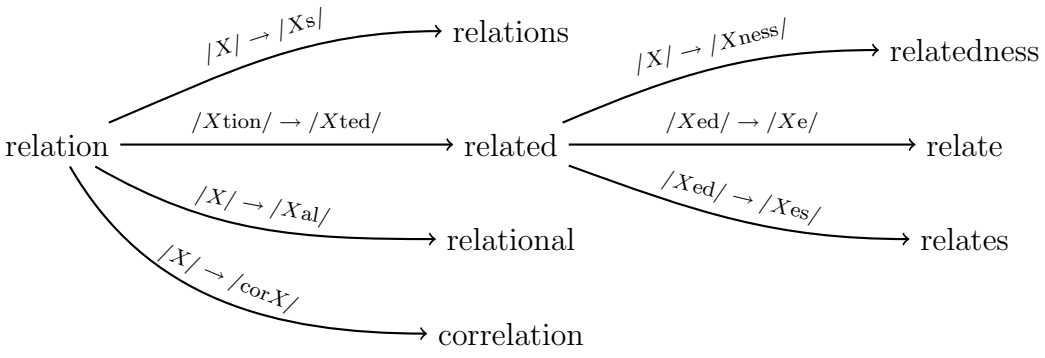

Figure 4.1: An example tree of word derivations.

the root words are listed explicitly. The edges correspond to the information that a rule applies to a certain word. From an information-theoretic perspective, this results in a compression of information compared to the case where each word had to be listed explicitly and independently. The redundancy arising from structural similarities between related words is thus avoided.

4.1.1 The Basic Model

Let V denote the set of vertices of the graph, R the set of known morphological rules, $E \subset V \times V \times R$ the set of labeled edges and $V_0 \subseteq V$ the set of root nodes. Furthermore, let $\theta = \langle \theta_{root}, \theta_{edge} \rangle$ denote parameters of the model components, which will be explained in Sec. 4.2. The joint probability of vertices and edges given rules and model parameters is defined as follows:

$$\begin{aligned} P(V, E|R,\theta) \propto & \prod_{v \in V_0} \lambda_V P_{root}(v|\theta_{root}) \\ & \times \prod_{v \in V} \prod_{r \in R} \prod_{v' \in r(v)} \begin{cases} p_{edge}(v, v', r|\theta_{edge}) & \text{if } (v, v', r) \in E \\ 1 - p_{edge}(v, v', r|\theta_{edge}) & \text{if } (v, v', r) \notin E \end{cases} \end{aligned} \tag{4.1}$$

The model consists of two components: a *root distribution* P_{root} and an *edge probability function* p_{edge}. The root distribution is a probability distribution over strings, defined over whole of Σ^+. It is used to generate the root words in the graph. Each word in the graph generates a set of *possible edges*, which may arise by applying a known rule to the word in consideration to derive a new word. The edge probability function p_{edge} assigns a probability to a possible edge. Note that p_{edge} is not itself a probability distribution: it does not sum to one over all

possible edges. Rather, for each possible edge (v, v', r), p_{edge} generates a Bernoulli distribution with success probability $p_{edge}(v, v', r|\theta_{edge})$. λ_V is a hyperparameter corresponding to the expected number of root nodes. Its inclusion is explained in Sec. 4.1.2.

4.1.2 Distributions on Subsets of a Set

Let $\mathcal{A}$ be a countable set and $f : \mathcal{A} \mapsto [0, 1]$ be a probability distribution function on $\mathcal{A}$. Then we can define a probability distribution g on finite subsets of $\mathcal{A}$ in the following way:

$$g(A) \propto P(S = |A|) \cdot |A|! \cdot \prod_{a \in A} f(a) \tag{4.2}$$

The model consists of drawing a size S of the set and then drawing S elements according to f. The factorial term is due to the fact that the elements can be chosen in any order. Note that the right side of (4.2) does not sum to one over all finite subsets, because the set elements are drawn with replacement, but multisets are not included in the domain. However, for our purposes it is sufficient to know the distribution up to a normalizing constant.

If we assume that the set size S has Poisson distribution, the formula further reduces to:

$$g(A) \propto e^{-\lambda} \frac{\lambda^{|A|}}{|A|!} \cdot |A|! \cdot \prod_{a \in A} f(a) \propto \prod_{a \in A} \lambda f(a) \tag{4.3}$$

with λ being the expected set size. In this formulation, we can also view λ as an arbitrarily defined marginal cost of adding one more element to the set. Especially if we want to model a preference of small sets (e.g. when modeling the set of morphological rules), even $\lambda < 1$ might be a reasonable choice. In this interpretation, it obviously does not have to reflect the expected set size in real data. Rather, as usual in Bayesian statistics, we use the hyperparameters to control the behavior of the model and drive it into the desired direction.

4.1.3 Penalties on Tree Height

The model defined by (4.1) contains one serious flaw: the edge probability is independent of the depth in the tree, at which the edge occurs. Thus, while Fig. 4.1 presents a desirable picture of a tree with small height, the trees actually produced by the model too often resemble the one depicted in Fig. 4.2: the derivations are arranged almost in a chain, producing a tree with unnecessarily large height and

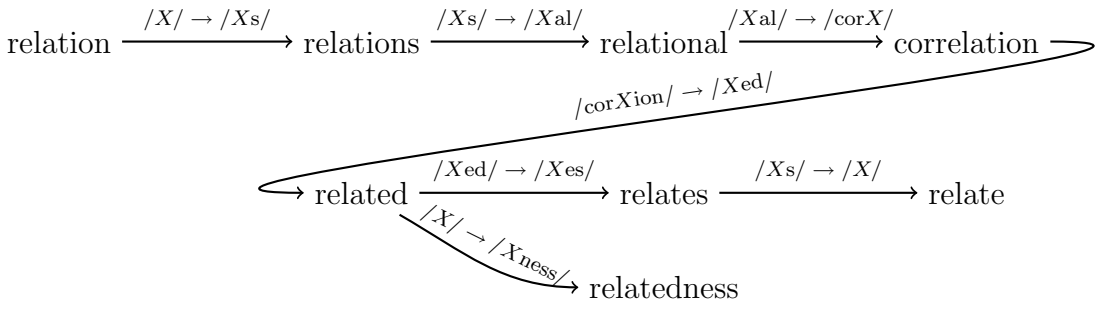

Figure 4.2: A 'bad' tree caused by the lack of a constraint on tree height.

small mean node degree. There is no incentive for the model to prefer the tree from Fig. 4.1 over the one from Fig. 4.2. While this may not matter for some applications (like vocabulary expansion), it might be fatal for others, especially those involving graph clustering.

The solution proposed here is, admittedly, going to be an ad-hoc one. It consists of introducing an additional parameter (let us call it α), which is a number between 0 and 1. For each node in the graph being at depth d, there is a probability $(1 - \alpha^d)$, that this node is a leaf. Thus, if the node contains outgoing edges, an additional factor α^d has to be included in the calculations.

4.1.4 Part-of-Speech Tags

The model formulated so far takes only string forms of words into account. There are two different ways to modify the model so that it includes POS tags, as shown in rule (3.9). For both of them, let $\mathcal{T}$ denote a finite set of possible tags. The presumably most obvious way is to treat pairs (v, t) of a word (character string) $v \in \Sigma^+$ and a tag $t \in \mathcal{T}$ as graph vertices. The rules are then functions $\Sigma^+ \times \mathcal{T} \mapsto 2^{\Sigma^+ \times \mathcal{T}}$ and the edges are quintuples $\langle v, t, v', t', r \rangle$ representing the derivation of a tagged word (v', t') from (v, t) with the rule r. Let θ contain all numeric parameters, i.e. in this case $\theta = \langle \theta_{root}, \theta_{roottag}, \theta_{edge} \rangle$. The model is then formulated as follows:

$$P(V, E|R, \theta) = \prod_{(v,t) \in V_0} P_{root}(v|\theta_{root}) P_{roottag}(t|v, \theta_{roottag}) \times \prod_{(v,t) \in V} \prod_{r \in R} \prod_{(v',t') \in r(v,t)} \begin{cases} p_{edge}(v, t, v', t', r|\theta_{edge}) & \text{if } (v, t, v', t', r) \in E \\ 1 - p_{edge}(v, t, v', t', r|\theta_{edge}) & \text{if } (v, t, v', t', r) \notin E \end{cases} \tag{4.4}$$

$P_{roottag}$ is an additional model component: a distribution of tag given the string form of a word, provided that it is a root in the graph. The tags of non-root nodes are determined by the rule that derives the respective node. Note that the model in this formulation permits tag ambiguity: each pair of a word form and a tag yields a different node. All tagged models employed in Chap. 5 use this formulation.

Another way of formulating a model with tags is to group the tags of all words in a separate variable, named T. In this variant, nodes and edges are defined as in (4.1), i.e. there is one graph node per word form (i.e. string). The rules, however, are tagged as in (4.4). $T \in \mathcal{T}^V$ is a vector indexed by elements of V, attributing a tag to each word. Thus, T_v denotes the tag of word v. The graph probability is computed in the same way as previously, yielding the following formula:

$$\begin{aligned} P(V,T,E|R,\theta) \propto & \prod_{v\in V_0} P_{root}(v|\theta_{root}) P_{roottag}(T_v|v,\theta_{roottag}) \\ & \times \prod_{v\in V} \prod_{r\in R} \prod_{\substack{(v',t')\in r(v,T_v) \\ t'=T_{v'}}} \begin{cases} p_{edge}(v,v',r|\theta_{edge}) & \text{if } (v,v',r)\in E \\ 1-p_{edge}(v,v',r|\theta_{edge}) & \text{if } (v,v',r)\notin E \end{cases} \end{aligned} \tag{4.5}$$

Although this formulation might seem less intuitive, it is useful if tags are to be treated as latent variables, as in the approach developed in Sec. 6.2. Contrary to the previous variant, no tag ambiguity is allowed here: the graph contains one node per word and T assigns one tag to each node.

4.1.5 Numeric Features

Also numeric word features can be incorporated into the generative model. The two features easily obtainable from unannotated corpora are word frequency and word embeddings.

Let $K \in \mathbb{N}^V$ be a vector of frequencies for words from V. To simplify the notation, we will denote the frequency of words v, v' with k, k' (instead of $K_v, K_{v'}$). Then, K has the following probability distribution:

$$\begin{aligned} P(K|V,E,\theta_{rootfreq},\theta_{edgefreq}) = & \prod_{v\in V_0} P_{rootfreq}(k|\theta_{rootfreq}) \\ & \times \prod_{(v,v',r)\in E} P_{edgefreq}(k'|k,r,\theta_{edgefreq}) \end{aligned} \tag{4.6}$$

Similarly, for word embeddings, let us assume that each word v has an associated d-dimensional vector of real numbers. Let $X \in \mathbb{R}^{V\times d}$ be a matrix of feature

values and x, x' the feature vectors associated with words v, v', respectively. The probability density for X is expressed as follows:

$$f(X|V, E, \theta_{rootvec}, \theta_{edgevec}) = \prod_{v \in V_0} f_{rootvec}(x|\theta_{rootvec}) \prod_{(v,v',r) \in E} f_{edgevec}(x'|x, r, \theta_{edgevec}) \tag{4.7}$$

In case further numeric features are available, they could be included using the same pattern. In general, we always need two distributions: one for modeling the feature values of root nodes and another for modeling the feature values of derived nodes given the value of the respective feature for the source node and the deriving rule. Because the model introduced here is generative, the feature values of derived nodes have to be modeled explicitly. Discriminative models, like the one of Luo et al. [2017], allow more flexibility in this regard, as it is possible to model a one-dimensional similarity measure (like cosine similarity) of the source and target vector, instead of modeling whole vectors.

High-dimensional numeric features correspond to the product of a large number of one-dimensional densities, which may yield very small likelihoods. Therefore, it is sometimes beneficial to multiply their contribution to the overall log-likelihood by a weight, which reduces the influence of the feature. For example, 30-dimensional word embeddings are weighted in the experiments with weight 0.1, so that their contribution is equivalent to a three-dimensional feature.

4.2 Model Components

The model introduced in the previous section consists of various component functions, like P_{root}, p_{edge} etc. For each of those functions, multiple alternatives can be considered. As each of the components can be changed independently of others, this produces a large number of possible configurations. Thus, not all of them are evaluated in the subsequent chapters.

Some of the models utilize neural networks to model complex transformations in a flexible, trainable way. Those should be understood as 'naïve', proof-of-concept models. Not much work was invested in the network architectures, as this would inflate the scope of this thesis. The networks were implemented using the keras[2] library in Python.

The neural network models for edge probabilities and edge word embeddings produce embeddings for morphological rules, which could be interesting on their

[2] `www.keras.io`

own. Similar rule embeddings in the edge model correspond to rules that are applied to similar groups of words, which could be used to learn paradigms. Similar rule embeddings in the edge vector model mean rules that conduct a similar transformation on the word embedding, which could be used to detect groups of rules with equivalent function (e.g. regular and irregular past tense formation). Those hypotheses were however not examined in the present work and remain topic for further research.

4.2.1 Root Models

The root models define the probability distribution $P_{root}(v|\theta_{root})$ of drawing the word v as a root of a tree. It is a distribution over arbitrary strings and it provides a general model of how the words of the language in question look like: which characters are used, character frequencies, basic phonotactics etc., while simultaneously avoiding to model morphology in any way.

Unigram Root Model

The simplest idea for a root model is to define a probability of each character independently. The probability of a character sequence (i.e. a word) is then the product of the probabilities of their characters. In order to obtain a valid probability distribution, we have to use a special symbol '#' as an end marker. The resulting model is defined formally as follows:

$$\Theta_{root} = \Delta^{\Sigma \cup \{\text{`\#'}\}} \tag{4.8}$$

$$P_{root}(v = c_1 c_2 \dots c_n | \theta_{root}) = \theta_{root}(\text{`\#'}) \prod_{i=1}^{n} \theta_{root}(c_i) \tag{4.9}$$

Θ_{root} is the parameter space of the model – in this case, it is equal to $\Delta^{\Sigma \cup \{\text{`\#'}\}}$, i.e. a simplex indexed with the characters from the alphabet. Each parameter value θ_{root} is a vector of probabilities for each character, summing to 1.

For practical reasons, it is useful to introduce a special symbol in the alphabet corresponding to an 'unknown' symbol and attribute some small probability to it. While computing the probabilities, all symbols from outside the alphabet in the string are converted to this unknown symbol. In this way, the model assigns nonzero probability to any character sequence.

ALERGIA Root Model

A probability distribution that makes statistically plausible generalizations from the training vocabulary can be obtained by applying the ALERGIA algorithm (Sec. 3.2.5). The resulting θ_{root} is a Deterministic Probabilistic Automaton (DPA) defining a distribution over strings. The parameter space Θ_{root} is in this case the space of all DPAs over the given alphabet.

Although the ALERGIA algorithm generalizes from the given sample, the resulting automaton still places a lot of restrictions on the possible words, i.e. it might attribute zero probabilities even to well-formed words of the learnt language. It is thus helpful to interpolate it with the unigram model, using a very small weight (e.g. 0.01) for the latter. This again ensures that every character sequence is assigned a non-zero probability.

4.2.2 Edge Models

The edge models define a function $p_{edge} : V \times V \times R \mapsto [0, 1]$, which attributes a 'success' probability to each edge. This is the only model component that is *not* a probability distribution.

Simple Edge Model

The 'simple' edge model attributes a fixed probability to each rule, independent of the word to which the rule is applied. It is thus formulated as follows:

$$\Theta_{edge} = [0, 1]^R \tag{4.10}$$

$$p_{edge}(v, v', r) = \theta_{edge}(r) \tag{4.11}$$

Neural Network Edge Model

A slightly more sophisticated idea for an edge model is to take into account the features of the source word (i.e. the word, to which the rule is applied), in addition to the rule. The source word is represented by a binary vector of letter n-gram features. The resulting edge probability is predicted by a small neural network shown in Fig. 4.3. The n-grams used as features are usually the top-100 most frequent word-initial or word-final n-grams across the whole corpus. The dimensionality of the internal layers (shown in brackets) is intentionally very small in order to avoid overfitting. Note that in a trivial case of using the embedding layer of size 1 and

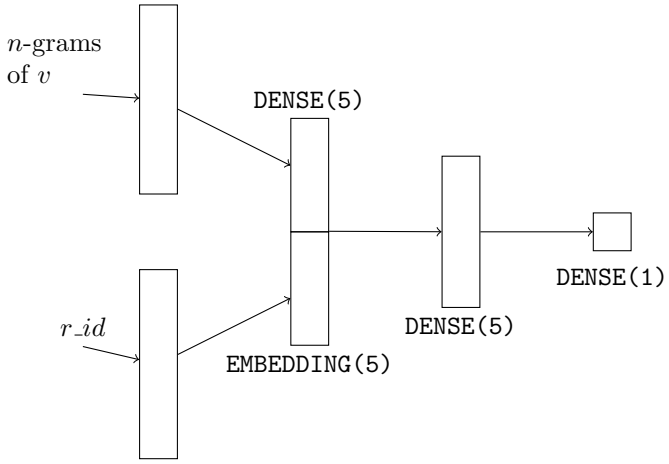

Figure 4.3: NN architecture for the edge model.

skipping the n-gram part, the NN model reduces to the 'simple' model. θ_{edge} are in this case the weights of the network.

4.2.3 Tag Models

The tag models are responsible for the distribution $P_{roottag}(t|v, \theta_{roottag})$ of a POS tag t given the string form of the word v. Only a distribution for the tags of the root nodes is needed, as the tags of derived words are determined by the deriving rule. The tag is not decomposed into single tag symbols, e.g. `<NN><NOM><FEM><SING>` is one tag. The set of all possible tags is denoted by $\mathcal{T}$.

Simple Tag Model

The simple tag model attributes a fixed probability to each possible tag, regardless of the string form of the word:

$$\Theta_{edge} = [0,1]^{\mathcal{T}} \tag{4.12}$$

$$P_{roottag}(t|v, \theta_{roottag}) = \theta_{roottag}(t) \tag{4.13}$$

RNN

The RNN tag model uses a neural network with a recurrent layer (Fig. 4.4) to predict the tag probabilities from the string form of the word. It consumes the characters of the word one by one, embeds them and feeds them to the recurrent layer. The softmax-activated layer at the end produces a probability distribution over tags. $\theta_{roottag}$ are the weights of the network.

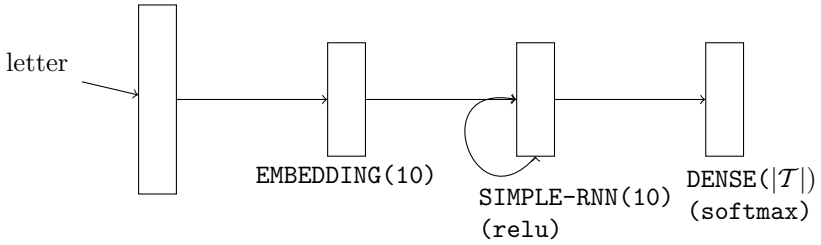

Figure 4.4: NN architecture for the root tag model.

4.2.4 Frequency Models

Zipf Root Frequency Model

The probability of some word type achieving a certain frequency in a corpus can be modeled using Zipf's law [Heyer et al. 2008, p. 91]. This results in the following parameter-less distribution:

$$f_{rootfreq}(k) = \frac{1}{k(k+1)} \tag{4.14}$$

Log-Normal Edge Frequency Model

Yarowsky and Wicentowski [2000] observed that the difference of log-frequencies (or equivalently, the logarithm of the frequency ratio) of word pairs following a morphological pattern forms a bell curve (Fig. 4.5). Without going into much theoretical detail, we can fit a Gaussian distribution (the default choice for bell curves, especially if outliers are unwelcome) to this data. For an edge $v \xrightarrow{r} v'$, if k and k' denote respectively the absolute frequencies of v and v', we obtain the following log-normal model of k' given k and the parameters (μ_r, σ_r^2) fitted for the rule:

$$\Theta_{edgefreq} = (\mathbb{R} \times \mathbb{R}_+)^R \tag{4.15}$$

$$f_{edgefreq}(k'|k, r, \theta_{edgefreq}) = f_{\mathcal{N}}(\log k - \log k'; \mu_r, \sigma_r^2) \tag{4.16}$$

with $f_{\mathcal{N}}$ being the Gaussian density:

$$f_{\mathcal{N}}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.17}$$

More exactly, the distribution should be discretized, which would yield the following formula: $f_{edgefreq}(k'|k, r, \theta_{edgefreq}) = \int_{k'}^{k'+1} f_{\mathcal{N}}(\log k - \log t; \mu_r, \sigma_r^2)dt$. However, in order to simplify the calculations, we assume that $\log k'$ and $\log(k'+1)$ are

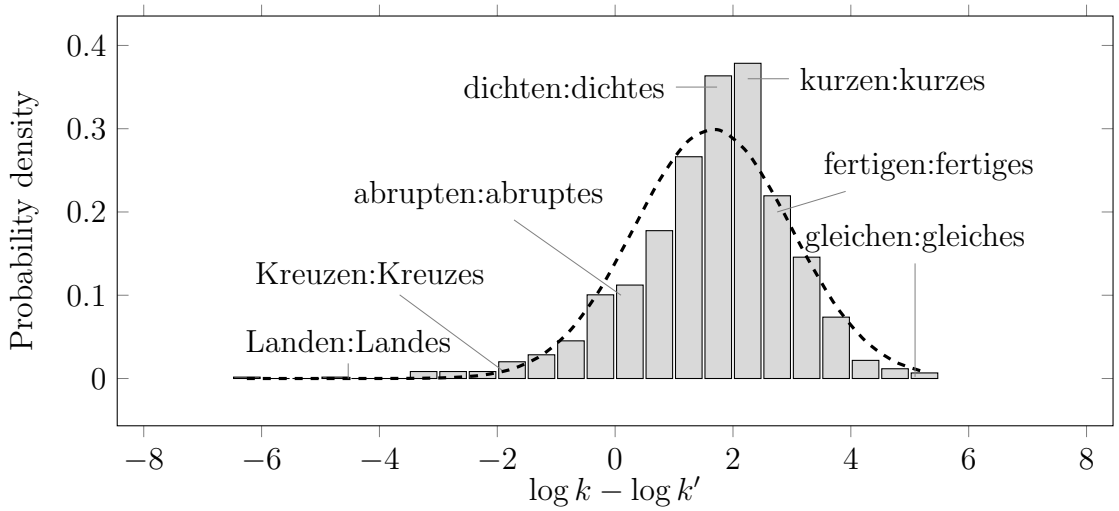

Figure 4.5: The difference in log-frequencies between words on the left and right side of the rule /*X*en/ → /*X*es/ in German. The dashed line is the best-fitting Gaussian density.

sufficiently close that the Gaussian density at both those points is roughly equal.

4.2.5 Word Embedding Models

The inclusion of word embeddings in the model is motivated by the observation that some morphological transformations (e.g. plural or comparative formation) correspond to a regular shift of the word vector [Mikolov et al. 2013b].

Gaussian Root Vector Model

The simplest way to model our uncertainty about some vectors of real values is to use a multivariate Gaussian distribution. The string form of the word is not taken into account. The model is then defined by the formulas given below, with $\theta_{rootvec} = \langle \mu, \sigma^2 \rangle$, both μ and σ^2 being d-dimensional vectors.

$$\Theta_{rootvec} = \mathbb{R}^d \times \mathbb{R}^d_+ \tag{4.18}$$

$$f_{rootvec}(x|v, \theta_{rootvec}) = f_{\mathcal{MN}}(x; \mu, \mathrm{diag}(\sigma^2)) \tag{4.19}$$

with $f_{\mathcal{MN}}$ being the multivariate Gaussian density:

$$f_{\mathcal{MN}}(x; \mu, \Sigma) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} e^{-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)} \tag{4.20}$$

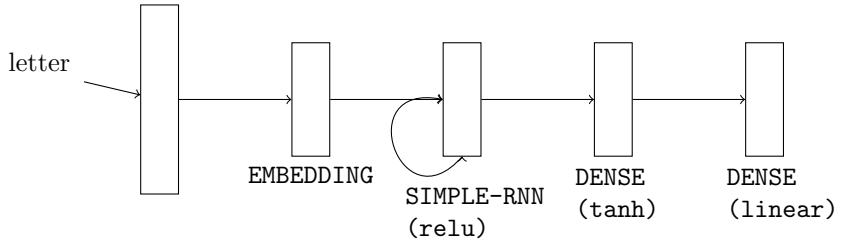

Figure 4.6: NN architecture for the root vector model.

As the components of an embedding vector are the outcome of dimensionality reduction, it is reasonable to assume their independence of each other. Thus, we use a diagonal covariance matrix, which simplifies the calculations significantly.

Gaussian Edge Vector Model

The underlying assumption of the Gaussian edge vector model is that applying a rule r to a word with embedding vector x yields a shifted vector $x+\mu_r$, where μ_r is the constant shift attributed to the rule. The difference between the observed and the expected vector is modeled as a Gaussian variable with zero mean and variance σ^2. In order to avoid overfitting (especially for rare rules), the variance is global to the whole model, rather than local for each rule. Thus, $\theta_{edgevec} = \langle\{\mu_r : r \in R\}, \sigma^2\rangle$. This yields the following model:

$$\Theta_{edgevec} = (\mathbb{R}^d)^R \times \mathbb{R}^d_+ \tag{4.21}$$

$$f_{edgevec}(x'|x, r, \theta_{edgevec}) = f_{\mathcal{MN}}(x' - x; \mu_r, \mathrm{diag}(\sigma^2)) \tag{4.22}$$

RNN Root Vector Model

The RNN root vector model models the distribution of embedding vectors for root words using the neural network architecture depicted in Fig. 4.6. The vector is predicted from the string form of the word, which is fed to the network letter by letter.

In order to convert the predicted vector into a distribution, we again use a multivariate Gaussian distribution. The 'error', i.e. the difference between the predicted and the observed vector, is modeled as a Gaussian variable with zero mean and covariance matrix $\mathrm{diag}(\sigma^2)$. The model parameters $\theta_{edgevec}$ thus consist of the weights of the network *and* the variance vector σ^2.

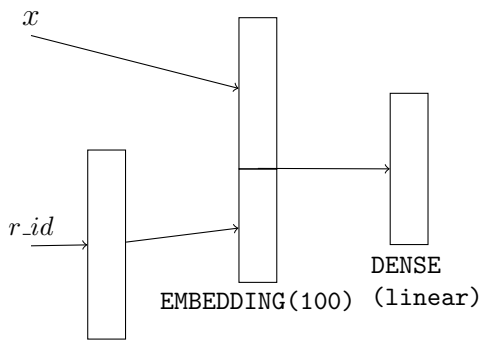

Figure 4.7: NN architecture for the edge vector model.

Neural Network Edge Vector Model

The NN edge vector model (Fig. 4.7) is a way to model more sophisticated transformations carried out on word embeddings by morphological rules. The input to the network is the vector of the source word, as well as the rule, which is embedded into a 100-dimensional vector space. The predicted output vector is computed by a single linear layer from both those values. In this way, instead of simple addition, more sophisticated ways of combining both values can be represented. The difference between the predicted and the observed vector is modeled as above, with a zero-mean Gaussian.

4.3 Inference

The inference methods on the model presented here are based on sampling from the distribution of graph edges, as the edges are a latent variable. In general, this problem fits into the framework of *learning in presence of hidden data*, which is formulated as follows: let X be the observed data, Y be the latent variable corresponding to the hidden data, and θ be the model parameter. The model defines the joint distribution $P(X, Y|\theta)$. If X and Y are discrete, like in the case of graph structures, the likelihood is expressed as follows:

$$\mathcal{L}(\theta; X) = P(X|\theta) = \sum_{Y} P(X, Y|\theta) \tag{4.23}$$

A usual goal is finding a parameter θ that maximizes the likelihood. This can be achieved by the *Expectation Maximization* (EM) algorithm [Dempster et al. 1977], which is based on the statement that maximizing the likelihood of the ob-

served data is equivalent to maximizing the following expected value:

$$\mathbb{E}_{Y|X,\theta} \log P(X, Y|\theta) \tag{4.24}$$

The maximization can be achieved by picking a θ_0 at random and iterating the following computation:

$$\theta_i = \arg\max_{\theta} \mathbb{E}_{Y|X,\theta_{i-1}} \log P(X, Y|\theta) \tag{4.25}$$

In many cases, computing the expected value directly is not possible, because it requires summing over all possible values of Y, which might be computationally prohibitive. Therefore, sampling approaches are often used to approximate the expectation. In particular, *Markov chain Monte Carlo* (MCMC) methods, that compute the next point of the sample from the previous point, are useful for complex models.

Sec. 4.3.1 contains a theoretical introduction to MCMC methods with focus on the Metropolis-Hastings (MH) algorithm. In Sec. 4.3.2, I develop a sampler based on the MH algorithm which can be used to draw samples of morphology graphs in the model presented in Sec. 4.1. As some model components may require negative examples, i.e. possible edges leading to non-existent vertices, for fitting, I describe a method of sampling such examples in Sec. 4.3.3. The graph sampler can be used for optimizing the model parameters (Sec. 4.3.4), as well as for model selection, i.e. finding the optimal set of morphological rules needed to explain the data (Sec. 4.3.5). Section 4.3.6 briefly mentions a different inference procedure, which is based on finding a single best graph, instead of computing expectations.

4.3.1 MCMC Methods and the Metropolis-Hastings Algorithm

The following introduction is based on Robert and Casella [2005]. Some definitions and theorems contain references to their direct counterparts in the source work. The proofs of the theorems are skipped here, as they can be found in the references. However, for the purposes of this thesis, we will consider only the case of Markov chains with a finite state space. This is sufficient for our applications and greatly simplifies the presentation of the theoretical foundations.

Let $\mathcal{X}$ be a finite set of *states* and f be a probability distribution over $\mathcal{X}$. In the following, we will be interested in drawing large samples $x^{(0)}, x^{(1)}, \ldots, x^{(n)}$, which are representative of f. In particular, we will use the sample to approximate

expected values of the form:

$$\mathbb{E}_f h(X) = \sum_{x \in \mathcal{X}} f(x)h(x) \approx \frac{1}{n} \sum_{i=0}^{n} h(x^{(i)}) \tag{4.26}$$

In order to achieve a good approximation, the sample need not necessarily be independent. In fact, we will work with samples, in which each point depends on the previous one. Such sample is the outcome of a *Markov chain*.

Definition 4.1. Given a transition matrix K, a sequence $X_0, X_1, \ldots, X_n, \ldots$ of random variables is a *Markov chain*, denoted by (X_n), if, for any t, the conditional distribution of X_t given $X_{t-1}, X_{t-2}, \ldots, X_0$ is the same as the distribution of X_t given X_{t-1}; that is:

$$P(X_t|X_0, X_1, \ldots, X_{t-1}) = P(X_t|X_{t-1}) = K(X_{t-1}, X) \tag{4.27}$$

In the case of a finite state space, K is simply a matrix of dimensions $|\mathcal{X}| \times |\mathcal{X}|$. However, I will use the notation $K(x, y)$ instead of k_{xy} for better readability. As $K(x, y)$ is the probability of a transition from x to y, it has to satisfy the condition $\forall_{x \in \mathcal{X}} \sum_{y \in \mathcal{X}} K(x, y) = 1$

Definition 4.2. A (discrete) Markov chain (X_n) is *irreducible* if all states communicate, namely if:

$$P(\tau_y < \infty | X_0 = x) > 0 \qquad \forall x, y \in \mathcal{X} \tag{4.28}$$

τ_y being the first time y is visited.

Definition 4.3. A probability distribution π over $\mathcal{X}$ is called the *stationary distribution* of a Markov chain with transition matrix K if it satisfies the following condition for every $x \in \mathcal{X}$:

$$\pi(x) = \sum_{y \in \mathcal{X}} \pi(y)K(y, x) \tag{4.29}$$

The stationary distribution gives the chain the following property: if $X_n \sim \pi$, then also $X_{n+1} \sim \pi$. In particular, if $X_0 \sim \pi$, it follows by induction that all X's are distributed according to π.

Definition 4.4. A Markov chain with transition matrix K satisfies the *detailed*

balance condition if there exists a function f satisfying:

$$K(y, x)f(y) = K(x, y)f(x) \tag{4.30}$$

for every (x, y).

Theorem 4.5 (Robert and Casella 2005, Th. 6.46, adjusted to the finite case)**.** Suppose that a Markov chain with transition matrix K satisfies the detailed balance condition with π a probability distribution. Then, the distribution π is the stationary distribution of the chain.

Definition 4.6 (Robert and Casella 2005, Def. 7.1)**.** A *Markov chain Monte Carlo (MCMC) method* for the simulation of a distribution f is any method producing an ergodic Markov chain (X_n) whose stationary distribution is f.

In great simplification, a Markov chain with finite state space is *ergodic* if the stationary distribution exists and is independent of the starting point. The details of this concept are not further relevant here, because in the following we are going to consider one particular MCMC method. Summarizing the above introduction, the foundation of MCMC methods is the convergence of an irreducible Markov chain to a stationary distribution, which justifies using the chain for drawing samples from this distribution.

We will now turn to a very general MCMC method called the *Metropolis-Hastings (MH) algorithm* [Metropolis et al. 1953; Hastings 1970]. This algorithm can be used to construct a Markov chain having an arbitrary probability distribution f as its stationary distribution. The main advantage is that sampling directly from f is not at all necessary. Instead, sample points are sampled from an instrumental distribution $q(\cdot|x)$, which might be completely unrelated to f.

Algorithm 4.1 presents a single iteration of the MH algorithm, computing the $t + 1$-st point of the sample from the previous one. The proposed new point is sampled from the instrumental distribution $q(\cdot|x^{(t)})$. Then, the algorithm decides randomly whether to accept it. In case of rejection, the chain stays at the current state.

The acceptance probability α is the key point of the algorithm, which combines the distributions f and q. Importantly, it does not require knowing f exactly: as it only involves ratios, it is sufficient that f is known up to a multiplicative constant. In case q is symmetric, i.e. $q(x|y) = q(y|x)$, the second ratio is dropped completely.

Algorithm 4.1: A single iteration of the Metropolis-Hastings algorithm.

Generate $Y_t \sim q(y|x^{(t)})$;
$\alpha \leftarrow \min\left\{\frac{f(Y_t)}{f(x^{(t)})}\frac{q(x^{(t)}|Y_t)}{q(Y_t|x^{(t)})}, 1\right\}$;
Generate $U \sim \text{Bernoulli}(\alpha)$;
if $U = 1$ **then**
  $x^{(t+1)} \leftarrow Y_t$
else
  $x^{(t+1)} \leftarrow x^{(t)}$
end

Theorem 4.7 (Robert and Casella 2005, Th. 7.2)**.** The Markov chain induced by the Metropolis-Hastings algorithm satisfies the detailed balance condition wrt. the target distribution f.

Corollary 4.8. The distribution f is stationary wrt. the Markov chain induced by the Metropolis-Hastings algorithm.

4.3.2 A MCMC Sampler for Morphology Graphs

The graph sampler described in this section has been proposed in [Sumalvico 2017]. It is inspired by similar approaches for dependency parsing [Mareček 2012; Teichmann 2014], which also use MCMC samplers for graphs. The key idea is to construct a proposal distribution which conducts a small change in the graph, like adding or removing a single edge.

Let $\mathcal{E}$ denote the set of all possible edges that were found in the preprocessing step described in Sec. 3.3. We will be interested in subsets $E \subseteq \mathcal{E}$ forming *branchings*, i.e. directed forests. The well-formedness conditions for a branching are that the graph contains no cycles and every node has at most one incoming edge.

Let $E^{(t)}$ denote the t-th branching in the sample. We will now formulate the proposal distribution $q(\cdot|E^{(t)})$, which can be used to propose the next branching. The main idea is to choose an edge from $\mathcal{E}$ with uniform probability and change its status in E, i.e. remove it if it is present, or add it if it is absent. Removing an edge always preserves the branching properties: the target node of the removed edge simply becomes a root. However, adding an edge can create a cycle or break the property of every node having at most one incoming edge. In this case, additional changes shall be proposed to repair the branching properties.

Let v_1 denote the source node and v_2 the target node of the edge that is supposed to be added. Furthermore, let v_3 denote the current parent of v_2, if it

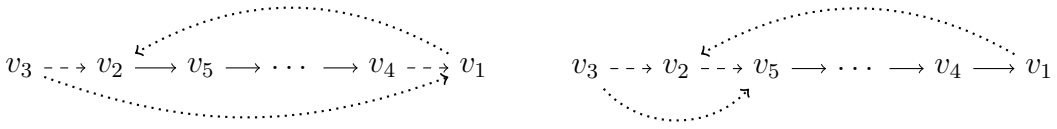

Figure 4.8: The two variants of the 'flip' move. The deleted edges are dashed and the newly added edges dotted. The goal of the operation is to make possible adding an edge from v_1 to v_2 without creating a cycle.

exists. If v_2 already contains an incoming edge, but v_1 is not a descendent of v_2, the proposal is to add the edge $v_1 \to v_2$ and remove the edge $v_3 \to v_2$. Note that in some settings, the same pair of vertices might be connected with multiple edges, labeled by different rules. In this case, v_3 might be equal to v_1. The procedure remains the same: the newly added edge is simply exchanged for the current incoming edge of v_2.

If v_1 is a descendent of v_2, adding an edge $v_1 \to v_2$ would create a cycle. In order to avoid that, I propose a change called 'flip', which is illustrated in Fig. 4.8. Here, v_4 denotes the parent of v_1 and v_5 denotes the direct child of v_2 on the path leading to v_1. As in the previous case, the overlapping of some of those nodes does not change the procedure. The move has two variants, each of which consisting of adding and removing two edges. In the first variant, we 'cut out' v_1 (together with the whole subtree rooted in it) from its current place and insert it between v_3 and v_2. In the second variant, we 'cut out' v_2 and move it (together with all its children except v_5) in a proper place to be the child of v_1.

Algorithm 4.2 presents the procedure for proposing the next branching. If a 'flip' operation is required, the variant is chosen randomly with equal probability. Note that a 'flip' move might be impossible if the other edge to be added ($v_3 \to v_1$ in the first and $v_3 \to v_5$ in the second variant) is not available in $\mathcal{E}$. In this case, the functions FLIP-* leave the current branching unchanged.

The choice of the next branching is uniquely determined by the choice of the edge (v_1, v_2, r) to be changed and, in case of the 'flip' move, the value of U. A move that adds or removes a single edge can be reversed by selecting the same edge again. A move that exchanges edges can be reversed by selecting the previously removed edge to be added again. Finally, each 'flip' move can be reversed by another 'flip' move. FLIP-1 is reversed by selecting $v_4 \to v_1$ to be added and applying FLIP-2, while variant 2 can be reversed by selecting $v_2 \to v_5$ and applying FLIP-1. In conclusion, the proposal distribution is symmetric.

Algorithm 4.2: Proposing the next branching in the sample.

Input: $E^{(t)}$ – current branching; $\mathcal{E}$ – set of all possible edges
Output: $Y^{(t)}$ – a proposal for the next branching
Choose an edge (v_1, v_2, r) randomly from $\mathcal{E}$;
if $(v_1, v_2, r) \in E^{(t)}$ **then**
  $Y^{(t)} \leftarrow E^{(t)} \setminus \{(v_1, v_2, r)\}$;
else if v_2 *is ancestor of* v_1 **then**
  Generate $U \sim \text{Bernoulli}(\frac{1}{2})$;
  if $U = 1$ **then**
    $Y^{(t)} \leftarrow \text{FLIP-1}(E^{(t)}, \mathcal{E}, v_1, v_2, r)$;
  else
    $Y^{(t)} \leftarrow \text{FLIP-2}(E^{(t)}, \mathcal{E}, v_1, v_2, r)$;
  end
else if $\text{parent}(v_2) \neq$ **nil then**
  Let v_3, r' be so that $(v_3, v_2, r') \in E^{(t)}$;
  $Y^{(t)} \leftarrow E^{(t)} \cup \{(v_1, v_2, r)\} \setminus \{(v_3, v_2, r')\}$;
else
  $Y^{(t)} \leftarrow E^{(t)} \cup \{(v_1, v_2, r)\}$;
end
return $Y^{(t)}$

The acceptance probability is thus calculated as follows:

$$\alpha^* = \exp\left[c(E^{(t)}) - c(Y^{(t)})\right] \tag{4.31}$$

$$= \exp\left[\sum_{(v_1,v_2,r)\in E^{(t)}\setminus Y^{(t)}} c(v_1, v_2, r) - \sum_{(v_1,v_2,r)\in Y^{(t)}\setminus E^{(t)}} c(v_1, v_2, r)\right] \tag{4.32}$$

with the final acceptance probability being: $\alpha = \min\{\alpha^*, 1\}$.

Note that the chain constructed this way is irreducible, because every branching can be reached from every other with nonzero (albeit very small) probability by first deleting all edges of the old branching and then adding all edges of the new branching.

Proposing the next branching has very low computational complexity: $O(h)$, where h is the maximum height of a tree in the branching. The only operation that is not performed in constant time is checking for cycles and determining v_5. If the tree height is kept low (for which, as described in Sec. 4.1.3, there are good reasons), the time complexity of proposing the next branching is as good as constant. This enables us to draw large samples in the order of magnitude of millions or billions branchings independently of $|\mathcal{E}|$, and thus input data.

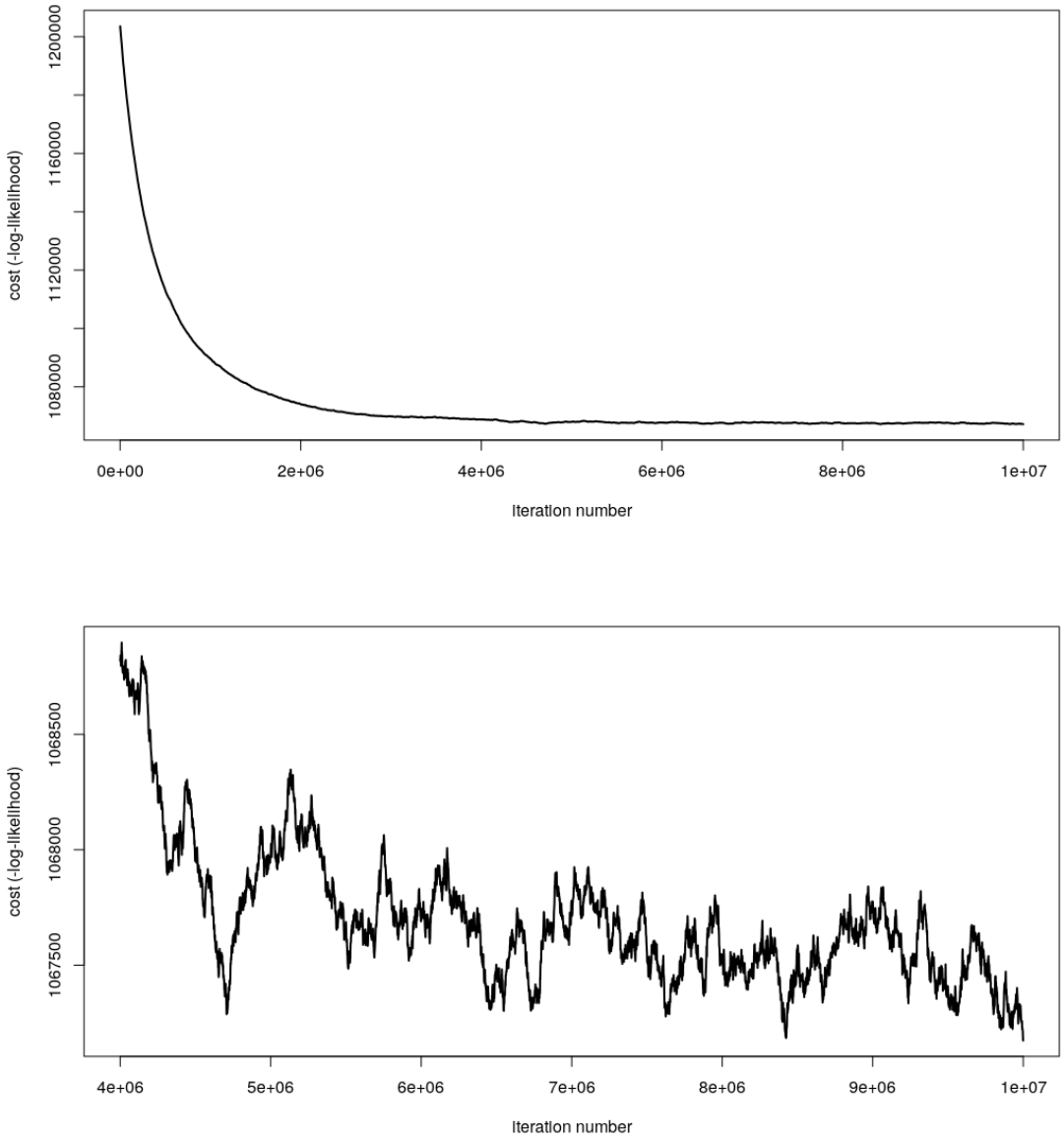

Figure 4.9: The trajectory of the graph sampler expressed as the cost (negative log-likelihood) of the graph at a given iteration. The top plot shows all 10 million sampling iterations, while the bottom plot shows iterations from 4 million onward. The acceptance rate in this run was 0.12.

Figure 4.9 illustrates the cost of the graphs obtained during sampling on a German dataset of 100,000 words. In total, 10,000,000 iterations of sampling were computed, preceded by 100,000 'warm-up' iterations, which were not counted into the computed expected value and are not shown in the graphs. The top graph shows the complete sample: it can be seen that the sampler converges towards branchings with high probabilities. The bottom graph is a magnifying picture of the long tail of the top graph: from the iteration number 4,000,000 onwards. It can be seen that once the sampler reaches branchings with high probabilities, it does not stay in one optimum, but rather oscillates in this area, still performing significant changes. The line thickness of both curves is equal – if the bottom curve appears much thicker, it is because of many small jumps back and forth all along the curve. If a small interval of this curve is magnified, a similar picture arises.

It can thus be seen that the sampler does not get locked in local optima, but rather is able to accept changes which temporary raise the cost of the graph, even in larger, mid-term trends. This guarantees that the sample space is properly explored. The acceptance rate in this sampling process was 0.12: roughly one move proposal in eight was accepted. Given the complexity of the graph model and, compared to that, the simplicity of the proposal distribution, it is definitely a satisfactory result.

A generally difficult problem with MCMC methods is finding a plausible stopping criterion: when does the distribution reach stationarity and when is the sample large enough to be representative for the purpose of computing expectations? In the experiments, I always used a fixed number of iterations (typically 10 million). The minimum number of iterations was tied to the number of possible edges (multiplied by a factor of between 2 and 10), so that there are reasonable chances of each possible edge being proposed at least once.

4.3.3 Sampling Negative Examples

Fitting the edge model requires negative examples, i.e. edges leading to non-existent nodes via one of the known rules. In case the edge probability is independent of the source node, like in the simple edge model, it is sufficient to count the negative examples, which can be achieved using Algorithm 3.2. However, the neural network model requires a more exact knowledge of each edge, i.e. the features extracted from the source node. Because of the large number of such edges, computing all of them is intractable. However, it might be sufficient to use a randomly selected sample of negative examples for training of the neural network. Algorithm 4.3 shows a simple

method to generate such a sample. The number of generated edges (n) is usually a multiple of the number of possible edges ($|\mathcal{E}|$), typically twice as much.

Algorithm 4.3: Sampling negative examples of edges.

```
Input: V, R, E, n
Output: S – a sample of random edges from outside E
S ← ∅;
while |S| < n do
    Choose v randomly from V;
    Choose r randomly from R;
    if r(v) ≠ ∅ then
        Choose v' randomly from r(v);
        if (v, v', r) ∉ S ∧ (v, v', r) ∉ E then
            S ← S ∪ {(v, v', r)};
        end
    end
end
return S
```

Because we are using only a subset of negative edges for training the neural network, they have to be weighted accordingly to the number of edges they represent. If k_r is the number of generated edges labeled with rule r, n_r is the number of possible edges (from $\mathcal{E}$) labeled with r and m_r is the total number of times r can be applied, each negative edge obtains the weight $\frac{m_r - n_r}{k_r}$. Algorithm 4.3 is quite a naïve sampling approach: it is easy to see that the probability of selecting a particular edge is not uniform (as it depends on $|r(v)|$).

4.3.4 Fitting the Model Parameters

Using the sampler introduced in Sec. 4.3.2, we can generate samples $E^{(1)}, E^{(2)}, \ldots$ from the conditional distribution $P(E|V, R, \theta)$. Such samples can be used to approximate expected values of functions over E. This enables us to apply the *Monte Carlo Expectation Maximization* (MCEM) algorithm [Wei and Tanner 1990] to find the maximum likelihood estimates of the model parameters. We aim to iteratively maximize the following expected value (cf. 4.25):

$$\theta_i = \arg\max_{\theta} \mathbb{E}_{E|V,R,\theta_{i-1}} \log P(V, E|R, \theta) \tag{4.33}$$

In the expectation step, besides the global log-likelihood, we also compute the expected relative frequency of each edge, i.e. the fraction of branchings in which

this edge appears. Those values constitute the edge weights used for fitting. The root weights can be easily obtained from the edge weights: the root weight of node v, i.e. the fraction of branchings in which it is a root, is one minus the sum of weights of all its incoming edges.

In the maximization step, the parameters of each model component are fit to maximize the likelihood of the sample. This is easy in case of most models. In the simple edge model, the ML estimate for rule probability is $\frac{n}{m}$, where n is the expected frequency of the rule (in this case, the sum of weights of all edges labeled with this rule) and m is the number of times the rule can be applied. The latter can be computed by applying Algorithm 3.2 to the composition of the lexicon and rule transducer. The Gaussian models can be fit to weighted samples by using weighted averages in estimation and the neural network models accept weighted samples as training data as well. In the latter case, the fitting consists of a complete, multi-epoch neural network training.

For fitting the neural edge model, we use the edge weights as targets, rather than sample weights. The training objective of the network is to predict for each edge the probability that it is included in a branching. In order to do it correctly, we also need to generate a weighted sample of non-existent edges with the method described in Sec. 4.3.3 and include it in the training data passed to the network.

The only model that poses problems for weighted fitting is the ALERGIA root model: it can only accept integer weights (interpreted as frequencies). However, the root model does not change significantly across fitting iterations, so we skip the maximization step for it.

The initial parameter value θ_0 is usually chosen randomly in the EM algorithm. However, in this case, we can easily guess a more-or-less reasonable value: it is the value fit to the full graph, i.e. when all edge and root weights are set to 1. We call this a 'naïve fit'. Using this value as θ_0 speeds up the convergence of the algorithm, because we already start in a region yielding relatively high likelihood.

Figure 4.10 shows the expected branching costs (minus log-likelihoods) obtained during the MCEM estimation. The first iteration, which is out of scale in the graph's Y axis, uses the parameters obtained via naïve fitting. The algorithm converges quickly: there is not much change after the 10th iteration, compared to the first 10. As I could not decide for any stopping criterion, and in order to maintain greater control of the duration of the estimation, I use a fixed number of iterations (typically 5) in the experiments.

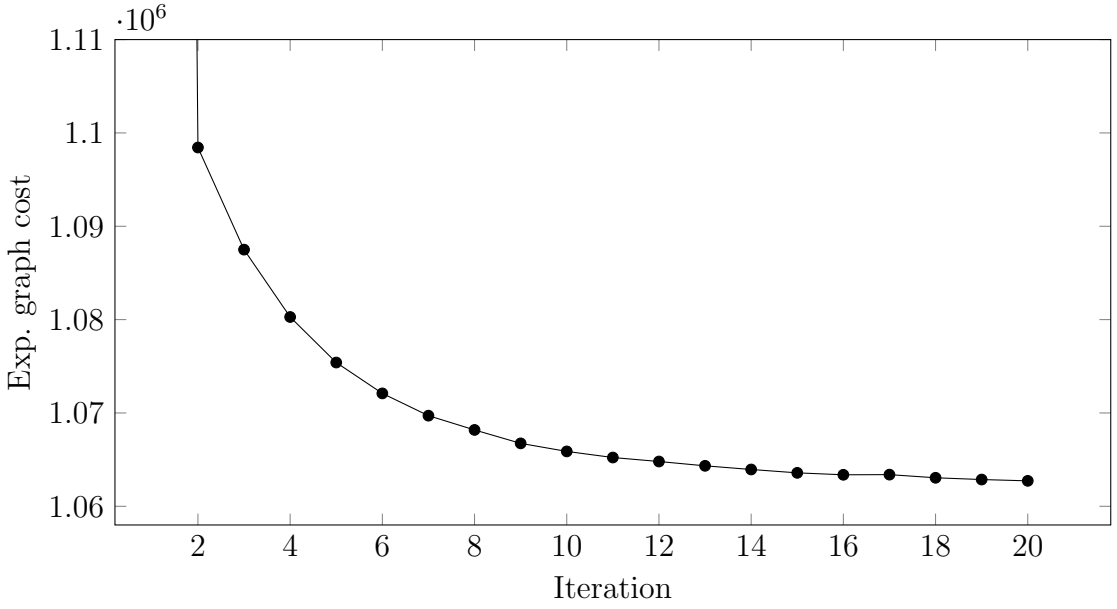

Figure 4.10: Convergence of MCEM fitting. The cost at the first iteration is far beyond the scale (around $1.6 \cdot 10^6$).

4.3.5 Model Selection

In addition to finding optimal parameters θ for a given rule set R, we can also attempt to optimize the rule set itself by deleting unnecessary rules. This would correspond to finding the following maximum:

$$\hat{R} = \arg\max_{R' \subseteq R} \int_{\Theta} \mathbb{E}_{E|V,R',\theta} \log P(V, E|R', \theta) d\theta \tag{4.34}$$

In [Sumalvico 2017], I proposed to search for the optimal rule set using simulated annealing [Kirkpatrick et al. 1983; Besag 2004]. For each rule, a score was computed based on the expected contribution of this rule to the graph log-likelihood. Those scores were used to propose the next rule set, so that rules with low scores had smaller chances of surviving to the next iteration. Then, the expected log-likelihood of the graph given the next rule set was computed and the rule set was either accepted or rejected.

Although this approach improved the evaluation results in [Sumalvico 2017], further experiments led to the conclusion that it is much too complicated for the benefit that it offers. Furthermore, its theoretical foundation is dubious: because each iteration requires a complete run of MCMC graph sampler, only a few iterations of annealing can be executed – typically up to 20, which is much too little to expect convergence of the annealing algorithm. Finally, the integration over Θ

is only tractable for the simple edge model and even there it is quite complicated to implement. Thus, I decided to abandon this approach.

Model selection can be also done in a simpler, although less theoretically sound way. To begin with, we replace the integral in (4.34) with a maximum. In this way, we can use regular fitting and sampling to evaluate a rule set. Furthermore, instead of attempting to search for the optimal rule set, we restrict ourselves to deletion of rules that appear useless after each sampling and fitting iteration. For this purpose, we define the score of a rule as follows:

$$score(r) = -\log \lambda_R P_{rule}(r) + \sum_{e=(v,v',r)\in\mathcal{E}} w_e \left[\log P_{root}(v'|\theta_{root}) - \log p_{edge}(v, v', r|\theta_{edge})\right] \tag{4.35}$$

The first term is a new model component: a *rule model*, defining a probability distribution over arbitrary rules. We use a very simple unigram model over pairs of input-output symbols in the subsequent transitions of the rule's FST. The term λ_R is due to the fact that we draw sets of rules from this distribution (see Sec. 4.1.2). The second term is the log-likelihood we gain by using the rule r to derive words, rather than having those words as roots. w_e is the sample weight of edge e. After each iteration, the rules with negative scores are deleted. The rule model introduces a cost of having an additional rule in the model, which has to be compensated by this rule's usefulness.

Figure 4.11 illustrates the outcomes of model selection after each iteration. The rule set is typically reduced to roughly half its original size. The edges labeled with the deleted rules are also deleted from $\mathcal{E}$. The number of possible edges decreases more slowly than the number of rules because the deleted rules are typically infrequent. However, the set $\mathcal{E}$ of possible edges is also substantially reduced. For most experiments, evaluation results are virtually unaffected by model selection, but the net benefit of applying it is a smaller model, and therefore faster computation times.

4.3.6 Finding Optimal Branchings

Previous sections described drawing large samples of branchings and using them to estimate model parameters. This corresponds to the variant of the Expectation Maximization algorithm often called 'soft EM', which utilizes full expected values. Another commonly used variant is 'hard EM' [Samdani et al. 2012], in which the inference would be based on a single optimal graph. Although I previously

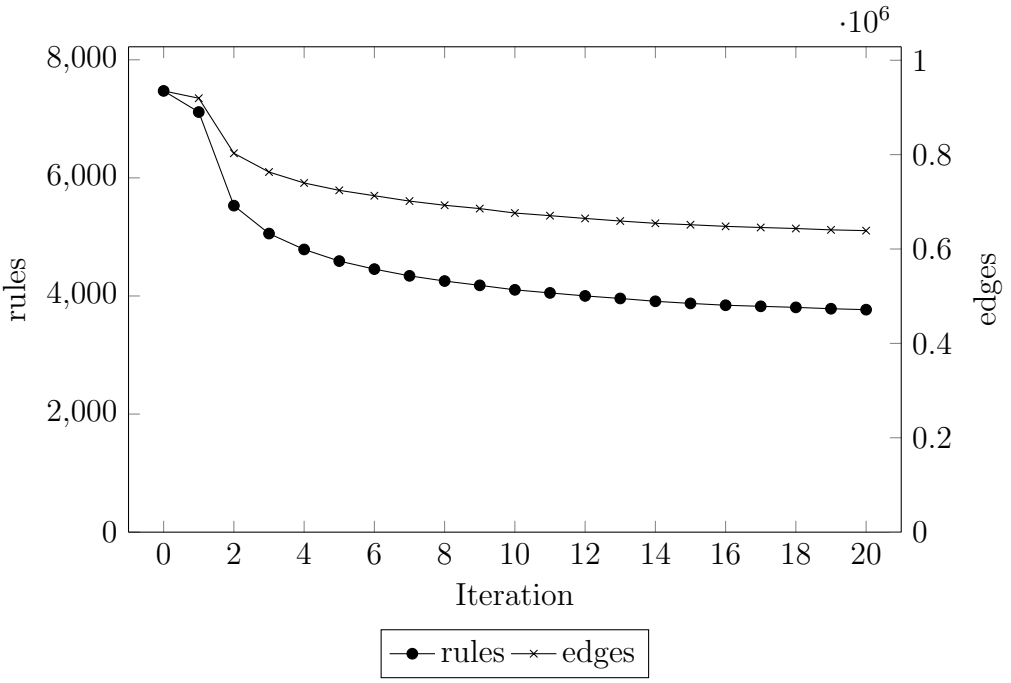

Figure 4.11: The number of rules and possible edges during model selection.

experimented with the hard-EM algorithm [Janicki 2015], I found that taking a single optimal graph is insufficient. Especially for rare rules, the difference between frequencies 0, 1 or 2 was very significant, but no intermediary values were possible. However, as the edge costs in the model are local and independent of each other, the optimum branching can be found using Chu-Liu-Edmonds algorithm [Tarjan 1977].

Chapter 5

Learning Inflectional Relations

In order to assess the practical usability of the model described in the previous chapter, I evaluate it on several different tasks related to inflection. Although the morphology learning method developed in this thesis is not especially targeted towards inflection – in fact, the evaluation results of the unsupervised experiments presented in this chapter suffer from the confusion of inflection and derivation, which is a consequence of Whole Word Morphology's theoretical assumptions – there are practical reasons for evaluating the quality of inflectional analysis.

The first and most important reason is that inflectional analysis is a well-defined task. In the most common formulation, it consists of assigning a lemma and an inflectional tag (a bundle of values of inflectional features, like case, number, person etc.) to words in text. This chapter concentrates especially on lemmatization and related tasks, like generating inflected forms from lemmas or identifying inflected forms of the same lemma. For such kind of problems, methods based on string transformations on word types seem appropriate. Inflectional tagging requires a different kind of approach: it is best done on the token basis, taking the local context of a token into account. A method targeting the tagging task in combination with morphology is presented in Chap. 6.

Another reason for an evaluation on inflection-related tasks is the availability of datasets. Lemmatized and tagged corpora are easily available. In contrast, there is no standardized task and annotation scheme for learning derivation, or inflection combined with derivation, especially in a relational formulation, i.e. focusing on relations between words, rather than internal word structure.[1] Finally,

[1]The tree annotation provided by the CELEX lexical database [Baayen et al. 1995] comes close to being the right annotation for this kind of problems. Unfortunately, it is only available for three closely related languages (English, German and Dutch), all exhibiting a fairly simple inflectional and derivational morphology.

lemmatization is the kind of morphological analysis most often used in downstream applications, like parsing, text mining etc. Learning derivation, while theoretically interesting, has much less practical relevance.

5.1 Datasets

Most experiments are conducted on seven different languages: Finnish, German, Latin, Latvian, Polish, Romanian and Russian. For those languages, suitable resources (described below) are easily obtainable.

Finding lemmatized and tagged corpora for a wide variety of languages in a uniform, easy-to-use format has become straightforward thanks to the Universal Dependencies project.[2] They provide corpora corpora in CoNLL format (a plain text format containing tab-separated columns) with multiple layers of annotation, up to a dependency tree. For the experiments conducted in this chapter, three kinds of annotation are especially relevant: lemma, tag according to the UPOS tagset and inflectional features. In order to utilize the UD corpora for tasks related to the learning of inflection, I made several simplifying assumptions, which do not always hold:

1. The coarse-grained tag is always the same for the lemma and the inflected form, as it should correspond to the part-of-speech, which is not changed by inflection. The reason why this assumption is important is because the morphology model needs tags for lemmas as well, but they are usually not provided by the datasets. This principle is violated e.g. in the German dataset, where *vorgesehen*$_{\text{ADJ}}$ is lemmatized as *vorsehen* (a verb infinitive), although it is tagged as an adjective. In general, participles are the most common source of such problems.

2. The lemma must itself be an existing word. While this assumption seems obvious, it does not hold for some datasets:

   - German – in ambiguous cases, all possible lemmas are provided, with no disambiguation from context. For example, *geboten* lemmatizes to *bieten|gebieten*.
   - Finnish – compound splitting is additionally provided for lemmas, e.g. *yliopiston* lemmatizes to *yli#opisto*.

[2] `www.universaldependencies.org`

- Latin – additional disambiguation information is provided for lemmas, e.g. *occides* lemmatizes to *occido#1*.

While the latter two cases seem to provide useful additional information, the proper conversion of the datasets would require some effort and expertise – e.g. on the first sight it is unclear whether the Finnish lemmas can *always* be converted to real words just by deleting the separator, or whether the distinction in Latin data is relevant to morphology or only semantic. In general, I decided against language-specific preprocessing and excluded those languages from lemmatization experiments.

3. A word that is lemma of other words is also its own lemma, i.e. lemmatization does not form chains or cycles. This is again violated in German, e.g. $vorgesehene_{\mathrm{ADJ}}$ is lemmatized to *vorgesehen*, but $vorgesehen_{\mathrm{ADJ}}$ to *vorsehen*. Other datasets contain similar cases.

As a source of plain, unannotated corpora employed for fully unsupervised training, I utilize the Leipzig Corpora Collection[3] [Biemann et al. 2007]. For the most languages, the standard-sized corpora of 1 million sentences coming from Wikipedia are used. Exceptions are Latin, where the full Wikipedia is used, and Latvian, where a 1 million sentence corpus of news crawled from the Internet is used instead. Both exceptions are due to small sizes of Wikipedias in those languages (both around 300k sentences).

5.2 Unsupervised Clustering of Inflected Forms

Given only an unannotated corpus or a list of words, the only kind of inflectional analysis that can be realistically expected is the clustering of inflected forms coming from the same lemma. As the choice of a certain inflectional form as citation form is more or less arbitrary (e.g. verb infinitive in many languages, but 1st person singular in Latin and Greek, 3rd person singular in Hungarian etc.), it is, in my opinion, not possible to establish reliable criteria allowing for automatic discovery of lemmas.[4]

[3] `corpora.uni-leipzig.de`

[4] Attempts have been made to determine the lemma based on, for example, the frequency of inflectional categories [Chan 2008] or efficiency of a grammar deriving inflected forms from a lemma [Albright 2002]. However, the former concentrates on identifying *some* base form, from which other forms can be derived, but which is not necessarily the lemma accepted in the language's grammar. The latter concentrates on theoretical linguistic aspects and does not provide

Experiment setup. In order to identify clusters of morphologically related words (which will mostly be different inflected forms of the same lemma), I employ the Chinese Whispers graph clustering algorithm [Biemann 2006].

As weights of the edges in the morphology graph, I use the expected edge frequencies determined during sampling. They have a useful property: the sum of weights of all incoming edges in a node, plus the expected frequency of that node being a root, equals one. In other words, the incoming edges are mutually exclusive events. This gives a straightforward probabilistic interpretation to the weight-adding done by Chinese Whispers: it corresponds to the event of the parent node of the considered node being in a certain cluster. Furthermore, the event of a node being a root is treated as a loop edge: it contributes its weight to the proposal of staying in the current cluster. In this way, nodes that have incoming edges can still remain singletons (which would have been impossible without loops).

The experiments were carried out in several different variants in order to assess the influence of various model configurations on the overall performance. All experiments employ the ALERGIA root model. The labels 'single' and 'neural' correspond to the edge model used, '-freq' to an additional frequency model and '-emb' to an additional word embedding model (word embeddings with 30 dimensions were trained with word2vec). The models used for embeddings were the ones based on neural networks, Gaussian models were left out of the evaluation.

All experiments consisted of 5 iterations of model selection followed by another 5 iterations of fitting. A single sampling step consisted of 100,000 warm-up iterations and 10,000,000 sampling iterations while training. The final sampling, which determines the expected frequencies of edges using the trained model, contained 100,000,000 sampling iterations. Apart from the variants using an embedding model, all other variants used a parameter $\alpha = e^{-2}$ to control the tree height (see Sec. 4.1.3). The variants with embeddings used no such functionality (i.e. $\alpha = 1$), as the tree height is sufficiently limited by the expensive, unpredictable changes in word embeddings.

Experiments for all languages were done using the same parameter values, which were mostly guessed before the experiments as reasonable defaults. I deliberately avoided the fine-tuning of parameters to each dataset separately.

Baselines. As I am not aware of any publicly available algorithm implementation solving exactly the same task, the usefulness of the approach developed here will be demonstrated through a comparison to simple baselines. The most naïve approach

empirical results on corpora.

possible is to place every word in a separate cluster, which is called 'bl-singleton'. The other baseline, 'bl-logfreq', consists of applying the Chinese Whispers algorithm to a morphology graph, in which the edges are weighted by the logarithm frequency of the rule. This roughly corresponds to the approach taken by Janicki [2013].[5]

Evaluation measure. As a clustering evaluation measure, I use BCubed [Amigó et al. 2009], which provides results in the usual form of precision, recall and F1-score. Note that the reason why 'bl-singleton' sometimes achieves apparently good results (especially for German) is that the gold standard contains many singleton clusters. For once, many rare words occur in the corpus only in one inflected form. Furthermore, there are many types, which do not correspond to real words of the language (e.g. numbers, foreign words, abbreviations etc.). I deliberately did not exclude such cases from the test data.

Results. The results are shown in Table 5.1. In general, no consistent improvement over the 'bl-logfreq' baseline can be observed. The results range from a significant improvement (Latvian, Russian) over slight improvement (German, Romanian), no difference (Polish) up to a slight decrease (Latin). On the other hand, the good performance of the singleton baseline for German is an exception, as for all other datasets this baseline performs poorly.

Most results suffer from low precision, meaning that the clusters tend to be too large. The main reason for this is the inability of the model to distinguish between inflection and derivation. Especially in cases where there is not enough inflected forms to form a stable cluster, words related through derivation are clustered together instead. However, in Latin, a language with particularly large inflectional paradigms, a contrary trend is observed: some paradigms are so large that they are split into several clusters. A further reason for low precision are accidental similarities between short words. The probabilistic model helps to rule out some of such patterns (especially through model selection), but not all of them. Such similarities can form 'bridges' between unrelated clusters, leading to their merging.

No significant difference can be observed between the simple and the neural

[5] Janicki [2013] contained an additional step consisting of clustering the outgoing edges of each node and splitting nodes according to edge clusters. This step greatly improved the results (especially precision). However, it is a feature of the clustering algorithm, which could be applied regardless of the weighting. The main point of the evaluation provided here is to compare the weights obtained in sampling to log-frequencies, so the details of the clustering algorithm are orthogonal to this comparison.

Language	Model	Prec.	Rec.	F1
German	bl-singleton	100.0 %	81.2 %	**89.6 %**
	bl-logfreq	70.5 %	97.4 %	81.8 %
	simple	78.6 %	95.9 %	86.4 %
	simple-freq	77.7 %	96.2 %	86.0 %
	neural	77.9 %	95.9 %	86.0 %
	simple-emb	70.8 %	97.0 %	81.9 %
	neural-emb	69.6 %	97.3 %	81.1 %
Latin	bl-singleton	100.0 %	27.9 %	43.7 %
	bl-logfreq	74.4 %	74.0 %	**74.2 %**
	simple	80.0 %	61.1 %	69.3 %
	simple-freq	79.1 %	61.7 %	69.3 %
	neural	80.3 %	61.2 %	69.5 %
	simple-emb	67.3 %	67.7 %	67.5 %
	neural-emb	68.5 %	67.4 %	68.0 %
Latvian	bl-singleton	100.0 %	48.2 %	65.0 %
	bl-logfreq	46.5 %	80.2 %	58.9 %
	simple	76.8 %	82.9 %	79.7 %
	simple-freq	76.2 %	82.6 %	79.3 %
	neural	77.1 %	83.5 %	**80.1 %**
	simple-emb	69.4 %	86.5 %	77.0 %
	neural-emb	69.2 %	87.0 %	77.1 %
Polish	bl-singleton	100.0 %	47.7 %	64.6 %
	bl-logfreq	76.0 %	86.0 %	80.7 %
	simple	81.2 %	80.4 %	**80.8 %**
	simple-freq	80.7 %	80.8 %	80.7 %
	neural	81.0 %	80.5 %	80.7 %
	simple-emb	80.6 %	80.8 %	80.7 %
	neural-emb	80.7 %	80.7 %	80.7 %
Romanian	bl-singleton	100.0 %	50.0 %	66.7 %
	bl-logfreq	73.0 %	92.0 %	81.4 %
	simple	79.5 %	87.8 %	83.4 %
	simple-freq	78.3 %	88.4 %	83.0 %
	neural	79.8 %	88.3 %	**83.8 %**
	simple-emb	71.5 %	90.0 %	79.7 %
	neural-emb	71.9 %	90.2 %	80.0 %
Russian	bl-singleton	100.0 %	34.7 %	51.6 %
	bl-logfreq	51.8 %	76.3 %	61.7 %
	simple	78.5 %	84.8 %	**81.5 %**
	simple-freq	76.4 %	85.2 %	80.6 %
	neural	78.2 %	84.8 %	81.4 %
	simple-emb	68.7 %	86.1 %	76.4 %
	neural-emb	70.3 %	85.8 %	77.3 %

Table 5.1: Results of unsupervised clustering.

edge model, as well as between the variant with and without a frequency model. The models with word embeddings perform slightly worse because of their lack of penalty on tree height – the assumption that using the embeddings as an additional feature can restrict the tree height equally well, does not hold.

5.3 Unsupervised Lemmatization

As justified in the previous section, I do not think that a completely unsupervised lemmatization is possible, as the choice of lemmas is to a certain degree arbitrary. However, an experiment with very little supervision is possible: in addition to an unlemmatized corpus, we provide the algorithm with a list of lemmas. The list is not necessarily complete: lemmas for some forms from the corpus may be missing. The task of the algorithm is to assign the inflected forms from the corpus to the correct lemma from the provided list (if the correct lemma is available), then to learn lemmatizing rules from those assignments and finally to apply the learnt rules to generate missing lemmas.

Experiment setup. I use whole Universal Dependencies corpora for both training and testing. For training, the list of unlemmatized types from the corpus is provided, either without part-of-speech tags or with coarse-grained tags. The list of lemmas provided additionally contains only words that themselves appear in the corpus as tokens, so a fair amount of lemmas is missing. For training, I use the 'left and right' restrictions described in Sec. 3.3.4: only edges leading from a lemma to an inflected form (i.e. a form appearing in the corpus which is not itself a lemma) are allowed. I use the ALERGIA root model and either simple or neural edge model. For the experiments with tagged data, the simple root tag model is used.

Back-formation. A trained model can generate missing lemmas through *back-formation*. The idea is that the root cost (i.e. the negative log-probability according to the distribution P_{root}) of an inflected form might be larger that the cost of a lemma plus the cost of an edge deriving the inflected form from the lemma. This is especially likely if POS tags are used: the training data contain mostly roots with tags characteristic to lemmas, so the distribution $P_{roottag}$ will attribute high probabilities to such tags and very low probabilities to tags characteristic to inflected forms. Therefore, for each inflected form that is supposed to be lemmatized, the learnt rules are applied to generate all possible lemmas. The cost of an edge

corresponding to each rule is computed. Furthermore, if the lemma is not present in the training vocabulary, its cost as a root is added to the cost of the analysis. Formally:

$$lemma(v) = \arg \min_{\substack{r \in R \\ v' \in r^{-1}(v)}} [-\ln p_{edge}(v', v, r|\theta_{edge}) - (1 - 1_V(v')) \ln P_{root}(v'|\theta_{root})] \tag{5.1}$$

$1_V(\cdot)$ is the indicator function of set V, i.e. 1 if the argument belongs to V and 0 otherwise. In case a tagged model is used, the last term must be additionally multiplied by $P_{roottag}(t'|v', \theta_{roottag})$.

Baseline. The baseline of this experiment consists of extracting candidate rules between lemmas and inflected forms and using the most frequent rule for each form to match it to a lemma. Inflected forms, for which no rules are extracted (e.g. because the true lemma is missing) are left unmatched. Using the most frequent rule is superior to using the rule with minimum edit distance, because the latter method would be prone to accidental word similarities (like substituting the first letter) which are not common patterns.

Results. The evaluation results are shown in Table 5.2. While the results on untagged datasets are mixed, on tagged datasets at least the simple edge model provides a consistent slight improvement (with the exception of Latvian, where the result is equal to the baseline). The neural edge model performs in general significantly worse. This is not surprising: in the unsupervised setting, where the training data are not reliable, a more flexible model runs more into the risk of overfitting to wrong examples.

5.4 Supervised Lemmatization

In the supervised lemmatization experiment, a lemmatized corpus is split into two parts: the training and testing data. The training corpus is used to learn a morphology model, which is then applied to lemmatize words from the testing corpus. For supervised training, we use the method described in Sec. 3.3.4.

For the experiments described in this section, 30% of the corpus (more exactly: the first 30% running tokens)[6] is taken as training data. From the remaining

[6]The split is deterministic on purpose. A random split would have made the evaluation much more expensive, because each experiment would have to be repeated multiple times. As the split

Language	Model	untagged	coarse
Latvian	baseline	47.1 %	46.4 %
	simple	39.8 %	46.4 %
	neural	42.7 %	44.8 %
Polish	baseline	46.4 %	44.7 %
	simple	41.7 %	48.4 %
	neural	45.2 %	43.8 %
Romanian	baseline	62.8 %	63.0 %
	simple	64.5 %	65.1 %
	neural	50.9 %	63.5 %
Russian	baseline	60.0 %	59.4 %
	simple	69.2 %	62.1 %
	neural	52.4 %	60.9 %

Table 5.2: Results of unsupervised lemmatization.

Language	Dataset	untagged	coarse	fine
Ancient Greek	training	25,615	28,254	22,993
	evaluation	22,520	24,190	26,104
Latvian	training	15,927	18,788	19,112
	evaluation	12,467	14,855	15,709
Polish	training	14,644	24,128	24,796
	evaluation	14,373	16,853	18,876
Romanian	training	19,018	25,964	26,248
	evaluation	12,973	17,293	17,902
Russian	training	74,012	85,301	91,574
	evaluation	52,162	59,688	70,055

Table 5.3: Size (number of types) of the datasets used for supervised lemmatization.

70%, we remove all types occurring in the training data, so that only lemmatization of unknown words is evaluated. We are not interested in the ability of the model to reproduce lemmatizations encountered in the training data, because those could be easily memorized. The resulting size of training and testing vocabularies is roughly equal, as shown in Table 5.3.

For each language, we conduct experiments with three different variants of POS tagging (untagged, coarse-grained, fine-grained). Additionally, each experiment is conducted in two variants ('–Lemmas' and '+Lemmas' in Table 5.4). In the latter, all lemmas needed to lemmatize the testing data are known in advance (i.e. added to the training vocabulary), so that the model does not need to gen-

is done on token basis and the whole experiment concerns types, there is no reason to expect a significant difference between the type lists of various splits – and thus an influence of the split on the results.

erate lemmas on its own, only choose the right one. The former case corresponds to 'normal' supervised training, i.e. nothing from the testing dataset is known in advance. In this case, the model attempts to generate the missing lemmas through back-formation. The model configurations used in the experiments are same as in the previous section.

Baseline. Supervised lemmatization can be formulated as a classification problem, as shown for example by Chrupała et al. [2008]. In this formulation, the class is the lemmatizing rule (or, equivalently, the rule deriving the inflected form from the lemma). The features used to predict the class are the word's POS tag and letter n-grams.

As the system described by Chrupała et al. [2008] was geared towards joint lemmatization and POS tagging, it would have been incompatible with my experiments: it operated on token basis and made use of local context features. Instead, I apply a heavily simplified version of this approach as baseline: I use a Maximum Entropy classifier[7] to predict the word's lemmatizing rule from the POS tag (if present) and n-grams at the beginning and at the end of the word of lengths from 1 to 5. In order to restrict the number of features to a manageable size, I use the 1000 most frequent n-grams.

Results. The type-based accuracy of various setups is shown in Table 5.4. Apart from the untagged case without knowing lemmas in advance, all setups clearly outperform the baseline. The former case shows that an unlabeled string form of a word is usually not enough for reliable lemmatization. The presence of POS-tags, even coarse-grained, improves the situation significantly.

As in other experiments, also here it turns out that the neural edge model offers no benefit over the simple edge model. This is unexpected, as the greater flexibility of the neural model ought to be an advantage especially in supervised learning, where tighter fitting of the model to the training examples is beneficial.

The results for Ancient Greek are a clear outlier. Perhaps the most important reason for this is the polytonic orthography, which, together with accent shifts in morphological paradigms, greatly complicates the inflection.[8] Moreover, the Ancient Greek verb morphology is complicated compared to other considered languages: the paradigms are large and many possible transformations are rare or not

[7] More specifically, I use the classifier `LogisticRegression` implemented in the Python package `scikit-learn`. This is identical to the baseline I used in [Janicki 2015].

[8] In addition to accent shifts in morphological paradigms, even the same word form can occur with different accent types depending on its context in the sentence [Smyth 1920, p. 37].

present in training data at all. Also suppletion seems to be more common than in the other languages. Finally, it seems to me that some irregularities are due to non-standard historical spellings: for example, the word Θρῄκης 'Thrace.GEN' is lemmatized as Θρᾴκη, instead of Θρῄκη postulated by the model. Wiktionary[9] explains that the former is the standard form and the latter is a 'poetic' variant. The annotators of the corpus seemingly decided to use standard spellings for lemmas in cases where inflected forms are spelled in a non-standard way. This is understandable from the point of view of corpus annotation, but it makes inflection appear more irregular than it actually is.

In the variant where the lemmas are not known in advance (the left half of the table), the results with fine-grained tags are significantly higher than for coarse-grained tags. It turns out that coarse-grained tags are not enough to lemmatize properly as inflectional affixes may be ambiguous. For example, the Polish masculine animate noun *kot* 'cat' has following forms: *kot* 'cat-NOM.SG', *kot**a*** 'cat-GEN.SG', *kot**y*** 'cat-NOM.PL'. Compare this to the paradigm of *lina* 'rope', which is a feminine noun: *lin**a*** 'rope-NOM.SG', *lin**y*** 'rope-GEN.SG', *lin* 'rope-GEN.PL'. Thus, only knowing that the word is a noun ending in *-a* does not provide enough clue to determine the lemma.

The right half of the table shows the results in the case where all lemmas are known in advance. The results illustrate an important difference between my model and the baseline classifier: there is no straightforward way to make the classifier benefit from knowing the lemmas in advance. On the other hand, my model directly benefits from it by changing the costs computed by (5.1) accordingly, which has a high impact on the performance. Thus, the graph-based model provides an easy way to boost up the accuracy of the lemmatizer by feeding it with additional lexical resources.

5.5 Supervised Inflected Form Generation

Inflected form generation is the reverse of lemmatization: given a lemma and desired tag, for example (`machen<VERB>`, `<VERB><PART>`), the model is supposed to produce the suitable inflected form, in this case `gemacht`. As the desired inflected form has to be exactly specified, only fine-grained tags make sense in this context. Then, the learnt rules are used to generate all possible inflections from the given lemmas.

[9] `https://en.wiktionary.org/wiki/%CE%98%CF%81%E1%BF%84%CE%BA%CE%B7` accessed on Dec 11, 2018.

Language	Model	–Lemmas			+Lemmas		
		untagged	coarse	fine	untagged	coarse	fine
Ancient Greek	baseline	26.2 %	30.9 %	35.5 %	26.2 %	31.0 %	35.4 %
	simple	24.0 %	33.2 %	35.6 %	36.2 %	41.7 %	39.2 %
	neural	23.1 %	25.4 %	32.0 %	35.0 %	40.1 %	39.0 %
Latvian	baseline	52.1 %	61.5 %	71.7 %	52.3 %	61.3 %	72.0 %
	simple	45.0 %	60.7 %	77.2 %	76.8 %	83.7 %	80.3 %
	neural	47.7 %	56.8 %	74.8 %	76.6 %	83.4 %	80.1 %
Polish	baseline	54.0 %	60.9 %	69.2 %	54.1 %	60.9 %	69.1 %
	simple	55.1 %	69.9 %	79.2 %	81.6 %	87.7 %	82.7 %
	neural	57.3 %	63.1 %	77.1 %	82.3 %	87.5 %	82.7 %
Romanian	baseline	57.7 %	70.7 %	74.3 %	58.4 %	70.4 %	74.1 %
	simple	61.4 %	74.0 %	82.7 %	83.1 %	92.5 %	90.6 %
	neural	64.2 %	72.1 %	81.9 %	83.7 %	92.3 %	90.6 %
Russian	baseline	60.2 %	65.7 %	74.9 %	60.1 %	65.7 %	75.0 %
	simple	58.5 %	72.0 %	79.1 %	80.4 %	87.5 %	84.9 %
	neural	57.9 %	69.6 %	78.6 %	80.9 %	87.5 %	84.6 %

Table 5.4: Results of supervised lemmatization.

The obtained list is filtered to include only the entries matching the desired POS tag.

The baseline is similar to the one used in the previous experiment: a maximum entropy classifier utilizing prefix and suffix character n-grams and POS tag. The difference is that the n grams are now extracted from the lemma, while the tag used for classification is the tag of the desired word.

Results The results of the experiment are shown in Table 5.5. Because of the problems described in the previous section, Ancient Greek was excluded from this experiment. The results show a slight advantage of my model (especially with the simple edge model) over the baseline, with the exception of Russian. Bearing in mind that the Russian corpus is significantly larger than others, this difference might be a characteristic of the size of the training data rather than language. Nevertheless, the results of all methods are similar.

Note that while a classifier has to be trained for each task separately, the graph-based models used to solve this task are exactly the same as those used for lemmatization. It is satisfactory to see that a general model of 'morphological competence' achieves similar, or even slightly better scores, than baseline models dedicated to a single task.

Language	**Model**	**Accuracy**
Latvian	baseline	70.9 %
	simple	73.4 %
	neural	72.4 %
Romanian	baseline	85.8 %
	simple	88.9 %
	neural	84.1 %
Polish	baseline	80.1 %
	simple	82.8 %
	neural	81.5 %
Russian	baseline	87.0 %
	simple	85.4 %
	neural	84.9 %

Table 5.5: Results of supervised inflected form generation.

Chapter 6

Semi-Supervised Learning of POS Tagging

Part-of-speech tagging is nowadays commonly thought of as a solved problem, with accuracy scores easily achieving 95% or more. However, such results are typically reported for English or other resource-rich European languages, for which large amounts of high-quality training data are available. Those languages also tend to have a simple morphology and utilize small to mid-sized tagsets. However, for many other languages, the reality is different: training data are expensive or not available, more fine-grained tagsets are needed and complex morphology accounts for large numbers of OOV words in corpora. Straka and Straková [2017] present a contemporary evaluation of state-of-the-art POS tagging for a very wide variety of languages. The scores for tagging with UPOS[1] tagset lie below 90% for many languages. The lack of sufficient amounts of training data is undoubtedly one of the main reasons for such results.

In this chapter, I attempt to improve the POS tagging in a setting where only a small amount of labeled training data is available, as well as a significantly larger corpus of unlabeled text. I train a bigram Hidden Markov Model on the labeled part of the corpus and subsequently apply Baum-Welch estimation on the unlabeled part. Additionally, I use the labeled corpus to learn morphological rules, which are then applied to guess the possible tags of the words from the unlabeled part. For the latter step, I modify the graph sampler developed in Sec. 4.3.2 to treat the unknown tags as latent features which influence edge probabilities.

The choice of a bigram HMM for tagging is clearly suboptimal. However,

[1]The tagset used in the Universal Dependencies project. It is very coarse-grained, containing e.g. only a single tag for nouns and verbs, respectively.

improving the state-of-the-art in tagging is beyond the scope of this thesis. The evaluation presented in this chapter merely intends to show that tagging performance can benefit from guessing tags for unknown words based on morphological criteria. The choice of an HMM is dictated by the fact that it is easy to implement, fairly simple and well interpretable in its workings, possible to train on unlabeled data (using the Baum-Welch algorithm) and that it is possible to extend an existing model with new vocabulary without re-training. Furthermore, the closely related trigram HMMs used to be state-of-the-art for a long time and are still used in popular tools like HunPos [Halácsy et al. 2007; Megyesi 2009].

A similar problem was recently approached by Faruqui et al. [2016]. They employ a label propagation algorithm on a graph of words. However, their approach differs in many details from the one presented here. Firstly, the edges in the graph do not necessarily correspond to morphological transformations, but rather to a more general concept of 'morphological relatedness', which may also involve such relations as sharing a common affix. Secondly, the inflectional tags are decomposed into single features which are treated independently. While such approach allows for more powerful generalizations, it may also lead to inconsistent labels (like marking a noun for tense), which have to be corrected in a separate post-processing step.

In Sec. 6.1, I introduce the idea in an informal fashion, based on examples. The method is formalized in Sec. 6.2. Finally, I present empirical evaluation results in Sec. 6.3. The material presented in this chapter has not been published before.

6.1 A General Idea

6.1.1 Intrinsic and Extrinsic Tag Guessing

How can we guess possible tags for a word before seeing its occurrences in a text? In general, there are two possible answers to this question. *Intrinsically*, we can try to recognize parts of the word as known inflectional or derivational affixes, which would be characteristic for a certain part-of-speech. Or maybe some words just 'sound like' belonging to a certain category, although it is hard to explain why. On the other hand, an *extrinsic* approach would take into account the presence or absence of certain other, morphologically related words.

It is easy to find examples showing that a purely intrinsic approach is insufficient in many cases. For example, consider the following German words: *Fichten* 'spruce.N.PL', *richten* 'judge.V.INF', *rechten* 'right.ADJ.NOM.PL.DEF'.[2] Phonet-

[2]All cited words are ambiguous in their inflectional form. The glosses shown here are picked

ically and orthographically they are very similar and all include an inflectional suffix *-en*, which is highly ambiguous in German. The knowledge that German nouns are always capitalized does not provide much of a clue, because words belonging to any other part-of-speech may also occur capitalized. Even worse, many further similar words are ambiguous in their meaning and part-of-speech, e.g. *Dichten* ('density.N.PL' or capitalized 'dense.ADJ.NOM.PL.DEF' or 'compose (e.g. a poem).V.INF'), *schlichten* ('simple.ADJ.NOM.PL.DEF' or 'mediate.V.INF').

The situation becomes diametrically different if we take into account morphologically related words. For example, we might observe words like *richtet* or *richtete*, which together with *richten* look unambiguously like a verb paradigm. Similarly, the occurrence of a form like *rechtes* can convince us that *rechten* is an adjective, because verbs do not take the suffix *-es*. For ambiguous forms, we will likely find parts of different paradigms, for example *schlichtet* (verb) and *schlichtes* (adjective), which will allow us to notice the ambiguity. Of course, in order to conduct such analysis, we have to know which affix configurations are characteristic for which part of speech. This is the part that we are going to learn from labeled data.

6.1.2 Applying Tagged Rules to Untagged Words

Let us assume that we have learned a morphology model on tagged data. Now we are presented with a new set of words, possibly containing many words not present in the original training set. In this section, we will show how the trained model can be applied to derive guesses for tags in the new vocabulary. The approach follows the idea sketched in the previous section: the tag of the word will be determined by the neighboring words, together with the knowledge about the morphology contained in tagged rules.

To illustrate the approach with a minimal example, let us assume that our tagset consists of only three tags: NN, VVINF and VVFIN, and that the untagged vocabulary consists of the German words *machen*, *mache*, *macht*. We compute the edge probabilities for every edge that is possible according to the model, under every tagging. For example, the model might determine that the following three edges are possible (i.e. the corresponding tagged rules are included in the model)

---

as examples.

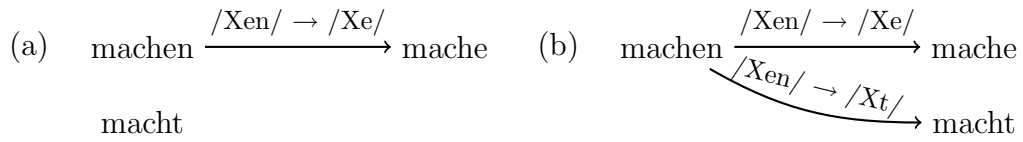

Figure 6.1: Two possible morphology graphs corresponding to the words *machen*, *mache*, *macht*. What does each of them tell us about the possible tags of those words according to (6.1)?

and give following probabilities for them:

$$\begin{aligned} p_{edge}(\text{machen}_{\text{NN}}, \text{mache}_{\text{NN}}, /X\text{en}/_{\text{NN}} \rightarrow /X\text{e}/_{\text{NN}}) &= 0.3 \\ p_{edge}(\text{machen}_{\text{VVINF}}, \text{mache}_{\text{VVFIN}}, /X\text{en}/_{\text{VVINF}} \rightarrow /X\text{e}/_{\text{VVFIN}}) &= 0.01 \\ p_{edge}(\text{machen}_{\text{VVINF}}, \text{macht}_{\text{VVFIN}}, /X\text{en}/_{\text{VVINF}} \rightarrow /X\text{t}/_{\text{VVFIN}}) &= 0.2 \end{aligned} \quad (6.1)$$

Using those values, we can reason about the possible taggings based on an untagged graph. Consider the two graphs shown in Fig. 6.1. What does each of them say about the possible taggings?

Graph 6.1a is consistent with either $\{machen_{\text{NN}}, mache_{\text{NN}}\}$ or $\{machen_{\text{VVINF}}, mache_{\text{VVFIN}}\}$, since the only edge in this graph is possible with both labelings. Note that the edge containing noun labels has much higher probability, so this graph suggests a strong preference for the noun hypothesis. It does not say anything about the possible tags of *macht*. On the other hand, the only tagging consistent with the graph 6.1b is $\{machen_{\text{VVINF}}, mache_{\text{VVFIN}}, macht_{\text{VVFIN}}\}$, since the edge between *machen* and *macht* is only possible if *machen* is a verb infinitive. It is important to notice that adding an edge between *machen* and *macht* diametrically changed the possible taggings for *mache*, although it is not touched by the edge in question. This illustrates how the graph model captures dependencies between the tags across a whole paradigm, although the edge probabilities are local.

6.2 The Method

Recall from Sec. 4.1.4 that a morphology model trained on tagged data defines a probability distribution $P(V, T, E|R, \theta)$ over tagged graphs, consisting of words V, edges E and tag assignments $T = \{T_v : v \in V\}$, given the rules R and model parameters θ. The tags of root nodes are predicted by a distribution $P_{roottag}(t|v, \theta_{roottag})$, which takes into account the string form of the word, v, and distribution parameters $\theta_{roottag}$.

Let $\tau_{v,t}$ denote the probability of word v having a tag t, which is supposed to be predicted by our morphology model. Obviously, $\forall_v \sum_t \tau_{v,t} = 1$. Furthermore, let $u(\cdot)$ be a 'rule untagging' function, which converts tagged rules into untagged ones. For example, $u(/X\text{en}/_{\text{NN}} \rightarrow /X\text{e}/_{\text{NN}}) = /X\text{en}/ \rightarrow /X\text{e}/$. Formally:

$$v' \in (u(r))(v) \leftrightarrow \exists_{t,t' \in \mathcal{T}} (v', t') \in r(v, t) \tag{6.2}$$

Given an untagged vocabulary V, a set of (tagged) rules R and a tagset $\mathcal{T}$, we derive the set of possible untagged edges $\mathcal{E}$ as follows:

$$\mathcal{E} = \{(v, v', u(r)) : \{v, v'\} \subseteq V \wedge r \in R \wedge \exists_{t,t' \in \mathcal{T}} (v', t') \in r(v, t)\} \tag{6.3}$$

That is, $\mathcal{E}$ contains edges between words from V obtainable by assigning to them some tags from $\mathcal{T}$ and applying a rule from R. Summing over all possible tag assignments $T \in \mathcal{T}^V$ and all possible edge sets $E \subseteq \mathcal{E}$, we can compute $\tau_{v,t}$ as follows:

$$\begin{aligned}
\tau_{v,t} &= \sum_{T,E} \frac{P(V,T,E|R,\theta)}{P(V|R,\theta)} \delta(T_v, t) && (6.4)\\
&= \sum_{E} \left[\frac{P(V,E|R,\theta)}{P(V|R,\theta)} \sum_{T} \frac{P(V,T,E|R,\theta)}{P(V,E|R,\theta)} \delta(T_v, t) \right] && (6.5)\\
&= \mathbb{E}_{E|V,R,\theta} \left[\mathbb{E}_{T|V,E,R,\theta} \delta(T_v, t) \right] && (6.6)
\end{aligned}$$

In the above formula, T_v denotes the tag of the word v in a concrete graph and $\delta(\cdot,\cdot)$ denotes the Kronecker delta. Thus, the computation involves taking two expectations: over the possible graph structures and over the taggings in a fixed graph. Then we simply count the possible graphs and taggings, in which the word v has tag t.

The inner expectation can be computed exactly by a variant of Forward-Backward algorithm introduced in Sec. 6.2.1. In order to approximate the outer expectation, we will use Markov Chain Monte Carlo sampling over untagged graphs as developed in Sec. 4.3.2. However, the sampling algorithm will need some modifications, as the edge probabilities will no longer be independent of the graph structure. I describe those modifications in Sec. 6.2.2. Finally, the computed values of $\tau_{v,t}$ will be fed to an already pre-trained HMM to provide it with guesses for the tags of unknown words, before it is reestimated on untagged text. I describe this procedure in detail in Sec. 6.2.3.

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow v_9$$

Figure 6.2: The Forward-Backward computation for a linear sequence in an HMM. $\alpha_{v_6,t} = P(v_1, \ldots, v_6, T_6 = t)$, whereas $\beta_{v_6,t} = P(v_7, v_8, v_9 | T_6 = t)$.

6.2.1 The Forward-Backward Algorithm for Trees

In order to compute $\tau_{v,t} = \mathbb{E}_{T|V,E,R,\theta}\delta(T_v, t)$ for a fixed graph (V, E), let us recall the well-known Forward-Backward algorithm used for Hidden Markov Models.[3] The HMMs employed for POS tagging operate on sentences, which are linear sequences of words (Fig. 6.2). The summing over all possible tag sequences is tackled by introducing the so-called *forward probability* (usually written as α) and *backward probability* (β), which are defined as follows:

$$\alpha_{v_i,t} = P(v_1, \ldots, v_i, T_i = t) \tag{6.7}$$

$$\beta_{v_i,t} = P(v_{i+1}, \ldots, v_n | T_i = t) \tag{6.8}$$

It is easy to see that the product of both, $\alpha_{v_i,t}\beta_{v_i,t}$, gives us the probability of the whole sequence *and* the node v_i having tag t. This can be used to derive $\tau_{v_i,t}$:

$$\tau_{v_i,t} = \frac{\alpha_{v_i,t}\beta_{v_i,t}}{\sum_{t'} \alpha_{v_i,t'}\beta_{v_i,t'}} \tag{6.9}$$

The point of forward-backward computation is that, due to the Markov property of HMMs, the forward and backward probability can be computed with recursive formulas, thus avoiding the combinatorial explosion caused by summing over all possible tag sequences:

$$\alpha_{v_i,t} = \sum_{t'} \alpha_{v_{i-1},t'} P(t|t') P(v_i|t) \tag{6.10}$$

$$\beta_{v_i,t} = \sum_{t'} P(t'|t) P(v_{i+1}|t') \beta_{v_{i+1},t'} \tag{6.11}$$

The general idea of forward-backward computation can be extended beyond linear sequences, which are a special case of trees, to arbitrary trees.[4] In this case, the backward probability of a node is the probability of the subtree rooted in it,

[3] See for example Manning and Schütze [1999] or Jelinek [1997] for an introduction to HMMs and the Forward-Backward algorithm.

[4] I was unable to find any publications describing a generalization of Forward-Backward computation to tree models. I do not believe that this is my innovation, though. I would be grateful for any hints on this topic from reviewers and readers.

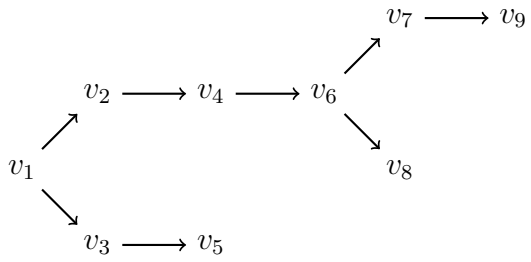

Figure 6.3: The Forward-Backward computation for a tree. Also here, $\alpha_{v_6,t} = P(v_1, \ldots, v_6, T_6 = t)$ and $\beta_{v_6,t} = P(v_7, v_8, v_9 | T_6 = t)$.

given a tag, while the forward probability is the probability of the rest of the tree *and* the tag (Fig. 6.3). Note that the forward probability involves not only the path leading from the root to the node in question (v_1, v_2, v_4, v_6 in Fig. 6.3), but also all side branches sprouting from this path (v_3, v_5).

In order to derive recursive formulas similar to (6.10)–(6.11) for the tree case, let us introduce a concept of *transition matrix* in this case. A transition matrix $T^{(v,v',r)}$ associated with an untagged edge (v, v', r) is a matrix corresponding to edge probabilities for every possible tagging of the source and target node. More specifically:

$$T^{(v,v',u(r))}_{t,t'} = \frac{p_{edge}(v, t, v', t', r)}{1 - p_{edge}(v, t, v', t', r)} \tag{6.12}$$

Continuing the example from 6.1.2, the probabilities in (6.1) yield the following transition matrix:

$$T^{(\text{machen,mache,}/X\text{en}/\rightarrow/X\text{e}/)} = \begin{array}{c} \\ \text{NN} \\ \text{VVINF} \\ \text{VVFIN} \end{array} \begin{array}{c} \begin{array}{ccc} \text{NN} & \text{VVINF} & \text{VVFIN} \end{array} \\ \begin{pmatrix} \frac{0.3}{1-0.3} & 0 & 0 \\ 0 & 0 & \frac{0.01}{1-0.01} \\ 0 & 0 & 0 \end{pmatrix} \end{array} \tag{6.13}$$

Furthermore, let $\lambda_{v,t}$ be the probability that the node v with tag t is a leaf, i.e. it has no outgoing edges. This value can be computed as follows:[5]

$$\lambda_{v,t} = \prod_{r \in R} \prod_{(v',t') \in r(v,t)} [1 - p_{edge}(v, t, v', t', r)] \tag{6.14}$$

[5]Two things can be said about the λ-values: they are extremely expensive to compute, because they involve a product over all hypothetical edges, also those leading to non-existent words, and they are of virtually no importance, since they tend to differ only slightly from tag to tag. As the terms of the product are mostly numbers very close to 1, although they are many, the result is still going to be fairly close to 1. Thus, although I include this value in the formulas for the sake of soundness of the theory, I actually ignored it in experiments, setting $\lambda_{v,t} = 1$ everywhere.

Now we can turn to computing the backward probability. A trivial observation is that $\beta_{v,t} = \lambda_{v,t}$ for leaf nodes. For a non-leaf node v, the backward probability will be equal to the product of backward probabilities of all children of v, multiplied by the probability of all outgoing edges of v. This has to be summed over all possible taggings of the child nodes. For example, taking the node v_6 in Fig. 6.3, we would have:

$$\begin{aligned}\beta_{v_6,t} &= \lambda_{v_6,t} \sum_{t_7} \sum_{t_8} T^{(v_6,v_7,r)}_{t,t_7} \beta_{v_7,t_7} T^{(v_6,v_8,r')}_{t,t_8} \beta_{v_8,t_8} \\ &= \lambda_{v_6,t} (\sum_{t_7} T^{(v_6,v_7,r)}_{t,t_7} \beta_{v_7,t_7}) (\sum_{t_8} T^{(v_6,v_8,r')}_{t,t_8} \beta_{v_8,t_8})\end{aligned}$$

The term $\lambda_{v_6,t}$ is due to the fact that v_6 contains no further outgoing edges apart from the two mentioned explicitly. The elements of the transition matrix contain 'one minus edge probability' in the denominator in order to remove this term from the product introduced by λ for edges that are present. The second line is due to a simple transformation: $\sum_i \sum_j a_i b_j = (\sum_i a_i)(\sum_j b_j)$. r and r' are simply the rules corresponding to the respective edges.

Using matrix and vector notation, let β_v and λ_v be $|\mathcal{T}|$-dimensional vectors. Further, let $out_{\mathcal{G}}(v)$ be the set of outgoing edges of v in graph $\mathcal{G}$ (which is our current graph). A general formula for the backward probability can be expressed as follows:[6]

$$\beta_v = \lambda_v * \prod_{(v,v',r) \in out_{\mathcal{G}}(v)} T^{(v,v',r)} \beta_{v'} \tag{6.15}$$

The vague idea for computing the forward probability is to take the forward probability of the parent node and multiply it by the probability of the edge leading to the node in question. However, the parent node might also have other children, which are not included in its forward probability. Looking at Fig. 6.3, the forward probability of v_2 must involve not only the forward probability of v_1 and the edge leading from v_1 to v_2, but also the subtree rooted in v_3. Thus, the general formula is as follows:

$$\alpha_v = \prod_{(v',v,r) \in in_{\mathcal{G}}(v)} \left[\alpha_{v'} * \prod_{\substack{(v',v'',r') \in out_{\mathcal{G}}(v') \\ v'' \neq v}} T^{(v',v'',r')} \beta_{v''} \right] \cdot T^{(v',v,r)} \tag{6.16}$$

$in_{\mathcal{G}}(v)$ denotes the set of incoming edges of v. Note that the set notation

[6] In the matrix formulas, the asterisk denotes element-wise multiplication and the dot or no symbol denotes dot product.

and the outer product is only for notational convenience, as the set is always a singleton. In this case, it means: 'pick v', that is the parent node of v'. The inner product goes over all children of v' except for v and includes the edges leading to them and the probabilities of the subtrees rooted in them. Finally, the last product corresponds to the edge leading from v' to v.

The last remaining issue is the forward probability of root nodes. It is simply equal to the probability of the root node defined by the model, which we call $\rho_{v,t}$ and compute as follows:

$$\rho_{v,t} = P_{root}(v|\theta_{root})P_{roottag}(t|v,\theta_{roottag}) \tag{6.17}$$

Thus, $\alpha_v = \rho_v$ for root nodes.

6.2.2 Modifications to the Sampling Algorithm

As we have seen in the analysis of Fig. 6.1, adding an edge typically has consequences for the whole subtree, in which the edge is added. The values τ_v for all nodes in the subtree may change, which in turn changes the probability of all edges in the subtree. This behavior constitutes a significant difference compared to the general sampling algorithm, in which the edge probabilities were independent of the graph structure and the cost of a change could be easily computed from the cost of added and removed edges (cf. 4.32). Nevertheless, we can apply the same sampler as the one developed in Sec. 4.3.2.

Observe that for every node v, the value $\sum_t \alpha_{v,t}\beta_{v,t}$ is the probability of whole subtree, to which v belongs. This property – being able to compute the probability of a whole subgraph using the values of a single node – is crucial in evaluating the sampler moves.

Adding or removing a single edge. Consider the graph in Fig. 6.4, to which the edge (v, v', r) is supposed to be added. Without this edge, we have two separate trees with a total probability expressed by $(\sum_t \alpha_{v,t}\beta_{v,t})(\sum_t \alpha_{v',t}\beta_{v',t})$. After adding this edge, we obtain a single tree. As v obtains a new child node, β_v will change. Let β'_v denote the new value, which can be computed as follows:

$$\beta'_v = \beta_v * T^{(v,v',r)}\beta_{v'} \tag{6.18}$$

Note that neither α_v nor $\beta_{v'}$ is affected by adding this edge. The probability of the new tree is simply $\sum_t \alpha_{v,t}\beta'_{v,t}$. If the move is accepted, the β values of all nodes on

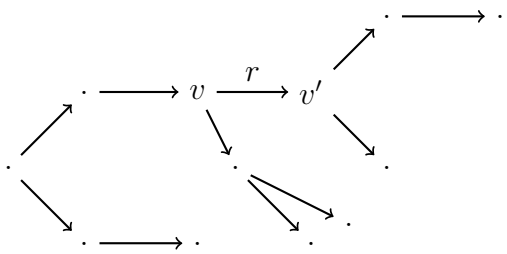

Figure 6.4: Adding or removing the edge (v, v', r).

the path from the root to v have to be updated, as well as the α values of all nodes except for this path.

Deleting an edge involves a very similar computation. In this case, the probability of the graph before deletion is $\sum_t \alpha_{v,t}\beta_{v,t}$, whereas the probability after deletion is $(\sum_t \alpha_{v,t}\beta'_{v,t})(\sum_t \rho_{v',t}\beta_{v,t})$. Here, $\beta'_{v,t}$ is the updated backward probability of v excluding the deleted edge.

Other moves. When exchanging a single edge to another one with the same target node, we already need to be careful, as two distinct cases arise: either the change takes place within one tree, or it involves two separate trees (Fig. 6.5). If we proceeded as in the previous paragraph, those cases would require different formulas. Instead of conducting such a detailed analysis of the changes, we apply a more general (and, admittedly, more computationally expensive) approach, that covers the 'flip' moves as well.

First, we group all edges that are to be changed (added or deleted) according to the tree, to which they belong (more specifically, according to the root of the tree, to which the edge's source node currently belongs). In each tree, we look for a minimum subtree that contains all the changes (Fig. 6.6). We build a copy of this subtree with all changes applied and recompute the forward probability for its root and the backward probabilities for the whole subtree. Finally, we use the (newly computed) forward and backward probability of the subtree root to determine the probability of the whole tree after changes.

Note that the 'flip' move can change the root of a tree (Fig. 6.7). It is thus important to correctly recognize the root of the modified tree and to recompute its forward probability, as it changes is this case.

A note on numerical issues. As the computation of acceptance probabilities now involves probabilities of whole trees, the numbers are typically very small. Furthermore, the occurrence of addition and dot product in formulas means that

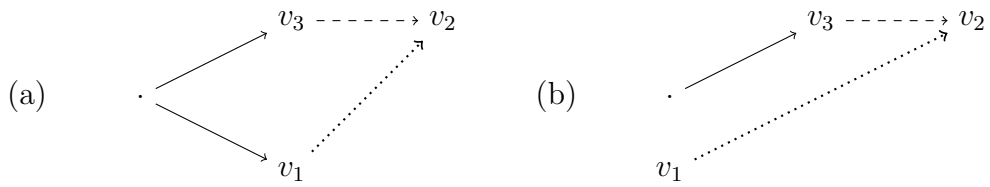

Figure 6.5: Exchanging an edge to another with the same target node. The change can take place within one tree (a) or involve two separate trees (b).

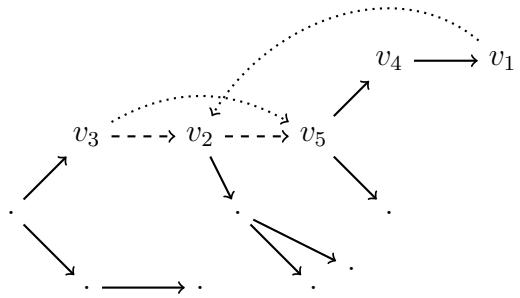

Figure 6.6: In case of a 'flip' move, the smallest subtree containing all changes is the one rooted in v_3. The deleted edges are dashed, while the newly added edges are dotted. In order to obtain the new β_{v_3}, we recompute the backward probabilities in the whole subtree. α_{v_3} is not affected by the changes. (The node labels are consistent with the definition in Fig. 4.8.)

Figure 6.7: A special case of the 'flip' move, which changes the root of the tree. (a) – the tree before the move (the edge to delete is dashed, while the edge to be added is dotted); (b) – the tree after the move. The node labels agree with the ones in Fig. 4.8, with $v_4 = v_2$, $v_5 = v_1$ and v_3 not existing. The difference between the two variants of 'flip' is neutralized in this case.

the operations cannot be carried out on log-probabilities instead. This means that at some point (with sufficiently large trees) the floating point computations will eventually result in zeros, although in theory the tree probability is nonzero.

If a probability of the proposed graph is zero, it does not pose much of a problem: such graph simply will not be accepted, as the acceptance probability will be zero. A problem arises if the probability of the current graph turns out to be zero. In this case, we obtain a zero in the denominator of the formula for acceptance probability (see Algorithm 4.1) regardless of the proposed tree. This situation can arise when multiplying the probabilities of two separate trees. Even though probabilities of both trees are nonzero (otherwise they would not have been accepted), their product may be zero, i.e. smaller than the machine epsilon.[7] To solve this problem, we introduce an artificial bound on the probability of trees: trees with a total probability smaller than some ϵ are never accepted. ϵ is chosen to be at least the square root of the machine epsilon, so the product of probabilities of two trees is still always greater than 0.

6.2.3 Extending an HMM with New Vocabulary

The guessed tag probabilities $\tau_{v,t}$ are fed to a Hidden Markov Model before re-estimating its parameters on untagged text. Recall that an HMM possesses three kinds of parameters: *initial probabilities* (probability of starting in a given state), *transition probabilities* (probability of a state given the previous state) and *emission probabilities* (probability of emitting a certain output symbol given the current state). Only the emission probabilities are affected by the extension of the vocabulary.

The emission probabilities in an HMM are the probabilities[8] $P(w|t)$ of a word w given a tag t. First, we convert the values in an already trained HMM to probability of tag given a word, i.e. $P(t|w)$, using the Bayes formula:

$$P(t|w) = \frac{P(w|t)P(t)}{\sum_{t'} P(w|t')P(t')} \tag{6.19}$$

[7]The *machine epsilon* is the smallest number greater than zero that can be represented as a floating-point number at a given precision. When using double precision, like the type `float64` in the NumPy library, this number equals $5 \cdot 10^{-324}$.

[8]This presentation takes a shortcut by calling HMM parameters directly *probabilities*. I assumed that the reader is already familiar with HMMs and statistical modeling in general, so no confusion should arise. However, such simplifications should be used with caution, especially when introducing statistical models, because they can obscure the understanding of what probability actually is.

The probability $P(t)$ of seeing a tag t at a random position in the corpus can be computed from tag frequencies during the fitting of the HMM on labeled training data. In this way, we obtain the probability of a tag given word, $P(t|w)$, for the words known to the tagger. For newly added words, we use the guesses obtained from the morphology component, i.e. $\tau_{w,t}$. As especially the extrinsic guessing approach often yields sparse vectors with many zeros, we apply an additional smoothing step using a parameter α.

$$P(t|w) := (1 - |\mathcal{T}|\alpha)\tau_{w,t} + \alpha \tag{6.20}$$

In this way, the guesses are used to guide the tagger in a certain direction, but it still has the opportunity to apply different taggings if the context motivates it. Furthermore, zero sentence probabilities can be avoided this way. I set α to 10^{-10} in the experiments, so it has negligible influence.

Finally, we apply the Bayes formula again to obtain the new emission probabilities:

$$P(w|t) = \frac{P(t|w)}{\sum_{w'} P(t|w')} \tag{6.21}$$

Note that the terms $P(w)$ are missing in the latter formula. This is because they are simply word frequencies, which are independent of the tagging. Thus, it does not influence the HMM estimation if we initially assume that all words have equal frequency. During estimation, the values will be automatically fitted to reflect the word frequencies in the corpus.

6.3 Evaluation

6.3.1 Experiment Setup

I conducted evaluation experiments for 9 languages: German, Finnish, Gothic, Ancient Greek, Latin, Latvian, Polish, Romanian and Russian, using the Universal Dependencies corpora. I chose languages that displayed sufficient morphological complexity for the approach to make sense (e.g. English is excluded for this reason) and for which sufficiently large corpora were available. The chosen languages exhibit diversity in genetic classification (Finnic, Germanic, Slavic, Italic, Baltic), complexity of morphology (from simple fusional in German, through complex fusional in Russian and Latin, to complex agglutinating in Finnish) and alphabet (Latin, Cyryllic, Greek). The datasets vary also in size, as shown in Tables 6.2 and

1	Fitting on the training set.
2	**1** + estimation on the development set.
3	**2** + extension of the vocabulary with random initialization.
4	**2** + extension of the vocabulary with intrinsic tag guesses.
5	**2** + extension of the vocabulary with extrinsic tag guesses.
6	**2** + extension of the vocabulary with gold standard tag guesses.

Table 6.1: Different setups of the HMM tagger used in the tagging experiment.

6.3.

Each corpus is randomly split into training, development and testing dataset in proportions 10%:80%:10%. An HMM tagger is fitted to the (labeled) training data using ML estimation. The training dataset is also used to learn tagged morphological rules. Then, we remove the labels from the development corpus and re-estimate the HMM on this corpus using Baum-Welch algorithm. This step is performed in several configurations: either with or without vocabulary expansion. In the latter case, all types occurring in the development corpus, but not known to the HMM (i.e. not occurring in the training corpus) are added to the vocabulary before the estimation. Finally, the tagging accuracy is assessed on the testing dataset. The details of the possible configurations are shown in Table 6.1.

For each corpus, two kinds of datasets are prepared: with *coarse-grained* and *fine-grained* tags. In the former case, the UPOS tagset is used, which amounts to around 15 tags for every language. In the latter, all inflectional information provided by the corpus annotation is additionally included. For example, in the Latin corpus, we have tokens like `beati<ADJ>` in the coarse-grained and `beati<ADJ><NOM><POS><MASC><PLUR>` in the fine-grained case.

The extrinsic tag guessing approach developed in the previous section is compared to intrinsic guesses. Those are provided directly by the distribution $P_{roottag}$, which, in this case, uses a character-based recurrent neural network to predict the word's tag from its string form (see Sec. 4.2.3).

6.3.2 Evaluation Measures

Two kinds of evaluation are performed: *lexicon* and *tagging* evaluation. In the first case, the quality of tag guesses for unknown words, τ_v, is measured directly. It is desirable for those values to not only predict the correct tag for unambiguous words, but also to handle ambiguity correctly, which means providing probabilities that correspond to the expected frequency of a word with the certain tag. We derive the gold standard data from the labels in the development set using the

following formula:

$$\hat{\tau}_{v,t} = \frac{n_{v,t}}{\sum_{t'} n_{v,t'}} \tag{6.22}$$

with $n_{v,t}$ being the number of occurrences of word v with tag t in the development set. This way, true ambiguities (with roughly equal frequency of different taggings) are treated differently than rare taggings, which may result from tagging errors or some obscure, infrequent meanings. The accuracy is computed as follows:

$$accuracy = \frac{1}{|V|} \sum_{v \in V} \sum_{t} \min\{\tau_{v,t}, \hat{\tau}_{v,t}\} \tag{6.23}$$

It is intentionally a very demanding measure: it achieves 100% for a given word only if the probability mass is distributed exactly according to the corpus frequency of the tagging variants, which is virtually impossible for ambiguous words. Hence, low scores according to this measure are not surprising and do not necessarily represent a bad-quality tagging. I decided for this measure, because it is easier to interpret than e.g. KL-divergence.

In the tagging evaluation, we evaluate the impact of providing tag guesses on a real POS-tagging task. The evaluation measure used there is the standard accuracy, i.e. the percentage of correctly tagged tokens.

6.3.3 Results

Tables 6.2 and 6.3 show the results of the lexicon evaluation, as well as some information about the sizes of the datasets. The first three columns show the number of distinct tags found in the training set, the size (number of types) of the training set and the size of the development set. Note that the sizes for the same language in both tables differ slightly: firstly, fine-grained tags would yield more types from the same corpus because of more possible taggings per word. Secondly, the tables describe distinct experiments with different random splits of the data. Thus, the datasets in Table 6.3 are typically slightly larger, but the reverse case is also possible.

The results show clearly that the extrinsic method outperforms the intrinsic in predicting possible tags for a given word type. Probably the most important reason for that is that the intrinsic method always makes unsharp predictions – it never attributes the whole probability mass to a single tag. On the other hand, the extrinsic method often makes unambiguous predictions because of the absence of rules allowing for alternatives (a phenomenon illustrated in Fig. 6.1). Thus, the

Language	**#tags**	**#train**	**#devel**	**Intrinsic**	**Extrinsic**
Ancient Greek	14	7.7k	34k	60.7 %	66.0 %
Finnish	15	10k	49k	50.7 %	64.3 %
German	16	10k	45k	61.6 %	67.6 %
Gothic	13	2.1k	7.6k	48.3 %	**66.1 %**
Latin	14	7k	26k	54.4 %	**70.4 %**
Latvian	17	5.1k	23k	50.6 %	64.6 %
Polish	15	5.6k	28k	58.7 %	69.5 %
Romanian	16	7.8k	30k	54.0 %	**70.5 %**
Russian	16	30k	110k	68.2 %	76.9 %

Table 6.2: Lexicon evaluation with coarse-grained tags.

Language	**#tags**	**#train**	**#devel**	**Intrinsic**	**Extrinsic**
Ancient Greek	481	8.2k	34k	22.9 %	30.0 %
Finnish	906	10k	48k	25.5 %	**41.1 %**
German	519	11k	45k	15.4 %	21.0 %
Gothic	333	2k	7.7k	15.7 %	30.8 %
Latin	632	7.5k	26k	23.6 %	**44.8 %**
Latvian	501	5.1k	23k	24.0 %	36.8 %
Polish	394	5.9k	28k	19.6 %	31.5 %
Romanian	185	7.7k	30k	24.1 %	30.1 %
Russian	597	33k	110k	33.0 %	**52.2 %**

Table 6.3: Lexicon evaluation with fine-grained tags.

Language	**1**	**2**	**3**	**4**	**5**	**6**
Ancient Greek	66.9 %	74.2 %	73.2 %	78.9 %	79.1 %	85.7 %
Finnish	60.6 %	74.4 %	72.8 %	79.5 %	**82.3 %**	83.8 %
German	77.1 %	75.5 %	83.1 %	84.9 %	85.1 %	83.6 %
Gothic	76.8 %	77.0 %	80.3 %	81.9 %	82.8 %	86.5 %
Latin	74.3 %	80.8 %	81.6 %	82.5 %	**86.6 %**	85.6 %
Latvian	67.0 %	72.3 %	73.1 %	79.6 %	80.3 %	84.8 %
Polish	71.5 %	78.7 %	74.5 %	78.1 %	**84.2 %**	82.9 %
Romanian	78.5 %	84.6 %	85.3 %	88.0 %	88.4 %	87.0 %
Russian	80.9 %	87.7 %	87.7 %	90.3 %	90.5 %	91.4 %

Table 6.4: Tagging evaluation with coarse-grained tags.

latter achieves better scores especially on correctly tagged unambiguous words.

The comparison of tagging accuracies is shown in Tables 6.4 and 6.5. The columns correspond to the tagger configurations explained in Table 6.1. In general, a rise of the score from left to right is to be expected.

The difference between columns 1 and 2 illustrates the influence of reestimating the trained model on unlabeled data without adding the OOV words to the vocabulary. Interestingly, this results in an improvement in case of the coarse-grained tagset, but in a decline when using the fine-grained tagset, both significant. However, adding the OOV words from the development set to the vocabulary, even with randomly initialized probabilities (column 3), further improves the accuracy (with a few exceptions), so that the result is consistently better than column 1. Column 4 introduces intrinsic tag guessing as initial probabilities for newly added words, rather than random values. This results in a further improvement, especially significant for Finnish, Ancient Greek and Latvian (both coarse-grained and fine-grained).

The most important comparison in this evaluation is between column 4 and 5. This illustrates the benefit of using extrinsic tag guessing (column 5), rather than intrinsic. This results in a consistent improvement, ranging from very slight to significant. The most significant improvements are shown in bold. Finally, column 6 displays what one might expect to be the upper bound on the accuracy: the one that would be achieved if tags were guessed perfectly (i.e. as $\hat{\tau}_v$). Surprisingly, it is not always the highest value in a row. It looks as if taking into account some wrong taggings during Baum-Welch estimation could accidentally improve the estimation, because the wrong tag might *also* have occurred in the given context. This seems especially plausible for cases like common and proper nouns, which are often confused. However, any speculation in such a case is not of much value.

Language	1	2	3	4	5	6
Ancient Greek	56.6 %	46.4 %	56.9 %	62.8 %	63.8 %	69.9 %
Finnish	53.5 %	37.7 %	56.7 %	65.1 %	**67.5 %**	71.7 %
German	59.4 %	49.8 %	58.7 %	61.8 %	62.0 %	63.1 %
Gothic	63.6 %	55.0 %	65.9 %	67.7 %	67.8 %	73.9 %
Latin	59.0 %	49.3 %	61.7 %	66.0 %	**68.7 %**	73.4 %
Latvian	58.5 %	47.1 %	59.4 %	65.7 %	66.8 %	71.3 %
Polish	55.4 %	43.7 %	57.5 %	62.5 %	62.7 %	67.9 %
Romanian	71.2 %	71.3 %	75.9 %	76.4 %	76.1 %	82.4 %
Russian	73.1 %	65.4 %	73.6 %	78.0 %	78.9 %	81.3 %

Table 6.5: Tagging evaluation with fine-grained tags.

Although the results speak consistently in favor of using extrinsic tag guessing, as well as using tag guessing at all, the benefits are somewhat less clear than I expected. Especially in the case of fine-grained tags, my expectation was that, due to the discrete nature of morphological rules, at least the tags of unambiguous words would be identified mostly correctly. This was supposed to greatly improve the Baum-Welch estimation, as instead of considering many hundred possible tags, the correct one is already known, which turns the estimation into almost supervised learning. However, I underestimated the impact of the small size of training corpus on the morphology component. Most fine-grained tags are very rare, so many morphological rules related to such forms are not learnt. As the set of possible tags is finite, this problem could perhaps be less significant if larger corpora were available for both training and development.

6.3.4 Remarks

The main limitation of the approach presented here is that it fails to capture regularities that do not change the surface form, like e.g. the relationship between the German verb infinitive and 3rd person plural, which always correspond to the same surface form.[9] The untagged graph contains one node per surface form, so edges preserving the surface form would have to be loops, which are not possible in the current model. Any further work on this approach should address this problem.

The task presented here is a neat example of the usefulness of treating inflection and derivation uniformly, as WWM does. Both inflectional and derivational regularities can provide clues about the word's lexical category and inflectional

[9]Or a large portion of English verb inflection. This is another reason why this approach currently does not make sense for a language like English.

form.

Perhaps the forward-backward computation on morphology graphs presented in this chapter could be generalized to predicting other latent word features that are affected by morphological rules in a regular way, or applied in a different graph model.

An adjustment of the approach presented here to fully unsupervised learning of morphological paradigms is also imaginable. In this variant, we would start with random transition matrices and an untrained root tag distribution and fit both using an expectation-maximization approach based on sampled frequencies. This would be similar to the approach taken by Chan [2006], but with an improved model of morphological transformations.[10] On the other hand, it would almost certainly constitute an improvement over my early attempts to learn morphological paradigms [Janicki 2013]. However, some problems remain to be solved: for example, in the semi-supervised case, the transition matrices are sparse. In the fully unsupervised case, they would be dense, which would make the computations prohibitive for large numbers of tags. A small number of tags would however fail to distinguish different inflectional forms, which in inflected languages, like Polish or Latin, behave very differently in the graph representation (i.e. take edges labeled with different rules). Thus, the fully unsupervised case remains a topic for further work.

[10] Chan [2006] uses a very simple model, in which every word form is derived from the base via a single suffix change

Chapter 7

Unsupervised Vocabulary Expansion

One of the signs of mastering the morphology of a given language is the ability to utter or anticipate words that one has never heard before. In an unsupervised learning setting, this corresponds to taking a list of words as input and suggesting further words as output. Recent works call this task *vocabulary expansion* [Rasooli et al. 2014; Varjokallio and Klakow 2016].[1] The practical applications of vocabulary expansion include especially tasks like speech recognition or optical character recognition (OCR), in which words listed in some lexicon have to be recognized in a noisy representation.

As an evaluation task for unsupervised learning of morphology, vocabulary expansion has a huge advantage: it is theory-agnostic. Instead of discovering some underlying representation that has to be consistent with a certain linguistic theory, like segmentation or lemmatization, we predict the mere existence of words. By using out-of-vocabulary rate reduction as evaluation measure, we can replace the question of the existence of a word by the question of its occurrence in a certain corpus, thus further simplifying the evaluation. In this way, we can use unannotated corpora as both training and evaluation data. The linguistic agnosticism makes this task suitable for comparing different approaches to morphology, e.g. segmentation vs. word-based.

[1] This should not be confused with *lexicon expansion*, which usually means predicting some features for new words, more in the spirit of Chap. 6. In Janicki [2015], I erroneously used the term *lexicon expansion* while referring to *vocabulary expansion*.

7.1 Related Work

Chan [2008] observed that inflected forms of a lexeme have a frequency distribution exhibiting the general tendency of Zipf's law (although there are not enough data points to establish the Zipf's law in a stricter sense): a few forms tend to be common, while many more are rare. For middle and low frequency lexemes, this means that the rare forms are expected to be missing, especially in smaller corpora. This phenomenon creates a large number of words that will typically not be observed in the training corpus, while they can be anticipated from observed words with knowledge of the language's morphology.

As an evaluation task for unsupervised learning of morphology, vocabulary expansion has been used already by Neuvel and Fulop [2002] for their Whole Word Morphology model. Similarly to the present work, they needed an evaluation measure that was agnostic of the underlying model of morphology, unlike segmentation. More recently, vocabulary expansion was approached as a separate task by Rasooli et al. [2014]. They apply a state-of-the-art unsupervised morphological segmentation method (Morfessor) and subsequently induce a finite grammar (implemented as an FST) from the obtained segmentations. This line of research was further explored by Varjokallio and Klakow [2016], who train language models (N-gram and RNN) on the segmentations and use them to sample new words. Both approaches are evaluated by measuring the reduction of out-of-vocabulary (OOV) word rate on a separate corpus.

My own preliminary work on the graph-based morphology model also involved evaluation through vocabulary expansion [Janicki 2015; Sumalvico 2017]. I reported precision at various sizes of generated vocabulary, measured against complete lists of correct words in a given language. As such lists are difficult to find and rarely really complete, OOV rate reduction seems more suited as an evaluation measure and will be used in this chapter.

7.2 The Method

Given a vocabulary V and a morphology model $\langle R, \theta \rangle$ trained on this vocabulary, we can compile two transducers: T_V, accepting all words from the lexicon, and T_R, being a disjunction of all known morphological rules. Then, we can apply the following transformation in order to compute T_N, an automaton that accepts new words:

$$T_N = \pi_{out}(T_V \circ T_R) \setminus T_V \tag{7.1}$$

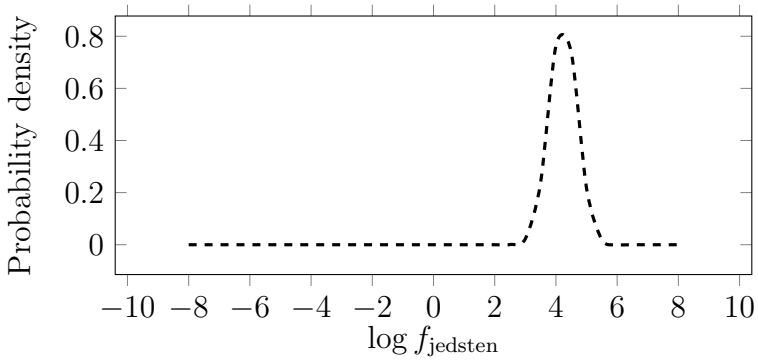

Figure 7.1: Predicted log-frequency of the hypothetical word **jedsten*, as derived from *jedes* via the rule /*X*es/ → /*X*sten/. The frequency model predicts the mean log-frequency $\mu = 4.24$ (corresponding to the frequency around 69) with standard deviation $\sigma = 0.457$. The probability of the word not occurring in the corpus (i.e. frequency < 1) corresponds to the area under the curve on the interval $(-\infty, 0)$, which is approximately 10^{-20}.

Subsequently, we analyze all words produced by T_N by computing possible edges deriving them from known words. For each edge, we compute its cost using the trained model. Finally, we compute the cost of adding the word to the vocabulary from the edge costs.

Additionally, we can use the frequency model developed in Sec. 4.2.4 to predict the frequency of newly generated words. This can be included in the edge costs: the generated words should have low frequencies, so that their absence from the training corpus is well-motivated. Sec. 7.2.1 explains this aspect in more detail, while Sec. 7.2.2 shows how word costs are computed from edge costs.

7.2.1 Predicting Word Frequency

A word frequency model, such as the one developed in Sec. 4.2.4, may be helpful in the task of predicting unseen words. The words that we aim at lie in a certain frequency range: they are too infrequent to be observed in the training corpus, but frequent enough to be expected in the (usually larger) development corpus.

Modeling word frequency is expected to help avoid a common mistake: applying highly productive rules to very frequent words. For example, the German adjective *jeder* 'each' displays a typical adjective paradigm: *jeder*, *jede*, *jedes*, *jeden*, *jedem* etc. However, comparative or superlative forms, like **jedere* or **jedsten*, do not exist. Using the frequency model, we can easily conclude that such forms indeed do not exist, instead of just happening not to occur in the corpus. Consider

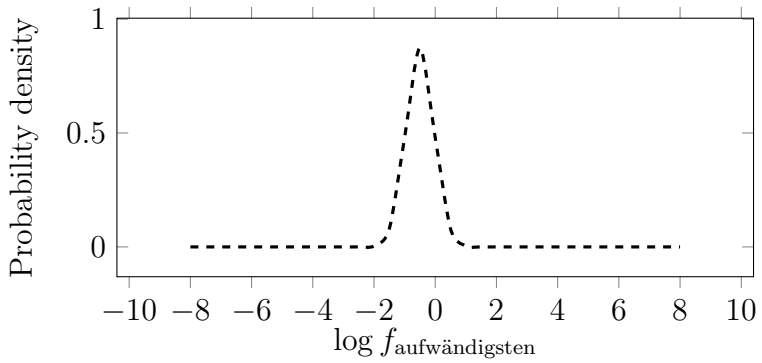

Figure 7.2: Predicted log-frequency of *aufwändigsten*, as derived from *aufwändiges* via the rule /*X*es/ → /*X*sten/. The probability of log-frequency being negative (i.e. the frequency being < 1) is here around 0.863 ($\mu = -0.5, \sigma = 0.457$).

deriving the word **jedsten* from *jedes* using the rule /*X*es/ → /*X*sten/. *jedes* has a corpus frequency of 114 and the mean difference of log-frequencies between source and target word fitted for the rule is 0.5. This yields the predicted mean frequency of $\exp(\ln 114 - 0.5) \approx 69$ (Fig. 7.1). Hence, if the word existed, it would have to be observed.

A different situation arises if the base word is rare. Consider applying the same rule to *aufwändiges*, which occurs only once in the training corpus. The resulting *aufwändigsten* has a predicted mean frequency of $\exp(\ln 1 - 0.5) \approx 0.6$, so it is quite likely to be missing from the corpus (Fig. 7.2). Admittedly, this method does not always work: there are many rare adjectives that cannot form superlative as well. However, in this way we are able to filter out at least some false positives with small chances of losing a true positive.

When considering the occurrence of a word in another corpus, rather than its existence, we might want to modify the integration interval. Instead of computing the probability of corpus frequency being smaller than 1, which amounts to integrating over the interval $(-\infty, \ln 1)$, we set the minimum frequency as well. For example integrating over the interval $(\ln 0.1, \ln 1)$ gives us the probability, that the word is not observed in the given training corpus, but will be observed in a ten times larger corpus. This is the approach that I took in the evaluation experiments.

7.2.2 Computing Word Costs from Edge Probabilities

We will now turn to computing the *cost* of adding a new word v to the vocabulary. A usual way of deriving a new word is applying a known and productive rule to an

already known word. However, especially in the absence of POS tags, this approach is likely to overgenerate massively: we apply a rule every time the pattern on its left-hand side is matched, regardless of the part-of-speech and inflectional form of the source word. In this way, we quickly end up adding for instance verb suffixes to nouns. In the worst case, the pattern on the left-hand side of the rule is just $/X/$, which results in applying the rule to every known word.

Recall from Sec. 6.1.1 that several possible base words give a much better indication of the part-of-speech and paradigm type of the derived word than just one. As in Whole Word Morphology there is no concept of a 'base word' or 'lemma', we are likely to find many different rules, by which a proposed new word can be derived from various known words. In the remainder of this section, I present a method of computing costs for new words that takes into account all possible ways of deriving a word. Each possible derivation lowers the cost of the resulting word. Thus, the more known words 'support' a proposed new word, the cheaper it becomes.

Formally, the cost of adding a new word v to the vocabulary equals the following log-likelihood change:

$$cost(v) = -\log \frac{P(V \cup \{v\}|R)}{P(V|R)} = -\log \frac{\sum_E P(V \cup \{v\}, E|R)}{\sum_E P(V, E|R)} \tag{7.2}$$

The sum goes over all edges that can be created using words from the given vocabulary and rules from R.

Let p_0 denote the root probability of the newly added word v and $p_1, \ldots, p_n$ denote the probabilities of edges that derive v from some word in V using some rule in R. It is easy to see that the probability of each graph not containing v (i.e. the graphs considered in the denominator of (7.2)) will contain the term $\prod_{i=1}^{n}(1-p_i)$: the considered edges are possible (the source node is in V and the deriving rule is in R), but not present, since the node v is not present. Let us draw this term in front of the sum and call the rest of the sum Z:

$$\sum_E P(V, E|R) = \prod_{i=1}^{n}(1-p_i)Z \tag{7.3}$$

Z contains the probabilities of edges not involving v, summed over all possible graph structures.

Now let us turn to the graphs containing v, which form the numerator of (7.2). In order to make the computation tractable, we will consider only graphs,

in which v is a leaf node. In this case, it is either a root node, or is derived by one of the edges. Note that for each graph not containing v, there is exactly one graph containing v as a root (simply the same graph, plus v as a root). Furthermore, each graph containing v as a root, contains the term: $p_0 \prod_{i=1}^{n}(1-p_i)$. On the other hand, for each graph not containing v and each possible edge deriving v, there is exactly one graph containing this extra edge. If v is derived with the k-th edge, the graph probability contains the term: $\prod_{i=1}^{k-1}(1-p_i)p_k \prod_{i=k+1}^{n}(1-p_i) = \frac{p_k}{1-p_k}\prod_{i=1}^{n}(1-p_i)$.

In result, by considering in the numerator only graphs, in which v is a leaf, we obtain the following approximation for (7.2):

$$cost(v) = -\log \frac{\left(p_0 + \sum_{j=1}^{n} \frac{p_j}{1-p_j}\right)\prod_{i=1}^{n}(1-p_i)Z}{\prod_{i=1}^{n}(1-p_i)Z} = -\log\left(p_0 + \sum_{j=1}^{n}\frac{p_j}{1-p_j}\right) \quad (7.4)$$

Thus, the cost of adding a word can be obtained by simply summing the probabilities of different ways of deriving it, and taking the negative logarithm of the result. In practice, p_0 is usually several orders of magnitude smaller than the edge probabilities, so it can be safely omitted.

7.3 Evaluation

7.3.1 Experiment Setup

Four different configurations are evaluated wrt. the methods presented in this chapter. On one hand, using as many edges as possible ('COMBINED') to compute the cost of a new word in the way presented in Sec. 7.2.2 is compared to using just the edge with the highest probability ('SINGLE'). Both cases are further split in two by either using a frequency model ('-FREQ') or omitting it. In case the frequency model is used, the interval for the frequency of proposed words is set to $[0.1, 2]$. The model components used are the ALERGIA root model (Sec. 4.2.1) and the simple edge model (Sec. 4.2.2). All models were trained by 5 iterations of fitting, without model selection. In order to speed up the computation, rules with $-\log p > 8$ are discarded and the maximum cost of generated words is set to 6. This results in around 2 to 5 million unique new words, depending on the dataset and model variant.

Baseline. The present approach is compared to a simplified version of the method presented by Varjokallio and Klakow [2016]: segmenting the training corpus with

Language	Training corpus		Development corpus			
	Types	Tokens	Types	Tokens	Token OOV	Type OOV
German	218,043	1,935,822	972,596	17,430,771	8.03 %	84.74 %
Finnish	276,798	1,430,189	1,225,944	12,850,115	13.82 %	84.49 %
Latin	77,777	411,148	321,336	3,671,912	13.63 %	82.46 %
Latvian	169,876	1,855,796	598,446	16,690,119	5.41 %	77.80 %
Polish	214,856	1,730,214	788,680	15,569,684	7.80 %	78.88 %
Romanian	176,539	2,262,719	622,247	20,377,801	4.69 %	77.69 %
Russian	231,818	1,756,566	889,958	15,852,287	8.52 %	80.33 %

Table 7.1: The datasets used for evaluation.

Morfessor [Virpioja et al. 2013] and training an RNN language model on the morph sequences using the RNNLM toolkit [Mikolov et al. 2010; Mikolov 2012]. While the authors found that a linear interpolation of an RNN and an n-gram language model yielded slightly better results, the difference was rather minor, as illustrated by Figure 1 in Varjokallio and Klakow's paper. The setup used for both tools is based on the original paper: the parameter α for Morfessor is set to 0.8 and the RNN model uses 50 classes and a hidden layer of size 20. I sampled 100 million words from the RNN model, which, depending on the language, yielded between 5.4 and 7.7 million unique new words.

Datasets. As in Section 5.2, I used the corpora from the Leipzig Corpora Collection for the experiments. The size of the corpora was 1 million sentences, except for the Latin dataset, which was smaller (around 243k sentences). The corpora were randomly split into training and development datasets in proportion 1:9. Table 7.1 presents some statistics about the vocabulary of the resulting datasets.

Evaluation measure. The evaluation measure used in the experiments is the *out-of-vocabulary rate reduction*: it measures, how much the rate of out-of-vocabulary words decreases after vocabulary expansion. If OOV_{base} denotes the OOV rate before expansion (one of the two last columns of Table 7.1) and OOV_{exp} the OOV rate after expansion, the reduction is given by the following formula:

$$OOV_{reduction} = \frac{OOV_{base} - OOV_{exp}}{OOV_{base}} \tag{7.5}$$

7.3.2 Results

Tables 7.2 and 7.3 present the OOV reduction rates for various numbers of generated words. Two sizes are particularly interesting: 200k, which corresponds roughly

Language	Model	10k	50k	100k	200k	500k	1M
German	COMBINED+FREQ	1.85%	4.75%	7.99%	11.41%	17.48%	21.97%
	COMBINED	1.88%	5.45%	8.13%	11.78%	17.23%	21.50%
	SINGLE+FREQ	0.86%	3.66%	5.82%	8.83%	15.49%	20.87%
	SINGLE	1.19%	4.11%	6.94%	10.59%	15.52%	19.65%
	Morfessor+RNNLM	0.21%	1.30%	2.54%	4.54%	8.68%	11.98%
Finnish	COMBINED+FREQ	2.31%	6.86%	9.89%	14.84%	22.04%	28.45%
	COMBINED	2.39%	6.95%	10.32%	15.61%	23.25%	29.39%
	SINGLE+FREQ	0.90%	3.52%	6.30%	10.24%	17.94%	25.92%
	SINGLE	1.05%	4.20%	6.89%	11.73%	19.66%	26.65%
	Morfessor+RNNLM	0.53%	2.29%	3.91%	6.27%	10.75%	14.58%
Latin	COMBINED+FREQ	7.67%	18.75%	25.70%	33.66%	42.56%	47.56%
	COMBINED	7.27%	19.43%	26.21%	33.39%	42.94%	47.77%
	SINGLE+FREQ	3.17%	13.21%	20.67%	30.35%	41.08%	46.71%
	SINGLE	4.49%	14.84%	21.80%	30.55%	41.12%	46.93%
	Morfessor+RNNLM	0.82%	3.82%	6.70%	10.84%	19.02%	25.23%
Latvian	COMBINED+FREQ	5.81%	16.30%	23.55%	32.14%	43.35%	50.97%
	COMBINED	5.62%	15.80%	22.89%	31.03%	42.84%	50.72%
	SINGLE+FREQ	2.39%	9.89%	16.96%	26.75%	40.24%	49.35%
	SINGLE	2.60%	10.86%	17.96%	26.67%	39.76%	48.50%
	Morfessor+RNNLM	0.49%	2.19%	4.40%	8.30%	16.41%	23.69%
Polish	COMBINED+FREQ	4.69%	13.27%	19.18%	26.77%	37.55%	44.24%
	COMBINED	4.58%	13.07%	19.20%	26.56%	37.17%	44.32%
	SINGLE+FREQ	1.97%	8.94%	14.50%	22.09%	34.93%	42.88%
	SINGLE	2.38%	9.41%	14.67%	21.91%	34.25%	42.97%
	Morfessor+RNNLM	0.27%	1.24%	2.70%	5.57%	12.06%	19.40%
Romanian	COMBINED+FREQ	4.25%	11.96%	17.87%	24.36%	32.15%	37.80%
	COMBINED	4.25%	12.36%	17.67%	23.66%	32.28%	38.09%
	SINGLE+FREQ	2.11%	8.63%	14.20%	21.23%	30.10%	35.94%
	SINGLE	2.67%	10.07%	14.50%	20.57%	29.88%	36.09%
	Morfessor+RNNLM	0.62%	2.56%	4.38%	7.11%	12.47%	17.32%
Russian	COMBINED+FREQ	4.39%	12.93%	18.90%	26.00%	36.65%	43.77%
	COMBINED	4.41%	12.49%	18.31%	25.56%	36.24%	43.32%
	SINGLE+FREQ	1.97%	7.30%	13.12%	21.38%	34.08%	42.41%
	SINGLE	2.02%	8.14%	14.01%	21.99%	33.13%	41.51%
	Morfessor+RNNLM	0.58%	2.53%	4.45%	7.53%	14.32%	20.49%

Table 7.2: Token-based OOV reduction rates for various numbers of generated words.

Language	Model	10k	50k	100k	200k	500k	1M
German	COMBINED+FREQ	0.61%	1.94%	3.44%	5.39%	9.16%	12.48%
	COMBINED	0.57%	2.03%	3.33%	5.28%	8.74%	11.99%
	SINGLE+FREQ	0.35%	1.59%	2.67%	4.20%	8.03%	11.76%
	SINGLE	0.43%	1.68%	2.90%	4.76%	7.77%	10.71%
	Morfessor+RNNLM	0.10%	0.56%	1.15%	2.18%	4.68%	6.84%
Finnish	COMBINED+FREQ	0.68%	2.63%	4.26%	7.10%	11.94%	16.73%
	COMBINED	0.67%	2.52%	4.23%	7.14%	12.31%	17.08%
	SINGLE+FREQ	0.39%	1.61%	2.96%	5.07%	9.61%	15.19%
	SINGLE	0.39%	1.72%	2.98%	5.40%	10.10%	14.97%
	Morfessor+RNNLM	0.19%	0.94%	1.72%	2.94%	5.42%	7.71%
Latin	COMBINED+FREQ	2.77%	9.00%	13.86%	20.08%	28.23%	33.53%
	COMBINED	2.41%	8.85%	13.60%	19.46%	28.43%	33.70%
	SINGLE+FREQ	1.50%	6.80%	11.39%	18.02%	26.93%	32.70%
	SINGLE	1.80%	7.03%	11.27%	17.61%	26.80%	32.82%
	Morfessor+RNNLM	0.44%	2.14%	3.84%	6.54%	12.34%	17.17%
Latvian	COMBINED+FREQ	1.71%	6.49%	10.74%	16.59%	25.74%	32.99%
	COMBINED	1.53%	5.91%	9.87%	15.33%	24.82%	32.45%
	SINGLE+FREQ	0.98%	4.45%	8.14%	13.77%	23.49%	31.53%
	SINGLE	0.94%	4.39%	7.92%	12.94%	22.49%	30.35%
	Morfessor+RNNLM	0.28%	1.23%	2.46%	4.80%	9.91%	14.59%
Polish	COMBINED+FREQ	1.36%	5.33%	8.77%	13.75%	21.91%	27.58%
	COMBINED	1.18%	4.78%	8.07%	12.73%	20.92%	27.38%
	SINGLE+FREQ	0.83%	3.97%	6.94%	11.57%	20.40%	26.54%
	SINGLE	0.85%	3.79%	6.37%	10.42%	18.87%	26.30%
	Morfessor+RNNLM	0.13%	0.65%	1.41%	2.91%	6.57%	10.97%
Romanian	COMBINED+FREQ	1.39%	4.96%	8.24%	12.63%	18.83%	23.79%
	COMBINED	1.25%	4.88%	7.85%	11.76%	18.77%	23.95%
	SINGLE+FREQ	0.91%	3.90%	6.75%	10.75%	17.41%	22.34%
	SINGLE	0.96%	4.07%	6.48%	10.04%	17.03%	22.44%
	Morfessor+RNNLM	0.29%	1.28%	2.26%	3.83%	7.17%	10.40%
Russian	COMBINED+FREQ	1.20%	4.88%	8.24%	12.82%	21.07%	27.39%
	COMBINED	1.07%	4.32%	7.36%	11.91%	20.07%	26.55%
	SINGLE+FREQ	0.79%	3.20%	6.02%	10.59%	19.28%	26.37%
	SINGLE	0.73%	3.15%	5.86%	10.22%	17.86%	24.99%
	Morfessor+RNNLM	0.24%	1.15%	2.12%	3.74%	7.49%	11.26%

Table 7.3: Type-based OOV reduction rates for various numbers of generated words.

to doubling the size of the training vocabulary (with the exception of Latin, where 100k is nearer), as well as 1M, which is around the size of the development corpus vocabulary for most languages.

Comparison between segmentation and Whole Word models. The results show a very clear advantage of the Whole Word Morphology model over the Morfessor+RNNLM model. The latter is outperformed by at least factor 2 for all vocabulary sizes, and for small sizes the difference is sometimes as high as factor 10 (e.g. Polish token-based 50k). There are multiple reasons for such a difference, which all amount to the fact that segmentation-based morphology models like Morfessor are not well-suited for vocabulary expansion. When taking the segmentation view of morphology, an accurate model of morphotactics is crucial in predicting which words can be formed. However, morphotactics is overlooked by models like Morfessor, which rather put emphasis on explaining the words that they are given. Using language models such as RNNLM as a model for morphotactics is a dubious solution, as the analogy between words in a sentence and morphs in a word seems quite far-fetched. On the other hand, the Whole Word Morphology approach of generating words from other existing words seems suitable for this task. It puts emphasis on generating missing inflected forms, rather than forming compounds or long chains of affixes. The 'combined' model prioritizes proposals that are supported by multiple existing words, thus minimizing the risk of applying a rule on a wrong base word. In general, the WWM approach tries to be 'safe' and stay as close to the known vocabulary as possible, exploring only the nearest neighborhood. In contrast, the 'disassemble and reassemble' approach of the segmentation model frequently produces words that are much more unlike anything seen in the training set. It is however necessary to note that the Whole Word approach works best for fusional morphologies and might run into trouble in highly agglutinating languages.

Finally, there is one more difference: the FST implementation of the Whole Word model makes it possible to discard proposals with high costs through simple thresholds. By doing so, we can compute *all* proposals with cost smaller than some threshold and subsequently sort them by cost. On the other hand, best-first search in models like RNNLM is not readily available (although probably possible), so we have to generate through sampling. Some high-probability candidates might be overlooked in this process.

Comparison of different Whole Word models. Slightly surprising is the fact that the frequency model turned out to be irrelevant in this task. It is yet unclear whether this is due to wrong parameter values, insufficient fitting, or simply lack of necessity of modeling the frequency. Cases like *jedes* → **jedsten* might turn out to be so rare that they do not impact the results. On the other hand, they could be countered by failing to generate some existing words because of their atypical frequency patterns.

The 'COMBINED' models perform consistently better than 'SINGLE'. However, the difference is in most cases not critical, indicating that either many generated words are supported by only a single existing word or that the differences in edge probabilities are so large that (7.4) is often dominated by a single term. The latter is indeed often the case.

There is a clear difference between the results for German and Finnish and other languages. German and Finnish are indeed different from the rest in that they make heavy use of concatenative morphology (especially compounding) and many OOV words are compounds. Additionally, the German inflectional morphology is fairly limited compared to the other languages (including Finnish). This accounts for lower difference between the performance of 'SINGLE' and 'COMBINED' models for German.

On the other hand, the five other languages all exhibit highly fusional morphology and extensive inflection. As my model aims primarily at generating missing inflected forms of words, for which other inflected forms are known, it is no surprise that it works best for languages, in which missing inflections, rather than e.g. compounds, contribute most significantly to OOV rates. Interestingly, this phenomenon can be also observed for the Morfessor+RNNLM approach, although to a smaller extent. While this model has little difficulty with generating new compounds, this strategy is less likely to produce words occurring in the development corpus than just exchanging an inflectional affix.

Comparison to maximum OOV reduction. The OOV reduction rates are somewhat difficult to interpret for small numbers of generated words: for instance, is 0.61% for German 10k-word experiment a good or bad result? For this reason, it is helpful to consider a different evaluation metric: the ratio of the achieved OOV reduction to the *maximum* achievable type-based OOV reduction at a given size. For example, assuming that all $10,000$ generated words occur in the development dataset, we can still achieve only $\frac{10{,}000}{972{,}596 \cdot 0.8474} \approx 1.21\%$ type-based OOV reduction. The value in the denominator is the number of OOV types in the development

Language	Model	10k	50k	100k	200k	500k	1M
German	COMBINED+FREQ	50.4 %	32.0 %	28.4 %	22.2 %	15.1 %	12.5 %
	Morfessor+RNNLM	8.0 %	9.3 %	9.5 %	9.0 %	7.7 %	6.8 %
Finnish	COMBINED+FREQ	70.4 %	54.5 %	44.1 %	36.8 %	24.7 %	17.3 %
	Morfessor+RNNLM	20.0 %	19.5 %	17.8 %	15.2 %	11.2 %	8.0 %
Latin	COMBINED+FREQ	73.5 %	47.7 %	36.7 %	26.6 %	28.2 %	33.5 %
	Morfessor+RNNLM	11.7 %	11.3 %	10.2 %	8.7 %	12.3 %	17.2 %
Latvian	COMBINED+FREQ	79.5 %	60.4 %	50.0 %	38.6 %	25.7 %	33.0 %
	Morfessor+RNNLM	13.0 %	11.4 %	11.4 %	11.2 %	9.9 %	14.6 %
Polish	COMBINED+FREQ	84.4 %	66.3 %	54.6 %	42.8 %	27.3 %	27.6 %
	Morfessor+RNNLM	8.2 %	8.0 %	8.8 %	9.1 %	8.2 %	11.0 %
Romanian	COMBINED+FREQ	67.0 %	48.0 %	39.8 %	30.5 %	18.8 %	23.8 %
	Morfessor+RNNLM	14.1 %	12.4 %	10.9 %	9.3 %	7.2 %	10.4 %
Russian	COMBINED+FREQ	85.7 %	69.8 %	58.9 %	45.8 %	30.1 %	27.4 %
	Morfessor+RNNLM	17.3 %	16.4 %	15.1 %	13.4 %	10.7 %	11.3 %

Table 7.4: Type-based OOV reduction as the percentage of the maximal reduction at a given size.

dataset, which can be computed from values given in Table 7.1. Thus, in this case, almost exactly half of the generated words contributed to the OOV reduction.

Formally, let N_{OOV} denote the number of OOV words in the dataset and $N_{\text{generated}}$ the number of generated words. The maximum achievable reduction is computed as follows:

$$OOV_{\text{max. reduction}} = \frac{\min\{N_{\text{generated}}, N_{\text{OOV}}\}}{N_{\text{OOV}}} \tag{7.6}$$

Table 7.4 presents the ratios of achieved to maximum achievable OOV reduction. In other words, the numbers represent the percentage of generated types that are encountered in the development dataset. Only the best Whole Word Morphology model was compared to Morfessor+RNNLM.

The scores for the Whole Word Morphology model, especially in the low sizes, are very high: the vast majority of predicted words is indeed encountered. This result could be utilized in a manually post-corrected vocabulary expansion. If increasing the vocabulary size is computationally expensive because of complexity issues (e.g. in a speech recognition system operating in real-time), we do not want to introduce garbage into our vocabulary. In this case, it is imaginable that each suggested new word would have to be approved by a human. The lists taken for manual inspection would necessarily be relatively short: several thousand to several tens of thousands words. The results in Table 7.4 suggest that roughly between

a half and three quarters of words present in such a list would be correct, which makes the manual inspection worth the effort.

Chapter 8

Conclusion

> And furthermore, my son, be admonished: of making many books there is no end; and much study is a weariness of the flesh.
>
> Ecclesiastes 12:12

This thesis presented a computational and statistical model of morphology inspired by linguistic theories rejecting the notion of morpheme, with special emphasis on Whole Word Morphology. The adaptation of WWM to Natural Language Processing was motivated by two main reasons: the inherent difficulty of segmentational methods of morphological analysis in handling non-concatenative phenomena (like German umlaut) and their overemphasis on difficult and irrelevant questions (like whether or not to segment cranberry morphs or how to mark morph boundaries blurred by morphophonological processes or orthography). Furthermore, whereas segmentational approaches have proved successful in handling agglutinative morphology, there seems to exist a gap for fusional morphology, for which a word-based approach is better suited.

The proposed computational model consisted of a formal definition of a *morphological rule* operating on whole existing words, along with the implementation of such rules as Finite State Transducers. The natural data structure to represent a morphologically annotated lexicon in this formalism is a labeled graph. Furthermore, I presented performant algorithms for extracting candidates for morphological rules and graph edges from unannotated corpora.

In order to assess the productivity of morphological rules and eliminate unnecessary ones, a probabilistic model for morphology graphs was introduced. It is designed with unsupervised training in mind, whereas the additional possibility of supervised training arises as a by-product. The fitting of model parameters

is done using Monte Carlo Expectation Maximization (MCEM), with expected values approximated by a graph sampler implementing the Metropolis-Hastings algorithm. Although no formal analysis of convergence and plausible stopping criteria was done, an informal analysis confirms the capability of this approach to explore high-probability regions of the sample space.

Evaluation tasks and results. Finding plausible evaluation methods for the learning approach was challenging, as the underlying theory rejects any notions of 'word structure' or 'analysis'. The only plausible evaluation is thus explaining words (their features or mere existence) in terms of other words, without explicitly referring to any representation of linguistic knowledge like segmentation or derivation trees. The evaluation on various tasks related to inflection, presented in Chap. 5, was already problematic, as the theory does not distinguish between inflection and derivation. However, it was attempted because of the high usefulness of such tasks. Somewhat unsurprisingly, the model did not provide a clear, consistent advantage over simple baselines on such tasks.

In Chapter 6, we examined the capability of the model to suggest POS tags for unknown words based on their morphological relationships to one another and use those suggestions to improve a tagger by re-training it on unannotated text. The results showed that the guessing of tags for OOV words significantly improves tagging results and that the graph-based morphology model proposes significantly better tags than a character-based recurrent neural network, yielding a tiny, but consistent improvement in tagging accuracy. The question of reproducing this improvement for state-of-the-art tagging methods and applying it to improve taggers for resource-scarce languages offers a promising direction for further research.

Finally, in Chapter 7, we evaluated the model on the task of vocabulary expansion. This was the most theory-agnostic evaluation task: it utilized only information directly observable in unannotated corpora (word existence and frequency) for both training and evaluation. The model scored its best results there, outperforming a state-of-the-art approach based on segmentation by a clear margin. The inspection of results showed that the methods work in fundamentally different ways: while the segmentation-based approach essentially coins new words by combining random morphs, the whole-word approach typically applies small and well-motivated changes to known words. As vocabulary expansion has important real-world applications, it is presumably the area where the methods developed in this thesis can most likely prove themselves useful.

In general, we can conclude the evaluation by stating that the whole-word

model works best for tasks that are consistent with the assumptions of Whole Word Morphology, i.e. do not require or evaluate any explicit representation of linguistic knowledge (like segmentation), treat all words equally (no 'lemmas', no 'complex' words being derived from 'simpler' base words) and do not discriminate between different types of morphology (like inflection and derivation). Such tasks tend to have simple and theory-agnostic gold standards, typically consisting only of directly observable data (POS tagging being an acceptable exception, as it is widespread and rarely disputed). The tasks described in Chapters 6 and 7 fulfill those requirements.

Main directions for further work. The requirement on this work to be completed by one person in a limited period of time imposed obvious limitations on its scope. Many possibilities were not explored in full detail, leaving room for improvements and further research.

Although theoretical foundations for including compounding in a Whole Word Morphology description were laid in Sec. 1.1.3, second-order rules were not implemented. Their inclusion in the probabilistic model poses problems, as they would make the dependency between the vocabulary and the rule set circular. However, the necessity of including compounding in a morphology model is clear.

The lack of constraint on tree height or explicit model of node degree leads to unintuitive graphs ('chains') – a problem which is described in Sec. 4.1.3. The ad-hoc solution proposed there is unsatisfactory. This problem needs more careful consideration.

The graph model is formulated as a framework with replaceable components. The catalog of components proposed in Sec. 4.2 is by no means exhaustive and some components (especially those based on neural networks) were designed hastily, without much consideration given to their architecture. Especially the failure of the neural edge model to offer an improvement over the simple model, despite being able to capture individual word features relevant for the probability of rule application, requires further inspection (and perhaps a better proposal for such model).

Although the model makes it possible to include word embeddings in the data, the experiments carried out in this thesis showed no benefit of doing so. The possibilities offered by word embeddings should be studied in more detail. Also the (hypothetical) capability of the model to predict embeddings for unknown words, as well as its application to improving OOV-word handling in neural language models, should be evaluated. Furthermore, the neural network edge and embedding

models produce embedding vectors for morphological rules as by-product. Those could turn out to be useful as well, e.g. for discovering paradigms or identifying 'synonymous' rules.

Bibliography

Adam Albright. *The Identification of Bases in Morphological Paradigms*. PhD thesis, University of California, Los Angeles, 2002.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA 2007)*, pages 11–23, 2007.

Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Inf. Retr.*, 12(4): 461–486, 2009.

Jan W. Amtrup. Morphology in machine translation systems: Efficient integration of finite state transducers and feature structure descriptions. *Machine Translation*, 18: 213–235, 2005.

Stephen R. Anderson. *A-Morphous Morphology*. 1992.

Mark Aronoff. *Word Formation in Generative Grammar*. MIT Press, 1976.

Mark Aronoff. In the beginning was the word. *Language*, 83(4):803–830, 2007.

Mark Aronoff and Kirsten Fudeman. *What is Morphology?* Fundamentals of Linguistics. Wiley, 2004.

R. H. Baayen, R. Piepenbrock, and L. Gulikers. The CELEX lexical database (CD-ROM), release 2, 1995.

Marco Baroni, Johannes Matiasek, and Harald Trost. Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In *Proceedings of the 6th Workshop of the ACL Special Interest Group on Phonology*, volume 6, pages 48–57, 2002.

Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. Center for the Study of Language and Information, 2003.

Julian Besag. An introduction to Markov Chain Monte Carlo methods. In Mark Johnson, Sanjeev P. Khudanpur, Mari Ostendorf, and Roni Rosenfeld, editors, *Mathematical Foundations of Speech and Language Processing*, pages 247–270. Springer-Verlag New York, Inc., 2004.

Chris Biemann. Chinese Whispers - an efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of TextGraphs: The First Workshop on Graph Based Methods for Natural Language Processing*, 2006.

Chris Biemann. Unsupervised natural language processing using graph models. In *Proceedings of the NAACL-HLT 2007 Doctoral Consortium*, pages 37–40, 2007a.

Chris Biemann, Gerhard Heyer, Uwe Quasthoff, and Matthias Richter. The Leipzig Corpora Collection: Monolingual corpora of standard size. In *Proceedings of Corpus Linguistics 2007*, Birmingham, UK, 2007.

Christian Biemann. *Unsupervised and Knowledge-free Natural Language Processing in the Structure Discovery Paradigm*. PhD thesis, University of Leipzig, 2007b.

David M. Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, April 2012.

Leonard Bloomfield. *Language*. University of Chicago Press, 1933.

Thomas Bocek, Ela Hunt, and Burkhard Stiller. Fast Similarity Search in Large Dictionaries. Technical report, University of Zurich, 2007.

Rens Bod, Jennifer Hay, and Stefanie Jannedy, editors. *Probabilistic Linguistics*. MIT Press, 2003.

Stefan Bordag. Two-step approach to unsupervised morpheme segmentation. In *Proceedings of the PASCAL Challenges Workshop on Unsupervised Segmentation of Words into Morphemes*, Venice, Italy, April 2006.

Stefan Bordag. *Elements of Knowledge-free and Unsupervised Lexical Acquisition*. PhD thesis, University of Leipzig, 2007.

Stefan Bordag. Unsupervised and knowledge-free morpheme segmentation and analysis. In Carol Peters, Valentin Jijkoun, Thomas Mandl, Henning Müller, Douglas W. Oard, Anselmo Peñas, Vivien Petras, and Diana Santos, editors, *Advances in Multilingual and Multimodal Information Retrieval*, pages 881–891. 2008.

Stefan Bordag and Gerhard Heyer. A structuralist framework for quantitative linguistics. In Alexander Mehler and Reinhard Köhler, editors, *Aspects of Automatic Text Analysis*, volume 209 of *Studies in Fuzziness and Soft Computing*. Springer, 2007.

Jan A. Botha and Phil Blunsom. Adaptor grammars for learning non-concatenative morphology. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 345–356, Seattle, Washington, 2013.

Jan A. Botha and Phil Blunsom. Compositional morphology for word representations and language modeling. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China, 2014.

Jan Abraham Botha. *Probabilistic Modelling of Morphologically Rich Languages*. PhD thesis, University of Oxford, 2014.

Burcu Can. *Statistical Models for Unsupervised Learning of Morphology and POS Tagging*. PhD thesis, University of York, 2011.

Julie Carson-Berndsen. *Time Map Phonology*, volume 5 of *Text, Speech and Language Technology*, chapter Finite State Techniques in Computational Phonology. Springer, Dordrecht, 1998.

Bruno Cartoni. Lexical morphology in machine translation: a feasibility study. In *Proceedings of the 12th Conference of the European Chapter of the ACL*, pages 130–138, 2009.

Çağrı Çöltekin. A freely available morphological analyzer for Turkish. In *LREC 2010, Seventh International Conference on Language Resources and Evaluation*, 2010.

Erwin Chan. Learning probabilistic paradigms for morphology in a latent class model. In *Proceedings of the Eighth Meeting of the ACL Special Interest Group on Computational Phonology at HLT-NAACL 2006*, pages 69–78, New York City, USA, June 2006.

Erwin Chan. *Structures and Distributions in Morphology Learning*. PhD thesis, University of Pennsylvania, 2008.

Noam Chomsky. Remarks on nominalization. In Roderic A. Jacobs and Peter S. Rosenbaum, editors, *Readings in English Transformational Grammar*. Georgetown University School of Language, 1970.

Grzegorz Chrupała, Georgiana Dinu, and Josef van Genabith. Learning morphology with morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, pages 2362–2367, 2008.

Alexander Clark. Memory-based learning of morphology with stochastic transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA, USA, 2002.

Ryan Cotterell, Thomas Müller, Alexander Fraser, and Hinrich Schütze. Labeled morphological segmentation with semi-markov models. In *Proceedings of CoNLL 2015*, 2015.

Mathias Creutz and Krista Lagus. Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0. Technical report, University of Helsinki, 2005a.

Mathias Creutz and Krista Lagus. Inducing the morphological lexicon of a natural language from unannotated text. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, 2005b.

Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arısoy, Murat Saraçlar, and Andreas Stolcke. Analysis of morph-based speech recognition and the modeling of out-of-vocabulary words across languages. In *Proceedings of NAACL HLT 2007*, 2007.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.

Colin de la Higuera and Franck Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In *Grammatical Inference: Algorithms and Applications – 5th international colloquium, ICGI 2000*, pages 141–156. Springer, 2000.

Ferdinand de Saussure. *Course in General Linguistics*. Philosophical Library, New York, 1916.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B. Methodological*, 39(1):1–38, 1977.

Christopher J. Dyer. The 'noisier channel': translation from morphologically complex languages. In *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague, 2007. Association for Computational Linguistics.

Manaal Faruqui, Ryan T. McDonald, and Radu Soricut. Morpho-syntactic lexicon generation using graph-based semi-supervised learning. *Transactions of the Association for Computational Linguistics*, 4:1–16, 2016.

Ronen Feldman and James Sanger. *The Text Mining Handbook*. Cambridge University Press, 2007.

Alan Ford, Rajendra Singh, and Gita Martohardjono. *Pace Pāṇini: Towards a word-based theory of morphology*. American University Studies. Series XIII, Linguistics, Vol. 34. Peter Lang Publishing, Incorporated, 1997.

Brendan S. Gillon. Pāṇini's aṣṭādhyāyī and linguistic theory. *Journal of Indian Philosophy*, 35(5-6):445–468, Dec 2007.

John Goldsmith. An algorithm for the unsupervised learning of morphology. *Natural Language Engineering*, 12(4):353–371, 2006.

Peter D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

Stig-Arne Grönroos, Sami Virpioja, Peter Smit, and Mikko Kurimo. Morfessor FlatCat: An HMM-based method for unsupervised and semi-supervised learning of morphology. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1177–1185, Dublin, Ireland, 2014.

Jan Hajič and Dan Zeman, editors. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, August 3-4, 2017*, 2017. Association for Computational Linguistics.

Morris Halle. Prolegomena to a theory of word formation. *Linguistic Inquiry*, 4(1), 1973.

Péter Halácsy, András Kornai, and Csaba Oravecz. Hunpos – an open source trigram tagger. In *Proceedings of the ACL 2007 Demo and Poster Sessions*, pages 209–212, Prague, 2007.

Harald Hammarström and Lars Borin. Unsupervised learning of morphology. *Computational Linguistics*, 37(2):309–350, 2011.

Zellig S. Harris. *Structural Linguistics*. The University of Chicago Press, 1951.

Zellig S. Harris. From phoneme to morpheme. *Language*, 31(2):190–222, 1955.

Zellig S. Harris. Morpheme boundaries within words: Report on a computer test. *Transformations and Discourse Analysis Papers*, 73, 1967.

Wilfred K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Jennifer B. Hay and R. Harald Baayen. Shifting paradigms: gradient structure in morphology. *TRENDS in Cognitive Sciences*, 9(7), July 2005.

Gerhard Heyer, Uwe Quasthoff, and Thomas Wittig. *Text Mining: Wissensrohstoff Text*. W3L Verlag, 2008.

Charles F. Hockett. Two models of grammatical description. *Word*, 10:210–234, 1954.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

Richard Hudson. *Word Grammar*. Basil Blackwell, 1984.

Maciej Janicki. Unsupervised learning of a-morphous inflection with graph clustering. In *Proceedings to RANLP 2013 Student Workshop*, Hissar, Bulgaria, 2013.

Maciej Janicki. Graphbasiertes unüberwachtes Lernen von Morphologie. MSc thesis, University of Leipzig, 2014.

Maciej Janicki. A multi-purpose Bayesian model for word-based morphology. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology – Fourth International Workshop, SFCM 2015*. Springer, 2015.

Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.

Bryan Jurish. Efficient online k-best lookup in weighted finite-state cascades. In Thomas Hanneforth and Gisbert Fanselow, editors, *Language and Logos: Studies in theoretical and computational linguistics*, pages 1–16. De Gruyter, 2010.

Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

Lauri Karttunen, Ronald M. Kaplan, and Annie Zaenen. Two-level morphology with composition. In *Proceedings of COLING-92*, pages 141–148, Nantes, France, 1992.

George Anton Kiraz. *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*. Cambridge University Press, New York, NY, USA, 2001.

Scott Kirkpatrick, Daniel C. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Amit Kirschenbaum. Unsupervised segmentation for different types of morphological processes using multiple sequence alignment. In *1st International Conference on Statistical Language and Speech Processing, SLSP*, pages 152–163, Tarragona, Spain, 2013.

Amit Kirschenbaum. To split or not, and if so, where? Theoretical and empirical aspects of unsupervised morphological segmentation. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, pages 139–150. Springer International Publishing, 2015.

Amit Kirschenbaum, Peter Wittenburg, and Gerhard Heyer. Unsupervised morphological analysis of small corpora: First experiments with kilivila. In Frank Seifart, Geoffrey Haig, Nikolaus P. Himmelmann, Dagmar Jung, Anna Margetts, and Paul Trilsbeek, editors, *Potentials of Language Documentation: Methods, Analyses and Utilization*, pages 25–31. 2012.

Friedrich Kluge. *Etymologisches Wörterbuch der deutschen Sprache*. de Gruyter, Berlin, 22. edition, 1989.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

Oskar Kohonen, Sami Virpioja, and Krista Lagus. Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78–86, Uppsala, Sweden, 2010.

Kimmo Koskenniemi. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. PhD thesis, University of Helsinki, 1983.

Mikko Kurimo, Sami Virpioja, Ville Turunen, and Krista Lagus. Morpho challenge 2005-2010: Evaluations and results. In *Proceedings of the 11th Meeting of the ACL-SIGMORPHON, ACL 2010*, pages 87–95, 2010.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009.

Jackson Lee and John Goldsmith. Linguistica 5: Unsupervised learning of linguistic structure. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 22–26, San Diego, California, June 2016. Association for Computational Linguistics.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Krister Lindén. A probabilistic model for guessing base forms of new words by analogy. In *CICling-2008, 9th International Conference on Intelligent Text Processing and Computational Linguistics*, Haifa, Israel, February 2008.

Krister Lindén. Entry generation by analogy – encoding new words for morphological lexicons. *Northern European Journal of Language Technology*, 1:1–25, 2009.

Krister Lindén and Jussi Tuovila. Corpus-based lexeme ranking for morphological guessers. In *Proceedings of the Workshop on Systems and Frameworks for Computational Morphology (SFCM)*, Zürich, Switzerland, 2009.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A. Pirinen, and Miikka Silfverberg. HFST – framework for compiling and applying morphologies. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology – Second International Workshop, SFCM 2011*. Springer, 2011.

Jiaming Luo, Karthik Narasimhan, and Regina Barzilay. Unsupervised learning of morphological forests. *Transactions of the Association for Computational Linguistics*, 5: 353–364, 2017.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

David Mareček. *Unsupervised Dependency Parsing*. PhD thesis, Charles University in Prague, 2012.

Beáta B. Megyesi. The open source tagger HunPoS for Swedish. In *Proceedings of the 17th Nordic Conference of Computational Linguistics (NODALIDA)*, 2009.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Andrei Mikheev. Automatic rule induction for unknown-word guessing. *Computational Linguistics*, 23(3):405–423, 1997.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Neural Information Processing Systems (NIPS)*, 2013a.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT 2013*, 2013b.

Tomáš Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *11th Annual Conference of the International Speech Communication Association 2010 (INTERSPEECH 2010)*, pages 1045–1048, 2010.

Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag Berlin Heidelberg, 2009.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. In *12th European Conference on Artificial Intelligence (ECAI 96)*, 1996.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. An unsupervised method for uncovering morphological chains. *Transactions of the Association for Computational Linguistics*, 3:157–167, 2015.

Sylvain Neuvel and Sean A Fulop. Unsupervised learning of morphology without morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning - Volume 6*, MPL '02, pages 31–40, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Eugene A. Nida. *Morphology: The Descriptive Analysis of Words*. University of Michigan Press, Ann Arbor, 1949.

Sonja Nießen and Hermann Ney. Improving SMT quality with morpho-syntactic analysis. In *Proceedings of the 18th Conference on Computational Linguistics (COLING '00)*, volume 2, pages 1081–1085, Saarbrücken, Germany, 2000.

Tommi A. Pirinen. *Weighted Finite-State Methods for Spell-Checking and Correction*. PhD thesis, University of Helsinki, 2014.

Tommi A. Pirinen. Development and use of computational morphology of Finnish in the open source and open science era: Notes on experiences with Omorfi development. *SKY Journal of Linguistics*, 28:381–393, 2015.

Hoifung Poon, Colin Cherry, and Kristina Toutanova. Unsupervised morphological segmentation with log-linear models. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL*, pages 209–217, 2009.

Maja Popović, Daniel Stein, and Hermann Ney. Statistical machine translation of German compound words. In *FinTAL - 5th International Conference on Natural Language Processing*, 2006.

Franz Rainer. Semantic fragmentation in word-formation: The case of Spanish -AZO. In Singh and Starosta [2003], pages 197–211.

Mohammad Sadegh Rasooli, Thomas Lippincott, Nizar Habash, and Owen Rambow. Unsupervised morphology-based vocabulary expansion. In *52nd Annual Meeting of the Association for Computational Linguistics*, pages 1349–1359, 2014.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Jorma Rissanen. An introduction to the MDL principle, 2005. URL `http://www.mdl-research.net/jorma.rissanen/pub/Intro.pdf`.

Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Teemu Ruokolainen, Oskar Kohonen, Sami Virpioja, and Mikko Kurimo. Supervised morphological segmentation in a low-resource learning setting using conditional random fields. In *CoNLL*, 2013.

David Rybach. *Investigations on Search Methods for Speech Recognition using Weighted Finite-State Transducers*. PhD thesis, RWTH Aachen, 2014.

Rajhans Samdani, Ming-Wei Chang, and Dan Roth. Unified expectation maximization. In *2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 688–698, 2012.

Helmut Schmid. A programming language for finite state transducers. In *Proceedings of the 5th International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 2005)*, Helsinki, Finland, 2005.

Patrick Schone and Daniel Jurafsky. Knowledge-free induction of morphology using latent semantic analysis. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning – CoNLL '00*, pages 67–72, Stroudsburg, PA, USA, 2000.

Elisabeth O. Selkirk. *The syntax of words*. MIT Press, 1982.

Miikka Silfverberg and Krister Lindén. HFST runtime format – a compacted transducer format allowing for fast lookup. In *Finite-State Methods and Natural Language Processing - FSMNLP 2009 Eight International Workshop*, 2009.

Miikka Silfverberg, Pekka Kauppinen, and Krister Lindén. Data-driven spelling correction using weighted finite-state methods. In *Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata*, pages 51–59, Berlin, Germany, 2016.

Rajendra Singh and Probal Dasgupta. On so-called compounds. In Singh and Starosta [2003], pages 77–89.

Rajendra Singh and Stanley Starosta, editors. *Explorations in Seamless Morphology*. SAGE Publications, New Delhi, 2003.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meating of the Association for Computational Linguistics*, 2005.

Herbert Weir Smyth. *A Greek Grammar for Colleges*. American Book Company, 1920.

Radu Soricut and Franz Josef Och. Unsupervised morphology induction using word embeddings. In *NAACL 2015*, pages 1626–1636, 2015.

Sebastian Spiegler. *Machine Learning for the Analysis of Morphologically Complex Languages*. PhD thesis, University of Bristol, 2011.

Sebastian Spiegler and Christian Monson. EMMA: A novel evaluation metric for morphological analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 1029–1037, Beijing, 2010.

Stanley Starosta. *The Case for Lexicase: An Outline of Lexicase Grammatical Theory*. Open linguistics series. Pinter Publishers, 1988.

Stanley Starosta. Do compounds have internal structure? A seamless analysis. In Singh and Starosta [2003], pages 116–147.

Milan Straka and Jana Straková. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, 2017.

Maciej Sumalvico. Unsupervised learning of morphology with graph sampling. In *Proceedings to RANLP 2017*, Varna, Bulgaria, 2017.

Robert E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

Christoph Teichmann. *Markov Chain Monte Carlo Sampling for Dependency Trees*. PhD thesis, University of Leipzig, 2014.

Antal van den Bosch and Walter Daelemans. Memory-based morphological analysis. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 285–292, 1999.

Matti Varjokallio and Dietrich Klakow. Unsupervised morph segmentation and statistical language models for vocabulary expansion. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 175–180, 2016.

Sami Virpioja, Jakko J. Väyrynen, Mathias Creutz, and Markus Sadeniemi. Morphology-aware statistical machine translation based on morphs induced in an unsupervised manner. In *Proceedings of the Machine Translation Summit XI*, pages 491–498, Copenhagen, Denmark, 2007.

Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Python implementation and extensions for Morfessor baseline. Technical report, Aalto University, Helsinki, 2013. URL `http://urn.fi/URN:ISBN:978-952-60-5501-5`.

Stephan Vogel. *Statistical Machine Translation with Cascaded Probabilistic Transducers*. PhD thesis, RWTH Aachen, 2005.

Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(I):168–173, 1974.

Greg C. G. Wei and Martin A. Tanner. A Monte Carlo Implementation of the EM Algorithm and the Poor Man's Data Augmentation Algorithms. *Journal of the American Statistical Association*, 85(411):699–704, 1990.

Richard Wicentowski. *Modeling and Learning Multilingual Inflectional Morphology in a Minimally Supervised Framework*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, 2002.

Kay-Michael Würzner and Bryan Jurish. Dsolve – morphological segmentation for German using conditional random fields. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology – Fourth International Workshop, SFCM 2015*, pages 94–103, 2015.

David Yarowsky and Richard Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 207–216, 2000.

Andrea Zielinski and Christian Simon. Morphisto – an open source morphological analyzer for German. In *Finite State Methods and Natural Language Processing, 7th International Workshop, FSMNLP 2008*, Ispra, Italy, 2008.

Bibliographische Daten

Autor: Maciej Janicki
Titel: Statistical and Computational Models for Whole Word Morphology
173 Seiten, 30 Abbildungen, 16 Tabellen, 13 Algorithmen

Das Ziel dieser Arbeit ist die Formulierung eines Ansatzes zum maschinellen Lernen von Sprachmorphologie, in dem letztere als Zeichenkettentransformationen auf ganzen Wörtern, und nicht als Zerlegung von Wörtern in kleinere stukturelle Einheiten, modelliert wird. Der Beitrag besteht aus zwei wesentlichen Teilen: zum einen wird ein Rechenmodell formuliert, in dem morphologische Regeln als Funktionen auf Zeichenketten definiert sind. Solche Funktionen lassen sich leicht zu endlichen Transduktoren übersetzen, was eine solide algorithmische Grundlage für den Ansatz liefert. Zum anderen wird ein statistisches Modell für Graphen von Wortableitungen eingeführt. Die Inferenz in diesem Modell erfolgt mithilfe des Monte Carlo Expectation Maximization-Algorithmus und die Erwartungswerte über Graphen werden durch einen Metropolis-Hastings-Sampler approximiert. Das Modell wird auf einer Reihe von praktischen Aufgaben evaluiert: Clustering flektierter Formen, Lernen von Lemmatisierung, Vorhersage von Wortart für unbekannte Wörter, sowie Generierung neuer Wörter.

Wissenschaftlicher Werdegang

2015-2018 Promotionsstipendium: Landesinnovationsstipendium aus den Mitteln des Europäischen Sozialfonds (ESF).
Thema: *Statistisches Lernen wortbasierter Morphologie.*
Betreuer: Prof. Dr. Uwe Quasthoff, Prof. Dr. Gerhard Heyer.

seit 2014 Promotionsstudent an der Universität Leipzig, Institut für Informatik, Abteilung Automatische Sprachverarbeitung;

Leitung von Lehrveranstaltungen:

- Linguistische Informatik (Übung und teilweise Vorlesung),
- Text Mining: Wissensrohstoff Text (Übung),
- Algorithmen und Datenstrukturen 1 (Übung),
- Anwendungen Linguistische Informatik (Seminar – Betreuung von Gruppenprojekten).

Teilnahme und Veröffentlichung an internationalen Fachtagungen: RANLP 2013, SFCM 2015, RANLP 2017, LREC 2018.

2012-2014 Masterstudium Informatik mit Ergänzungsfach Linguistik an der Universität Leipzig; DAAD-Stipendium für ausländische Studierende;
Abschlussnote: 1,2
Titel der Masterarbeit: *Graphbasiertes unüberwachtes Lernen von Morphologie.*
Betreuer: Prof. Dr. Uwe Quasthoff

2008-2012 Bachelorstudium Informatik an der Technischen Universität Wrocław, Polen (Wrocław University of Technology);
Abschlussnote: *celujący (lobenswert)*
Titel der Bachelorarbeit: *Usługa sieciowa do rozpoznawania nazw własnych z wykorzystaniem akwizycji wiedzy z dostępnych zasobów elektronicznych (Web service for the recognition of proper names based on the knowledge extraction from available electronic resources).*
Betreuer: Dr. Maciej Piasecki

2008 Abitur am Allgemeinbildenden Lyzeum Nr. 3 in Wrocław, Polen;
Wahlfächer: Mathematik, Physik.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den _____ _____