A NON-PARAMETRIC MODEL FOR THE DISCOVERY OF INFLECTIONAL PARADIGMS FROM PLAIN TEXT USING GRAPHICAL MODELS OVER STRINGS

by

Markus Dreyer

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

Version 1.0.1, April, 2011

Abstract

The field of statistical natural language processing has been turning toward morphologically rich languages. These languages have vocabularies that are often orders of magnitude larger than that of English, since words may be inflected in various different ways. This leads to problems with data sparseness and calls for models that can deal with this abundance of related words—models that can learn, analyze, reduce and generate morphological inflections. But surprisingly, statistical approaches to morphology are still rare, which stands in contrast to the many recent advances of sophisticated models in parsing, grammar induction, translation and many other areas of natural language processing.

This thesis presents a *novel, unified statistical approach to inflectional morphology*, an approach that can decode and encode the inflectional system of a language. At the center of this approach stands the notion of inflectional paradigms. These paradigms cluster the large vocabulary of a language into structured chunks; inflections of the same word, like *break, broke, breaks, breaking,* ..., all belong in the same paradigm. And moreover, each of these inflections has an exact place within a paradigm, since each paradigm has designated slots for each possible inflection; for verbs, there is a slot for the *first person singular*

indicative present, one for the *third person plural subjunctive past* and slots for all other possible forms. The main goal of this thesis is to *build probability models over inflectional paradigms*, and therefore to sort the large vocabulary of a morphologically rich language into structured clusters. These models can be learned with minimal supervision for any language that has inflectional morphology. As training data, some sample paradigms and a raw, unannotated text corpus can be used.

The models over morphological paradigms are developed in three main chapters that start with smaller components and build up to larger ones.

The first of these chapters (Chapter 2) presents novel probability models over strings and string pairs. These are applicable to lemmatization or to relate a past tense form to its associated present tense form, or for similar morphological tasks. It turns out they are general enough to tackle the popular task of transliteration very well, as well as other string-to-string tasks.

The second (Chapter 3) introduces the notion of a probability model over multiple strings, which is a novel variant of Markov Random Fields. These are used to relate the many inflections in an inflectional paradigm to one another, and they use the probability models from Chapter 2 as components. A novel version of belief propagation is presented, which propagates distributions over strings through a network of connected finite-state transducers, to perform inference in morphological paradigms (or other string fields).

Finally (Chapter 4), a non-parametric joint probability model over an unannotated text corpus and the morphological paradigms from Chapter 3 is presented. This model is based

on a generative story for inflectional morphology that naturally incorporates common linguistic notions, such as lexemes, paradigms and inflections. Sampling algorithms are presented that perform inference over large text corpora and their implicit, hidden morphological paradigms. We show that they are able to discover the morphological paradigms that are implicit in the corpora. The model is based on finite-state operations and seamlessly handles concatenative and nonconcatenative morphology.

Jason Eisner

David Yarowsky

Colin Wilson

Acknowledgments

In the years that led to this thesis, the CLSP with its intellectually stimulating atmosphere created by both faculty and students has made a lasting impression on me. There are many people I would like to thank.

My advisor Jason Eisner inspired me with his extremely creative thinking, his curiosity and abundance of research ideas. He has taught me to never stop asking questions, seeking improvements and thinking ahead in research. Fortunately, for him no research idea is too crazy to be considered and no path too new to be followed.

My committee members David Yarowsky and Colin Wilson were great supporters of this work. Their feedback and ideas were tremendously helpful in making stronger points and extending the scope of this work. Colin broadened my horizon in terms of the linguistic underpinning of my work. David provided me with many insights on what aspects of my work need to be stressed to create a strong impact on the NLP community.

Sadly, Fred Jelinek passed away, just a few weeks before I left Baltimore. I remember him as bright and joyful until the last minute, his positive mood was contagious. Every

CLSP seminar he opened with a smile or some funny remark. I am grateful that his visions paved the way for many of us in the field.

The many friends and brilliant students at the CLSP made it a much better place. I remember countless technical discussions with Zhifei Li, Jason Smith, Ariya Rastrow, Carolina Parada and Nicholas Andrews. Zhifei provided me with job-search advice in the last months; Nick helped me keep my sanity by dragging me out of the office for long-distance runs through the city. I always enjoyed spending time with the MT gurus Juri Ganitkevitch, Omar Zaidan and Jonathan Weese, although they were hidden on the other floor of the building. In the early days, the NLP juggernauts Noah Smith, David Smith, and Roy Tromble were still there, helpfully introducing me to a whole new world of being a (CLSP) grad student. I am glad I had them as office mates and collaborators.

Both Izhak Shafran and Keith Hall were friends and enthusiastic collaborators as well. I am glad our paths crossed at Johns Hopkins and am very thankful for all their support. Similarly, Sanjeev Khudanpur and Damianos Karakos were great collaborators and mentors. Chris Callison-Burch is a very positive force at CLSP and I am grateful for lots of good advice during my last year at Hopkins.

Special thanks go to the Human Language Technology Center of Excellence (HLT-COE), which funded the last few years of my work. I am thankful that I could be part of the COE since its beginnings and it was sad to leave when it was starting to become a major player in NLP. Mark Dredze, Christine Piatko and James Mayfield were great team

leaders and I am grateful for their support. I would also like to thank the National Science Foundation (NSF) which graciously funded part of my work.

Last but not least, my thanks go to my family in Germany who has always supported me in whatever I wanted to do. My parents instilled in me a great curiosity for language and knowledge in general, and without their love and support this thesis would not exist.

Dedication

This thesis is dedicated to my family and friends who made this work possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem

Statistical natural language processing can be difficult for morphologically rich languages. Morphological transformations on words increase the size of the observed vocabulary, which unfortunately masks important generalizations.

In a very highly inflected language like Polish, for example, each lexical verb has literally 100 forms (Janecki, 2000). That is, a single *lexeme*[1] may be realized in a corpus as many different word types, which have been differently inflected for person, number, gender, tense, mood, or others.

The existence of these morphological transformations and the resulting abundance of different surface forms for each word make lexical features even sparser than they would be otherwise. In machine translation or text generation, it is difficult to learn *separately* how to translate, or when to generate, each of these many word types. In text analysis, it is difficult to learn lexical features as cues to predict topic, syntax, semantics, or the next word, because lexical features that fire on unanalyzed words fail to generalize across multiple inflections. Even for a reasonably frequent lexeme, some of its inflections may occur rarely or never in the training corpus, but it would be desirable to derive statistics for these inflections from related inflections. In many text generation tasks in general, one wishes to generate the correct morphological inflections of the words in the output text, while at the same time being able to abstract away from such morphological spelling variety in core parts of the model that deal with the lexical resources and lexical transfer.

The obvious solution to these problems is to morphologically analyze each word token occurring in text. However, this is not trivial, since inflectional patterns may be unknown or irregular. Morphological transformations can be complex and nonconcatenative processes; they can involve stem changes, consonant doubling, prefix, infix or suffix changes and other changes that can make two related words look almost unrelated on the surface. Therefore,

[1] A lexeme is the abstract notion of a word, independent of its various specific realizations in text, e.g. the lexeme BRING can be realized in text by the inflected forms *bring*, *brings*, *brought*, *bringing*, ...

lemma		brechen	
singular	1st-person	breche	brach
	2nd-person	brichst	brachst
	3rd-person	bricht	brach
plural	1st-person	brechen	brachen
	2nd-person	brecht	bracht
	3rd-person	brechen	brachen
		present	past

Table 1.1: Simplified inflectional paradigm for the German lexeme BRECHEN.

manually constructing a good morphological analyzer for a language (Beesley and Karttunen, 2003) is difficult and usually includes creating both rules and lists of irregular forms.

1.2 Proposed Solution

This thesis tackles the problem of morphological generalization by developing a novel statistical approach to inflectional morphology. We take a Word-and-Paradigm approach (Matthews, 1972; Stump, 2001), where *paradigms* are central linguistic objects. Paradigms are grids of word types like the simple example shown in Table 1.1, where each cell contains a different inflectional form. Modeling paradigms requires modeling relationships between words as whole-word transformations, unlike previous work that assumes concatenative approaches (Section 2.5 on page 25 and Section 4.7 on page 114).

We develop a novel probability model over inflectional paradigms—a model that is flexible, robust, configurable, modular, unified and mathematically sound. We use empirical Bayesian inference to reason about data, without the need for handcoded rules or detailed expert knowledge about morphological properties of the language. The data used for learning can be a small set of example paradigms coupled with a raw, unannotated text corpus in the language.[2]

Our model learns how to cluster the large vocabulary of a morphologically rich language into inflectional paradigms; there is one per lexeme in the language. Once learned, the model can be used to make morphological generalizations. It can be used to generate the most likely spellings for a given morphological form; it can generate complete paradigms for a given lemma; and it can complete incompletely observed paradigms.

We also devise practical computational methods to carry out inference under this model; our inference method naturally combines serveral known approaches, such as dynamic programming, belief propagation and Gibbs sampling, in a new way.

[2]The example paradigms contain information about the possible inflectional forms of the language.

The proposed solution consists of several components, each of which is a novel contribution to the field by itself. These components will be presented one by one in the following chapters, starting from the smallest component and building up the final model, piece by piece.

We will now briefly describe what these components are and what they do (sections 1.2.1, 1.2.2 and 1.2.3); after that, we will describe the components again with the focus on how they work (sections 1.3.1, 1.3.2 and 1.3.3).

1.2.1 Latent-Variable Modeling of String Transductions

A probability model over inflectional paradigms must be able to assign a probability score to any instantiation of a given paradigm. For example, the correct paradigm instantiation shown in Table 1.1 should presumably receive a higher probability score than a similar instantiation where the second person singular present is changed from the correct form `brichst` to the incorrect `brechst`—a typical mistake that a language learner might make and an even lower score if that form is changed to a completely nonsensical form like `sadfgzxghsfdfdwqx`. How can we tell if a form at a given slot in a given paradigm is bad and should result in an overall lower score for the paradigm?

There are generally two ways to find out: (a) We can either look at the form by itself and judge if it is—using the example—a likely German second person singular present (e.g. "Does the form end in `-st`?" Good. "Does the form end in `-qx`?" Bad.), which includes judging if it is a likely German word in general and if it is a likely present-tense form in general, etc. (b) The other way to find out is to ask if it is good in the *context* of the other forms in the paradigm. If the second person form looks similar to the third person form with just the typical 2nd-to-3rd person changes applied, then the score should probably be high, but if a form looks nothing like its neighbor in the paradigm and there is no typical transformation to inflect one form to get the other, then we have a low-scoring candidate.

Focusing on these questions, Chapter 2 presents a novel solution to model string pairs by constructing highly configurable probability distributions $p(x, y)$ or $p(y \mid x)$ over two strings x and y. Using this model, we can learn typical morphological transformations that happen between form pairs, e.g. from the second person singular present to the third person plural subjunctive.

As we will see later, these probability models can be used as part of our bigger model that scores whole inflectional paradigms (Chapters 3 and 4).

But, moreover, because this model is data-independent and language-independent, it can potentially be applied to *any* string-to-string problem in natural language processing where some systematic mapping from an input string x to an output string y is needed, including the following:

- *phonology*: underlying representation $\leftrightarrow$ surface representation
- *orthography*: pronunciation $\leftrightarrow$ spelling

- *morphology*: inflected form ↔ lemma, or differently inflected form
- *fuzzy name matching* (duplicate detection) and *spelling correction*: spelling ↔ variant spelling
- *lexical translation* (cognates, loanwords, transliterated names): English word ↔ foreign word

This probability model for string pairs is a log-linear model, which scores string pairs by summing over all their alignments and evaluating flexible, potentially overlapping features on them, which may encode linguistic knowledge. The model is encoded as a finite-state transducer, and training and decoding can be done using efficient finite-state operations. Model features are encoded as features firing on particular transitions of the finite-state transducer, and their weights are trained from data. In addition to the string pair alignment, which is latent, this dissertation will show how to encode other latent variables such as clusters (conjugation classes) or regions. All models are evaluated on morphological tasks, but also a transliteration task.

1.2.2 Graphical Models over Multiple Strings

Defining these generally useful and novel probability models over string pairs is an important milestone that brings us closer to the main goal: to define probability models over whole inflectional paradigms of a language. Chapter 3 now presents such *bigger* models, using the smaller string-to-string models from Chapter 2 as ingredients. Here, a similar approach as in Chapter 2 is followed by formulating general, mathematically clean models, which are novel, this time over *more* than just one or two strings. These models have again many potential applications, but we will focus on the main goal of modeling inflectional paradigms. Other tasks for which the presented general multiple-string models may be useful include the following:

- mapping an English word to its foreign *transliteration* may be easier when one considers the orthographic *and* phonological forms of both words;
- similar *cognates* in multiple languages are naturally described together, in orthographic or phonological representations, or both;
- modern and ancestral word forms form a phylogenetic tree in *historical linguistics*;
- in *bioinformatics* and in *system combination*, multiple sequences need to be aligned in order to identify regions of similarity.

The general formulation of multiple-string models presented in this thesis is a novel formulation of Markov Random Fields (MRF), in which each variable is string-valued and each potential function is a finite-state machine.

Joint inference techniques can be used to predict multiple strings jointly. It is shown how a standard inference method like the belief propagation message-passing algorithm can be applied in this model, resulting in a novel view of belief propagation, in which each message is a finite-state acceptor; these acceptors can be seen as being passed around through a network of finite-state transducers (Section 3.4.1). It is shown why inference becomes intractable and what approximations can be used. Experimental results show that predicting strings jointly often outperforms separate predictions.

1.2.3 Discovering Morphological Paradigms from Plain Text

The last content chapter, Chapter 4, extends the work of the previous chapters into a model of the inflectional morphology of a language that can be learned from unannotated text, with minimal supervision. A joint probability model over inflectional morphological paradigms and the spellings in an unannotated text corpus is built. This is used to discover the morphological paradigms that are implicit and hidden in the corpus.

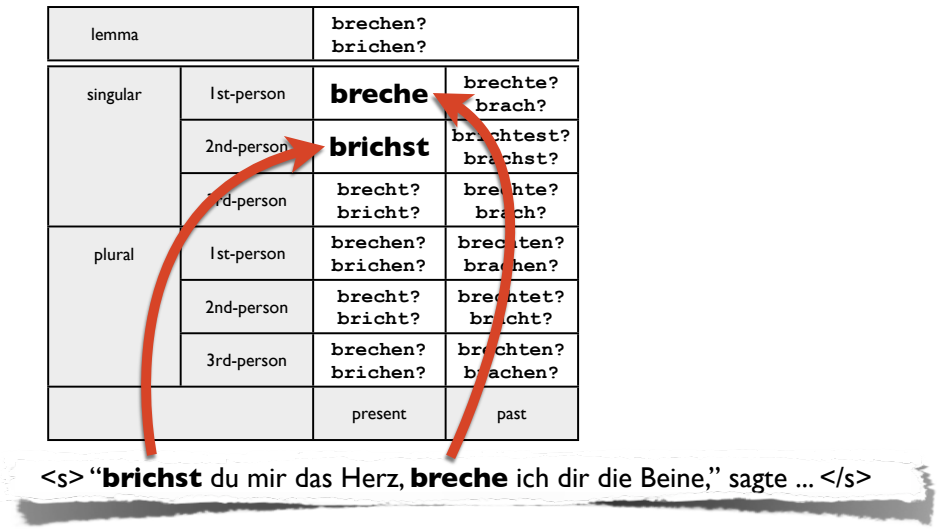

Figure 1.1: Illustration of the basic idea: Words from the text corpus are placed in inflectional paradigms; this reduces uncertainty in neighboring slots.

Figure 1.1 illustrates the basic idea of this approach. It shows an observed text corpus,[3] as well as a paradigm. Two slots in the paradigm are already filled with spellings found in the text; the model has predicted the exact paradigm cell for the spellings from

[3]English translation of this German text based on an album title by Ollie Schulz: <*s*> " *If you break my heart I break your legs", said* ... Note how both the German *brichst* and *breche* (and other forms) map to the uninflected *break* in English.

their characteristic suffixes and potentially other information. The remaining cells of the paradigm are still unfilled. But the cells that are already filled and "clamped" to particular spellings propagate information through some channel to the other, still unfilled variables (see Section 4.4.2), which results in hypothesized spellings for all remaining forms in the paradigm, e.g. for the third person singular present, *brecht* and *bricht* are hypothesized, both of which would make sense because they largely look like the forms found in the text but have the characteristic third person suffix `-t`.[4] Other words in the text may be placed in this or in other paradigms (not shown in the figure). The number of paradigms is data-dependent, and there is no upper limit. Each form in the text corpus may be placed in a paradigm that was previously created or in one that is newly created on seeing this form. A learned probability model decides where it fits best.

More formally, a nonparametric Dirichlet process mixture model determines for each corpus spelling what inflection in what paradigm it has been generated from; this way a distribution over infinitely many paradigms in the text corpus is learned. Each paradigm is modeled using a graphical model over strings (Chapter 3). As described in the example above, each word token is assigned a slot in a particular paradigm during inference. At any time during inference, some cells in some paradigms will be empty; the model maintains a posterior distribution over their possible values given the spellings of their neighboring slots, using belief propagation (Section 3.4).

It is shown how to add a small amount of supervision by observing a few paradigms as seed data, which facilitates learning. The model is evaluated on the tasks of learning correct morphological paradigms, as well as correctly determining the morphological forms of the spellings in a text corpus.

1.3 Approach

Now that we have briefly described *what methods* we propose to tackle the problem of morphological clustering and inflection, we give a brief overview of *how* it works. The following three subsections correspond to the three subsections above, 1.2.1, 1.2.2, and 1.2.3, respectively, but with the focus on how the proposed methods work.

1.3.1 Latent-Variable Modeling of String Transductions

How does the described log-linear model over string pairs (Chapter 2) work? Suppose we have two strings x and y and we would like to evaluate how well they go together for a particular task, i.e., we evaluate the goodness or (task-based) similarity between the strings. We may, for example, be interested in how good they are as a pair of present- and past-tense forms of a particular verb. In our solution, we look at all alignments of these

[4] These suffixes and other parameters are all learned from data in a semi-supervised way (see Section 4.5.3).

two strings; this can be done efficiently using dynamic programming. On each alignment we systematically identify certain features or properties. For example, in the first of the

Figure 1.2: Examples of one-to-one alignments of the two strings *breaking* and *broke*, including attached start and end symbols. An ϵ symbol means a non-character (empty character).

four alignments shown in Figure 1.2, we recognize that `b` in the upper string (*breaking*) is aligned to `b` in the lower one (*broke*), that `br` is aligned to `br`, that `re` is aligned to `r`ϵ,[5] and so forth, for all alignment substrings (called *alignment* n*-grams*) up to a specified length. We also make some linguistic generalizations (*backoff*), by mapping all substrings to linguistically relevant classes like vowels or consonants; in the example we find various consonant-to-consonant and vowel-to-vowel alignments as features.[6]

Each such feature that we detect on an alignment of x and y has a certain weight between negative and positive infinity (the weights are configured beforehand for the particular task (see below), indicating a degree of lower or higher task-based similarity. For a given alignment, the feature weights are summed and exponentiated, resulting in a non-negative score for that alignment. The same is done for all possible alignments of x and y,[7] and the overall sum is returned as similarity score for the string pair. We normalize to turn it into a probability score $p(y \mid x)$ or $p(x, y)$. Of course, the probability model is typically used to search for and predict the best output string for some input string, rather than just scoring two known strings, x and y.

[5]The character ϵ is traditionally the empty character; aligning an input character to an ϵ in the output means *deleting* the input character. The inverse case would be an *insertion*.

[6]See more details on features in Section 2.4 on page 24

[7]Note that even obviously implausible alignments are considered. These will typically have negtive weights, resulting in contributions close to zero after exponentiating, so they contribute very little to the overall score. (Practically, some implausible alignments may in fact be pruned away for more efficient computation, see Section 2.6.2 on page 31.)

We will see that the features just mentioned in fact predict output strings well (Section 2.9) but that it is often desirable to add further features classes—features that look at certain properties of x and y that are not given in the training data at all, since they would be too costly to annotate by hand. A conjugation class in morphology is an example. We never observe conjugation classes in training (or test) data, but our model can prefer a vowel change for irregular verbs and avoid it for regular verbs—when predicting a past-tense form, for example. How does this work? We just add features that are the conjunction of features described above with a conjugation class. In other words, we add specialized versions of the above features that are sensitive to the conjugation class of the string pair. The fact that the conjugation class is actually not observed means that we have to treat it as a latent variable and sum over all possibilities, just like we do for the alignment between x and y, described above. The model is therefore semi-supervised.

We do the same for another important piece of information not given in the data: string-pair regions. Some transductions (e.g. a vowel change) are likely only in certain regions. For example, a vowel change might be likely in an irregular verb in general, but it is even more likely *in a certain region* of an irregular verb. Consider the example (*ride*, *rode*). The `i` changes to *o*, but the `e` remains unchanged. Just as in the case with the conjugation classes, we add features that are sensitive to certain regions, e.g. a feature *`i` aligned to o in region 2*. The information where certain regions start or end is not given and is treated as a latent variable.

As mentioned, the score is typically configured, meaning its feature weights are trained, based on training data consisting of correct (x, y) pairs for a given task. The score can be regarded as a generalization of weighted edit distance (Ristad and Yianilos, 1998), with the addition of context, backoff features (e.g., vowel/consonant features) and latent variables, giving the model the ability to learn more characteristic and meaningful patterns from the data. It follows that the simple, untrained (negative) Levenshtein distance (Levenshtein, 1966) is also a special case of our score; it can be obtained by setting the weights of insertion, deletion and substitution unigrams to (negative) 1.0 and all other weights to 0.0,[8] regardless of any training data that might be available.

1.3.2 Graphical Models over Multiple Strings

How does the described graphical model over strings (Chapter 3) work?

As described above, this novel variant of graphical models is designed to predict multiple output strings jointly. A typical task would be to jointly predict all possible morphological forms of a verb given the lemma, or to predict the transliterations of a given name into many foreign alphabets.

To understand our approach, first consider graphical models in general. Graphical models decompose a big joint probability into smaller factors, each of which is relatively easy to compute (Jordan, 1998). A simple example of a graphical model is a conditional random

[8] No exponentiation is needed in that case; the Levenshtein distance is a linear, not a log-linear score.

field (CRF; Lafferty, McCallum, and Pereira, 2001b),[9] which defines a joint probability over a number of output variables given some input, where each output variable ranges over a set of possible values, e.g., part-of-speech tags. To make inference tractable, the joint probability is decomposed into small factors each of which considers only the output for two directly adjacent positions in the sequence. Such factors know what two tags go well with one another, e.g. a verb following a noun is better than an adjective following a noun in English. But the decomposition means there is no factor that would know what five or ten tags go well with one another in a sequence, since the cardinality would be too large.

Such a probability model is depicted in Figure 1.3 on the next page on the left, where some functions (drawn as black boxes) connect neighboring output variables (drawn as circles) to one another and some connect single output variables to the input—similar to transition and emission functions in a Hidden Markov Model (HMM; Rabiner and Juang, 1986). To find the best tag sequence in such a graph, the forward-backward algorithm is run (Jelinek, 1997), which propagates information about the possible values of the variables through the graph. In Figure 1.3, for example, if the first part-of-speech variable is almost certainly a proper noun—since the first word is *Obama*—the second variable is likely to be a verb since the factor in between knows that verbs tend to follow nouns and proper nouns.[10] That information is propagated to the third variable, for which the factor in between then may decide it should more likely be a noun than anything else (since nouns tend to follow verbs), and so forth. A similar pass is run from right to left as well, hence the name of the algorithm. In the machine-learning community, such propagation of information through a factor graph is known as *message passing*.

More complicated graph structures are possible, in which even output variables that are not sequence neighbors of each other are connected; graph structures may be tree-shaped or loopy. In these cases, the forward-backward algorithm is replaced by a more general message passing algorithm, loopy belief propagation (see Section 3.4)—a straightforward generalization of the forward-backward algorithm.

In our approach, we also build factor graphs and run the (loopy) belief propagation algorithm. A graphical model over strings might look like the one illustrated in Figure 1.3 on the following page on the right. The factor graph has similar structure as the one on the left,[11] but here each variable is *string-valued*. We attempt to depict this by placing small additional circles into the variable—after all, a variable here contains much more information than in the simple CRF case; it can be considered as being composed of several small variables, each one representing one charcater of the string. However, we do not know the lengths of the output strings, and so we cannot construct a factor graph that would have one variable per output character for each output string; instead each of the strings is represented by one string-valued variable that is not split into smaller pieces.

[9] To be more specific, these are *undirected* graphical models, which do not have the local normalization constraints that directed graphical models, like HMMs, have.

[10] After being trained on English data.

[11] Of course, the factor graph does not have to be chain-structured here either.

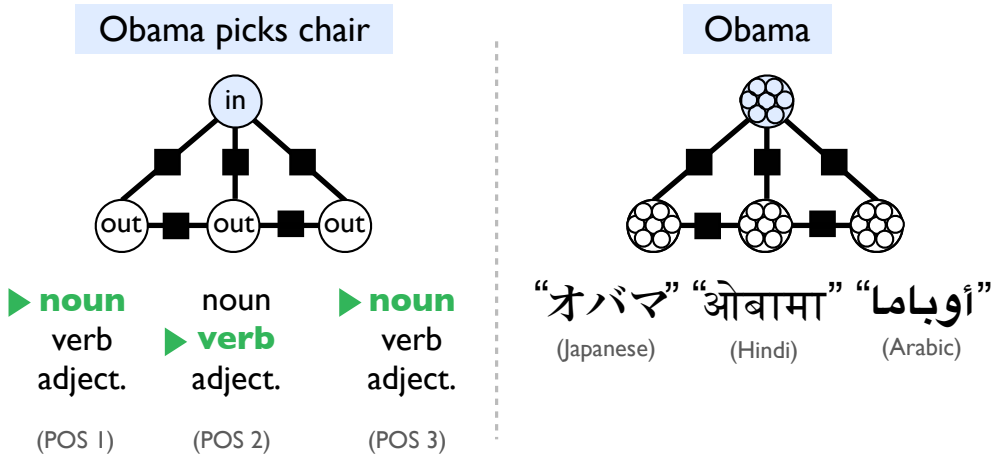

Figure 1.3: Factor graph examples for joint prediction. In both graphs, some input is given (shaded circle), and values for three output variables (the three lower circles) need to be predicted. **Left:** Joint prediction of tags means to select correct tags from a list of known tags in the context of other tags that are being predicted. **Right:** Joint prediction of strings means to generate strings by converting neighboring strings. In the example, the Hindi name transliteration is created by converting the English, the Japanese and the Arabic spellings into the Hindi spelling, where the Japanese and the Arabic transliterations themselves are being predicted by conversion of neighboring string values.

The novelty in our approach in Chapter 3 is that we generalize graphical models to define joint probability models over *string-valued* variables, and we re-formulate message passing for that scenario. Each factor between variables in a factor graph now relates *strings* to one another—a nested sequence modeling problem in itself! Whereas in the CRF case (and in other traditional graphical models), the factors are simple lookup tables that store goodness values for pairs (or triples, etc.) of tags (e.g., determining that noun-verb is x times better than noun-adjective), here we need a more complex mechanism to relate variables to one another. We use finite-state machines. The very purpose of a weighted finite-state transducer is to define a weighted relation between two strings, so they are the natural choice for the use in our graphical models over strings. To our knowledge, this has not been done before, and a general framework for joint probability models over strings with associated inference procedure did not exist before we introduced this work in (Dreyer and Eisner, 2009). We had introduced our novel finite-based string-to-string framework just a year before (Dreyer, Smith, and Eisner, 2008); due to its generality and configurability it is ideal for the use as factors in our multiple-string models.

In our version of belief propagation, the information sent from variable to variable—the *message* between the variables—is always a finite-state machine. To be exact, a message to or from a variable is a finite-state acceptor that contains the possible values of that variable associated with the goodness scores. In the CRF case, a variable sends a vector of its

possible (e.g., part-of-speech tag) values; here we send a whole finite-state acceptor which compactly represents all the possible string values of the variable—a possibly infinite number of strings encoded using a finite number of states. The belief propagation equations cleanly translate into well-known finite-state operations like intersection, composition, and projection (Mohri, 2009). As mentioned above, inference becomes intractable; we present methods to approximate the finite-state messages (Section 3.4.3 on page 67).

In the example in Figure 1.3 then, the Hindi spelling is influenced by the English input *Obama*, but also by the likely string values of the Japanese and the Arabic variables, which themselves are being predicted by looking at the likely values of the other string variables connected through the finite-state factors. That way, each variable prediction can be made using much more information than a model that would predict every foreign spelling separately just from the English spelling. Since this thesis is primarily concerned with morphology, we ran all experiments in Chapter 3 and Chapter 4 on morphological data. But the example shows that morphology is not the only scenario in which joint prediction of multiple strings can be useful and that the multiple-string framework that we present is general enough to handle various multiple-string problems.

1.3.3 Discovering Morphological Paradigms from Plain Text

How does the described discovery of morphological paradigms (Chapter 4) work? It will be easiest to understand the main ideas and intuitions by following an example: Figure 1.4 on page 15 illustrates a simplified run of the inference process we will present, on a small example text corpus. By just following the illustration row by row, it should become clear how we learn morphological paradigms. It will be shown how our approach starts from plain text, reads some seed paradigms, and repeatedly analyzes the tokens in the text, getting an increasingly clear estimate of token frequencies and type spelling probabilites, thereby learning morphological rules and extracting morphological paradigms from the text. After we present the example, we summarize the approach briefly in more general terms. All technical details and mathematical aspects of the model and the inference process are described in Chapter 4.

1.3.3.1 An Illustration of the Inference Process

Figure 1.4, first row.
We start by observing an unannotated text corpus, as shown in the figure. We would like to use that text to learn inflectional verb morphology, so we assign part-of-speech tags and select the verbs. (In Chapter 4, we will describe how *all* words could be processed.) We are prepared to assign each verb a lexeme and an inflection (see *Lex* and *Infl* on the right).

Figure 1.4, second row.
(1) We are given seed paradigms.[12] These paradigms are immediately analyzed, and a morphological grammar is learned. We learn from the shown German seed paradigm, for example, that the third person singular ends in a `-t`, or that particular stem vowel or suffix changes happen when converting from singular to plural or between other forms. This is exactly the kind of information that can be learned using the graphical models from Chapter 3.
(2) Then we analyze the first verb token, *bricht*. We need to assign a lexeme and an inflection, which is equivalent to finding a paradigm cell in which that spelling is likely. We see it cannot belong into the seed paradigm, so it must be analyzed as a new lexeme. We open a new paradigm for it. The decision about the particular inflection of *bricht* is probabilistic and takes into account at least two probabilities; let us assume third person singular is a hypothesis:

- How likely is it that a third person singular form is spelled *bricht*?
- And, how likely is it to encounter a third person singular form (or a third person form or a singular form in general, etc.) in text?

The answers to both these questions are not given to us beforehand, but we will learn them as we go. The seed paradigms do give us an initial estimate to answer the first question (*bricht* ends in a `-t`, just like the third person singular in the seed paradigm), but we will sharpen all our estimates as we go through the text and analyze more verbs. By analyzing the token *bricht* as third singular we increase the probability that other tokens we will encounter later are also analyzed as third singular. And we can use that spelling as a new example of the spelling of a third singular form, enriching our morphological grammar.[13] That is an instance of Monte Carlo EM (see Section 4.5).

When we analyze the spelling *bricht* as third singular we are determining that the third singular form in that paradigm be spelled *bricht*. We say that the verb token *bricht* moves to, or "sits down" at the third singular cell in the paradigm.[14]
(3) It is important that we now get an estimate of what spellings we will expect in the other cells of that paradigm. The third singular has just been clamped to *bricht*, and for the other forms we get (pruned) probability distributions over possible values by running the finite-state-based belief propagation from Chapter 3 (see Section 4.4.2).

Figure 1.4, third row.
(1) We now proceed to the second verb (*brechen*) in the corpus. Thanks to the belief propagation run, we find that it would fit well in two places of the recently opened paradigm.

[12] In the actual experiments, paradigms will have many more slots, but for illustration purposes we just show six morphological forms per paradigm.

[13] In practice, such updates to the morphological grammar are made only sporadically in batches, for better efficiency and stability.

[14] We use a Chinese Restaurant Process (Blei, Griffiths, Jordan, and Tenenbaum, 2004); in our version, the verb enters the lexeme restaurant and sits down at a particular inflection table.

(2) In the example, we move it to the third plural form. It is important to note that, in terms of token frequencies, the third plural is already slightly more likely than, say, the first plural because we have already observed another token that was analyzed as third person. Our token frequency estimates get sharper and sharper that way; this is an essential property of the Chinese Restaurant Process (Blei et al., 2004), which we use (Section 4.3). Note also that, when the word *brechen* moves to that third plural cell, the spelling for third plural in that paradigm (lexeme) is (tentatively) fixed to the value *brechen*; in our example this removes all incorrect hypotheses that we previously had obtained for that cell by running belief propagation. This is an important aspect of our sampler and one of the main reasons for enriching the methods of Chapter 3 with these token-based sampling mehods: Using the tokens from the text corpus, we can often exclude incorrect hyptheses that were generated by the graphical models. After assigning *brechen* to that third plural paradigm cell, we have to run belief propagation again, to propagate that new information to the other cells in the paradigm. We then move on to the third verb (*springt*) and open a new paradigm for it, since its spelling is so different from the other verbs seen so far.[15]
(3) We run belief propagation in the new paradigm, to get estimates for the possible values in all its cells.

Figure 1.4, fourth row.
(1) The fourth verb (*brechen*) has similar spelling as the second verb and likely moves into that same paradigm, i.e. is analyzed as the same lexeme, although there is a probability that it would open a new paradigm.[16]
(2) In the example, we place it in the same cell as the second verb, for two reasons: The third paradigm row (third person forms) has become more and more likely, since other verbs have been analyzed as third person. And the third plural in that paradigm is (currently) already fixed to that spelling, so the probability for that form to be spelled *brechen* is one, whereas it is lower in the first person cell in that paradigm.
(3) The fifth verb has the same spelling as a particular cell in the *seed* paradigm above (second singular), so it is very likely to move there and be analyzed as that particular morphological form—although there is a small probability that it opens a new paradigm instead.
(4) We move through the corpus many times[17] and reconsider earlier analyses in the light of newly acquired knowledge. This time, for example, the first verb might move from its previous place to the second person plural since the fact that we analyzed the fifth verb as a second person has made second person forms slightly more likely. As noted above, not only do we sharpen our token frequency estimates that way, by using our samples as

[15] In fact, due to pruning, there is a zero probability that *springt* sits at any of the previously opened paradigms.
[16] Proportional to a hyperparameter, which is also learned during inference (Section 4.5).
[17] We do not necessarily process the words in the given order; certain type-based samplers are more efficient, see Section 4.4.4.

training data, but we similarly keep reestimating our knowledge of how certain forms are spelled.

1.3.4 Summary of the Paradigm Discovery Approach

With this example, we have attempted to show how and why we can learn morphological paradigms from text with minimal supervision. A key aspect of the approach is that, by using sampling, we repeatedly analyze the tokens in the text, annotating them with lexeme and inflection information. The seed paradigms give us a good initialization point from which we can start recognizing other forms in text and assigning morphological analyses accordingly. These analyses help estimating frequencies of certain morphological forms in text, which in turn can help analyze further forms correctly. An analysis is always two-way; when we analyze a particular token spelling from the corpus by assigning a particular morphological type we are making a deterministic decision about the spelling of that morphological type. Whenever we are not sure yet about the spelling of a certain morphological type we estimate a probability distribution over the possible spellings by running our finite-state belief propagation (Chapter 3). In that way, we naturally combine the sampling process under a Chinese Restaurant Process with belief propagation in graphical models over strings, which in turn uses dynamic programming to compute messages—with the goal of discovering the morphological paradigms that are implicit in the text.

1.4 Conclusion

This dissertation presents a novel, unified and modular approach to statistical inflectional morphology. The main goal is to develop probability models over an unbounded number of inflectional paradigms, which can be learned from data. Several components of this model are presented in different chapters, including algorithms for efficient inference and learning. Two of these chapters are based on previous publications of the author, see (Dreyer et al., 2008) and (Dreyer and Eisner, 2009), and have been extended here by adding more exposition and connections to the bigger picture presented in this dissertation, and by more experiments and error analysis.

It will be shown that our model can successfully predict morphological inflections, lemmas, transliterations, and whole inflectional paradigms. Of course, there are also limitations. All our models or components are general-purpose and can be applied to different tasks; hand-tailored features or extensions might be better suited for some particular morphological phenomena, such as infixation or reduplication (see Section 2.10 on page 51). Our final model, which discovers inflectional paradigms from plain text, is a solid starting point for further explorations of this topic. In the future, one might want to add context or extend it to model agglutinative morphology (see Section 4.8 on page 118).

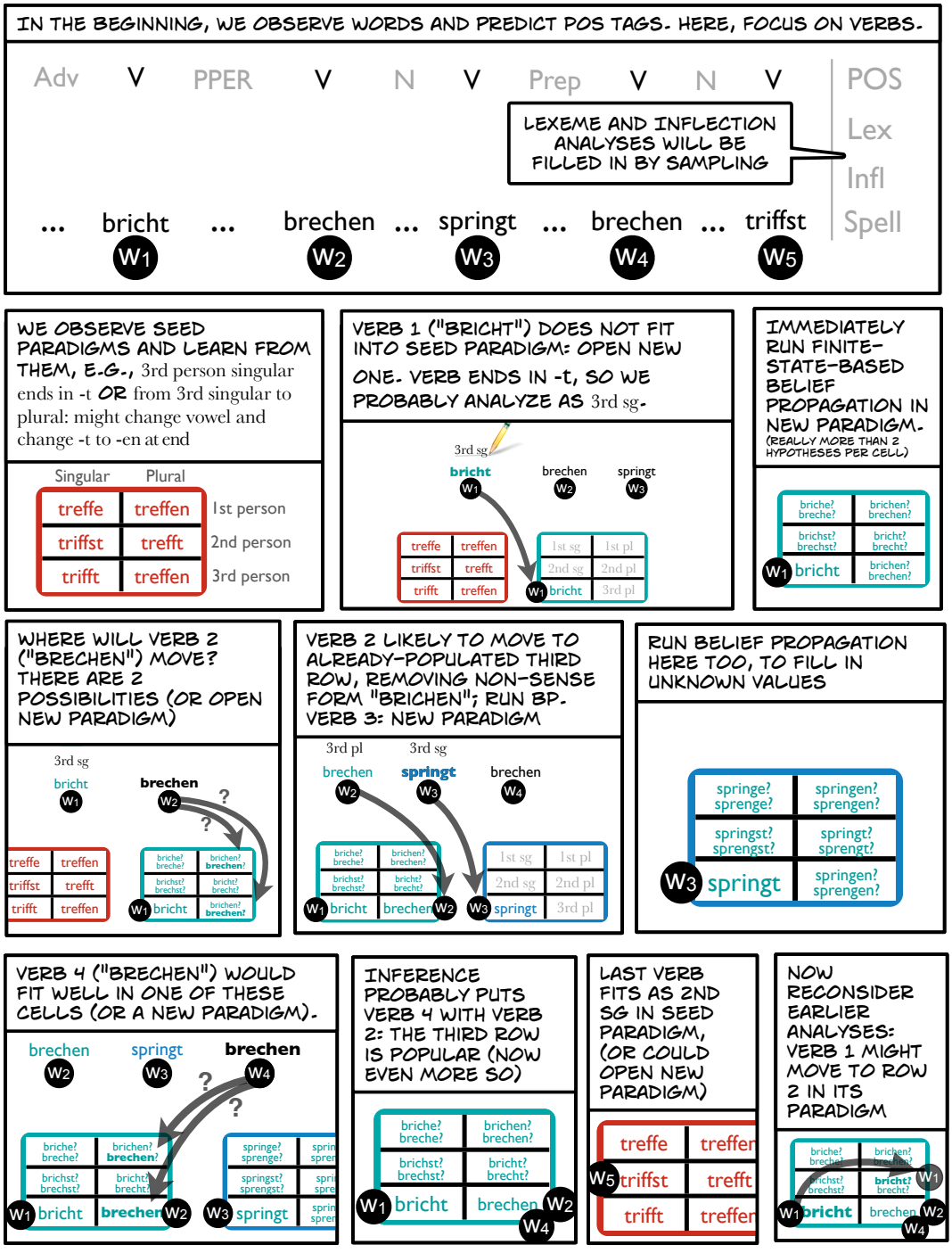

Figure 1.4: Illustration of the inference process: Corpus tokens are analyzed, and such analyses determine the spelling of morphological types.

Chapter 2

Latent-Variable Modeling of String Transductions with Finite-State Methods

2.1 Introduction

The most fundamental building block in our model for inflectional morphology is a novel approach to modeling strings and string pairs. The approach is described in this chapter; later chapters will refer to and make extensive use of it.

This new approach to modeling strings and string pairs is very general. It is not only applicable to morphology—although we will mainly focus on morphological applications—but is generally applicable to all tasks in computational linguistics and language processing that require some systematic mapping from an input string x to an output string y. Applications include:

- *phonology*: underlying representation $\leftrightarrow$ surface representation
- *orthography*: pronunciation $\leftrightarrow$ spelling
- *morphology*: inflected form $\leftrightarrow$ lemma, or differently inflected form
- *fuzzy name matching* (duplicate detection) and *spelling correction*: spelling $\leftrightarrow$ variant spelling
- *lexical translation* (cognates, loanwords, transliterated names): English word $\leftrightarrow$ foreign word

We present here a configurable and robust framework for solving such string or word transduction problems. This approach uses finite-state transducers that are trained as conditional log-linear probability models. Such an approach makes it possible to handle flexible and linguistically motivated features. Such features are often not available in supervised training data; we employ a semi-supervised strategy by introducing various classes of latent variables to discover the unannotated information. The presented results in morphology generation improve upon the state of the art.

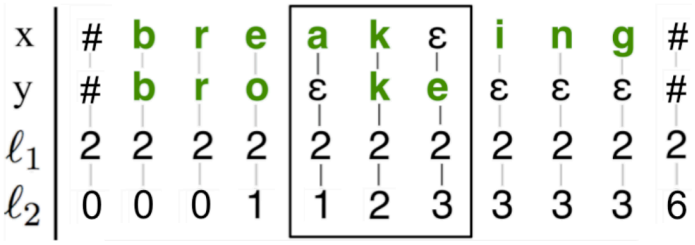

Figure 2.1: One of many possible alignment strings A for the observed pair *breaking/broke*, enriched with latent strings ℓ_1 and ℓ_2. Observed letters are shown in bold. The box marks a trigram to be scored. See Figure 2.2 on page 22 for features that fire on this trigram.

2.2 Model

The presented model is a probability model over string pairs, which can also be used as a model over single strings, using a trivial reduction.

The model is inspired by the weighted edit distance model (Ristad and Yianilos, 1998). However, edit distance considers each character in isolation. To consider more context, we pursue a very natural generalization. Given an input x, we evaluate a candidate output y by moving a sliding window over the aligned (x, y) pair. More precisely, since many alignments are possible, we sum over all these possibilities, evaluating each alignment *separately*.

At each window position, we accumulate log-probability based on the material that appears within the current window. The window is a few characters wide, and successive window positions *overlap*. In other words, our basic approach is a log-linear n-gram model over (x, y) pairs.

This stands in contrast to a competing approach (Sherif and Kondrak, 2007; Zhao, Bach, Lane, and Vogel, 2007) that is inspired by phrase-based machine translation (Koehn, Hoang, Birch, Callison-Burch, Federico, Bertoldi, Cowan, Shen, Moran, Zens, Dyer, Bojar, Constantin, and Herbst, 2007), which *segments* the input string into substrings that are transduced *independently*, ignoring context. Section 2.5 will discuss related approaches further.

Our basic approach is further advanced by adding new *latent* dimensions to the (input, output) tuples (see Figure 2.1). This enables us to use certain linguistically inspired features and discover unannotated information. Our features consider *less* or *more* than a literal n-gram. On the one hand, we generalize with features that abstract away from the n-gram window contents; on the other, we specialize the n-gram with features that make use of the added latent linguistic structure (see Section 2.4 on page 24).

Our framework uses familiar log-linear techniques for stochastic modeling, and weighted finite-state methods both for implementation and for specifying features. It appears general enough to cover most prior work on word transduction. One can easily add

new, linguistically interesting classes of features, each class defined by a regular expression.

2.2.1 Basic Notation and Definitions

2.2.1.1 Alphabets

We use an **input alphabet** Σ_{x} and **output alphabet** Σ_{y}. We conventionally use $x \in \Sigma_{\text{x}}^*$ to denote the input string and $y \in \Sigma_{\text{y}}^*$ to denote the output string.

There are many possible alignments between x and y. We represent each as an **alignment string** $A \in \Sigma_{\text{xy}}^*$, over an **alignment alphabet** of ordered pairs, $\Sigma_{\text{xy}} \stackrel{\text{def}}{=} ((\Sigma_{\text{x}} \cup \{\epsilon\}) \times (\Sigma_{\text{y}} \cup \{\epsilon\})) - \{(\epsilon, \epsilon)\}$.

For example, one alignment of $x = \texttt{breaking}$ with $y = \texttt{broke}$ is the 9-character string $A = (\texttt{b} : \texttt{b})(\texttt{r} : \texttt{r})(\texttt{e} : \texttt{o})(\texttt{a} : \epsilon)(\texttt{k} : \texttt{k})(\epsilon : \texttt{e})(\texttt{i} : \epsilon)(\texttt{n} : \epsilon)(\texttt{g} : \epsilon)$. It is pictured in the first two lines of Figure 2.1 on the preceding page.

The remainder of Figure 2.1 shows how we introduce latent variables, by enriching the alignment characters to be tuples rather than pairs. Let $\Sigma \stackrel{\text{def}}{=} (\Sigma_{\text{xy}} \times \Sigma_{\ell_1} \times \Sigma_{\ell_2} \times \cdots \times \Sigma_{\ell_K})$, where Σ_{ℓ_i} are alphabets used for the latent variables ℓ_i.

2.2.1.2 Finite-state Automata

In this chapter and throughout this dissertation, we will use finite-state automata to efficiently represent and compute with sets of strings or string pairs. In particular, we will use *weighted* automata, in which each string or string pair is assigned a weight, which is an element from a semiring (see page 19), which may, for example, be some unnormalized model score or a probability.

Finite-state automata in their weighted and unweighted forms are popular tools in natural language processing and computational linguistics and have been applied to problems in morphology and phonology (Kaplan and Kay, 1994; Karttunen, Kaplan, and Zaenen, 1992), language modeling (Roark, Saraclar, Collins, and Johnson, 2004a), speech recognition (Pereira, Riley, and Mohri, 2002) and machine translation (Knight and Al-Onaizan, 1998). However, a generic formulation of log-linear string transduction models with latent variables using finite-state machines is a new contribution.

Finite-state *acceptors* (FSAs) are automata that operate on single strings. Finite-state *transducers* (FSTs) generalize this to the *string-pair* case; they can be thought of as accepting string pairs or rewriting an input string into an output string.

We give the formal definition of a weighted finite-state transducer, following Mohri (2009):

Definition A weighted finite-state transducer (WFST) T over a semiring $(W, \oplus, \otimes, \bar{0}, \bar{1})$ (see below) is an 8-tuple $T = (\Sigma_x, \Sigma_y, Q, I, F, E, \lambda, \rho)$. The symbols have the following meanings:

Name	W	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	{0,1}	$\vee$	$\wedge$	0	1
Real	$\mathbb{R}$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup -\infty$	logplus	$+$	$-\infty$	0
Tropical	$\mathbb{R} \cup -\infty$	max	$+$	$-\infty$	0

Table 2.1: Semiring examples ($\text{logplus}(v, w)$ is defined as $\log(\exp v + \exp w)$).

- Σ_x: finite input alphabet (also known as upper alphabet)
- Σ_y: finite output alphabet (also known as lower alphabet)
- Q: finite set of states
- $I \subset Q$: set of initial states[1]
- $F \subset Q$: set of initial states
- E: a finite multiset of transitions, which are elements of $Q \times (\Sigma_x \cup \epsilon) \times (\Sigma_y \cup \epsilon) \times W \times Q$
- $\lambda : I \to W$: initial weight function
- $\rho : F \to W$: final weight function

As can be seen in the WFST definition above, the transitions as well as the initial and final weight functions make use of a weight from a **semiring** $(W, \oplus, \otimes, \bar{0}, \bar{1})$. The semiring associates the weight with definitions of how to add ($\oplus$) or multiply ($\otimes$) two weights. Formally, in a semiring, we have the following properties:

- $(W, \oplus, \bar{0})$ is a commutative monoid with identity element $\bar{0}$
- $(W, \otimes, \bar{1})$ is a monoid with identity element $\bar{1}$,
- $\otimes$ distributes over $\oplus$
- $\bar{0}$ is an annihilator for $\otimes$, i.e. $w \otimes \bar{0} = \bar{0} \otimes w = \bar{0}, \forall w \in W$

Table 2.1 shows examples of frequently used semirings. In addition, we make use of the expectation semiring in this work (Eisner, 2001, 2002b; Li and Eisner, 2009b).

Given a weighted finite-state transducer T, the weight of a string pair (x, y) is the sum of the weights of all paths leading from an initial to a final state that accept (x, y). A path π is an element of E^* with consecutive transitions. The weights of different paths are

[1] Typically, this set contains just *one* distinguished initial state.

summed using the semiring element $\oplus$, while the weight of a single path is the product (computed using $\otimes$) of its individual transition weights.

Weighted finite-state automata can be manipulated and combined using various algorithm, which Mohri (2009) describes in detail. In this and the following chapters, we make use of the following algorithms on weighted finite-state automata:

- intersection
- composition (denoted by the $\circ$ symbol)
- projection
- determinization
- minimization
- epsilon removal

Let T be a relation and w a string. We write $T[w]$ to denote the image of w under T (i.e., range$(w \circ T)$), a set of 0 or more strings. Similarly, if W is a weighted language (typically encoded by a WFSA), we write $W[w]$ to denote the weight of w in L.

Let $\pi_{\text{x}} \subseteq \Sigma^* \times \Sigma_{\text{x}}^*$ denote the deterministic regular relation that projects an alignment string to its corresponding input string, so that $\pi_{\text{x}}[A] = x$. Similarly, define $\pi_{\text{y}} \subseteq \Sigma^* \times \Sigma_{\text{y}}^*$ so that $\pi_{\text{y}}[A] = y$. Let $\boldsymbol{A}_{\boldsymbol{xy}}$ be the set of alignment strings A compatible with x and y; formally, $\boldsymbol{A}_{\boldsymbol{xy}} \stackrel{\text{def}}{=} \{A \in \Sigma^* : \pi_{\text{x}}[A] = x \wedge \pi_{\text{y}}[A] = y\}$. This set will range over all possible alignments between x and y, and *also* all possible configurations of the latent variables.

2.2.2 Log-linear Modeling

We use a standard log-linear model whose features are defined on alignment strings $A \in \boldsymbol{A}_{\boldsymbol{xy}}$, allowing them to be sensitive to the alignment of x and y. The model is globally normalized over the range of complete possible output strings given an input string.

We define a collection of features $f_i : \Sigma^* \to \mathbb{R}$ with associated weights $\theta_i \in \mathbb{R}$; the conditional likelihood of the training data is

$$p_{\boldsymbol{\theta}}(y \mid x) = \frac{\sum_{A \in \boldsymbol{A}_{\boldsymbol{xy}}} \exp \sum_i \theta_i f_i(A)}{\sum_{y'} \sum_{A \in \boldsymbol{A}_{\boldsymbol{xy'}}} \exp \sum_i \theta_i f_i(A)} \tag{2.1}$$

Given a parameter vector $\boldsymbol{\theta}$, we compute Equation 2.1 using a finite-state machine. We define a WFSA, $U_{\boldsymbol{\theta}}$, such that $U_{\boldsymbol{\theta}}[A]$ yields the unnormalized probability $u_{\boldsymbol{\theta}}(A) \stackrel{\text{def}}{=} \exp \sum_i \theta_i f_i(A)$ for any $A \in \Sigma^*$. (See Section 2.6.3 on page 32 for methods of constructing such a machine.) To obtain the numerator of Equation 2.1, with its $\sum_{A \in \boldsymbol{A}_{\boldsymbol{xy}}}$, we sum over all paths in $U_{\boldsymbol{\theta}}$ that are compatible with x and y. That is, we build the transducer

$x \circ \pi_{\mathrm{x}}^{-1} \circ U_{\boldsymbol{\theta}} \circ \pi_{\mathrm{y}} \circ y$ and sum over all paths. For the denominator we build a transducer $x \circ \pi_{\mathrm{x}}^{-1} \circ U_{\boldsymbol{\theta}}$, which is larger since it is not constrained to any particular y, and again compute the pathsum. We use standard algorithms (Eisner, 2002b) to compute the pathsums as well as their gradients with respect to $\boldsymbol{\theta}$ for optimization (Section 2.7).

The sums over alignment strings A in Equations 2.1 and 2.2 are restricted to only include *valid* alignment strings in Σ^*. Only monotone alignments, without reordering, are valid. The restriction is done by constructing the finite-state machines such that *invalid* alignments are not accepted.

Typically, $f_i(A)$ counts the number of n-gram window positions in A with property i. The properties of a single window position are illustrated in Figure 2.2.

For optimization, we can compute the gradient of $\log p_{\boldsymbol{\theta}}(y \mid x)$ with respect to its features θ_i according to

$$\frac{\partial \log p_{\boldsymbol{\theta}}(y \mid x)}{\partial \theta_i} = \sum_{A \in \boldsymbol{A}_{xy}} p_{\boldsymbol{\theta}}(A \mid x, y) f_i(A) - \sum_{y'} \sum_{A \in \boldsymbol{A}_{xy'}} p_{\boldsymbol{\theta}}(y', A \mid x) f_i(A), \tag{2.2}$$

which is the difference of the feature expectations given x and y and the feature expectations given just x. The alignment between the two strings is always unobserved, so we take expectations.

Note that, on the surface, our model is similar to a conditional random field (CRF, see Lafferty, McCallum, and Pereira (2001a)), with the alignment added as a latent variable, similar to hidden conditional random fields (Gunawardana, Mahajan, Acero, and Platt, 2005; Quattoni, Wang, Morency, Collins, and Darrell, 2007). However, a key difference is that in our model, x and y are not simply fixed-length vectors of simple multinomials, as in typical random fields that, for example, model sequences of part-of-speech tags. In contrast, x and y in our model are *structured objects* of potentially unbounded lengths. The length of the output string y is unknown and could, due to insertions and deletions, be longer or shorter than the input string x. In that respect, our model is similar to Finkel, Kleeman, and Manning (2008b), which defines a globally normalized model over parse trees given an input string; however, in that model, no latent variables are used (all trees are observed). Petrov and Klein (2008) define a log-linear model over trees with latent variables, with the difference that it is locally normalized per rewrite rule.

Since we model more complex variables than does a traditional sequence (hidden) conditional random field, we need more involved inference algorithms and representations of the data. Fortunately, these are well-known for weighted finite-state machines; they just have never been used to model strings in a CRF-like probability model. Choosing the finite-state framework is a crucial aspect of our modeling approach that makes it easily configurable and allows to seamlessly extend this model into bigger models (Chapters 3 and 4).

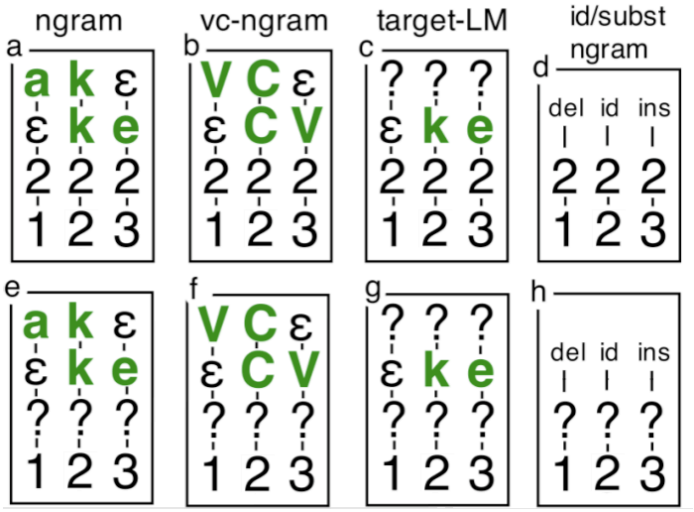

Figure 2.2: The boxes (a)-(h) represent some of the features that fire on the trigram shown in Figure 2.1. These features are explained in detail in Section 2.4.

Note that the conditional model (2.1) can be easily changed to a joint model $p_{\boldsymbol{\theta}}(x, y)$ by summing over all y' *and* x' in the denominator:

$$p_{\boldsymbol{\theta}}(x, y) = \frac{\sum_{A \in \boldsymbol{A}_{xy}} \exp \sum_i \theta_i f_i(A)}{\sum_{x', y'} \sum_{A' \in \boldsymbol{A}_{x'y'}} \exp \sum_i \theta_i f_i(A')} \tag{2.3}$$

This sum can still be efficiently computed, if the vocabulary of possible characters is not too large.

If a model $p_{\boldsymbol{\theta}}(x)$ over just *one* string is needed, the same equation (2.3) can be used, where y is set to equal the string x; the strings A just enrich x with any latent variables that we wish to include (Section 2.3), but there is no ambiguous *alignment*. In this case, the number of variables is fixed and no variable-length variables are involved, which makes it similar to conventional hidden-variable models (Quattoni et al., 2007).[2]

2.3 Latent Variables

The alignment between x and y is a latent explanatory variable that helps model the distribution $p(y \mid x)$ but is not observed in training. Other latent variables can also be useful. Morphophonological changes are often sensitive to:

- *phonemes* (whereas x and y may consist of graphemes)

[2]Practically, we still use our finite-state-based implementation for such cases; the set of allowed alignment characters is then reduced to the identity characters (a:a), (b:b) etc.

- *syllable boundaries*
- *conjugation class*
- *morpheme boundaries*
- the *position* of the change within the form.

Thus, as mentioned in Section 2.2.1, we enrich the alignment string A so that it specifies additional latent variables to which features may refer. In Figure 2.1, two latent strings are added, enabling the features in Figure 2.2(a)–(h). The first character is not just an input/output pair, but the 4-tuple $(\texttt{b} : \texttt{b} : 2 : 1)$.

Latent Word Classes. Here, ℓ_1 indicates that this form pair (*breaking / broke*) as a whole is in a particular cluster, or word class, labeled with the arbitrary number 2. Notice in Figure 2.1 that the class 2 is visible in all local windows throughout the string. It allows us to model how certain phenomena, e.g., the vowel change from ea to o, are more likely in one class than in another. Form pairs in the same class as the *breaking / broke* example might include the following Germanic verbs: *speak*, *break*, *steal*, *tear*, and *bear*.

Of course, word classes are latent (not labeled in our training data). Given x and y, $\boldsymbol{A_{xy}}$ will include alignment strings that specify class 1, and others that are identical except that they specify class 2; Equation 2.1 sums over both possibilities.[3] In a valid alignment string A, ℓ_1 must be a constant string such as 111... or 222..., as in Figure 2.1, so that it specifies a single class for the entire form pair. See sections 2.9.1.2 and 2.9.2.2 for examples of what classes were learned in our experiments.

Latent Change Regions. The latent string ℓ_2 splits the string pair into numbered regions. In a valid alignment string, the region numbers must increase throughout ℓ_2, although numbers may be skipped to permit omitted regions. To guide the model to make a useful division into regions, we also require that identity characters such as $(\texttt{b} : \texttt{b})$ fall in even regions while change characters such as $(\texttt{e} : \texttt{o})$ (substitutions, deletions, or insertions) fall in odd regions.[4] Region numbers must not increase within a sequence of consecutive changes or consecutive identities.[5] In Figure 2.1, the start of region 1 is triggered by e:o, the start of region 2 by the identity k:k, region 3 by ϵ:e.

Allowing region numbers to be skipped makes it possible to consistently assign similar labels to similar regions across different training examples. Table 2.3, for example, shows pairs that contain a vowel change in the middle, some of which contain an additional insertion of ge in the beginning (*verbinden / verbunden*, *reibt / gerieben*). We expect the model

[3] The latent class is comparable to the latent variable on the tree root symbol *S* in Matsuzaki, Miyao, and Tsujii (2005), see also Dreyer and Eisner (2006) and Petrov, Barrett, Thibaux, and Klein (2006).

[4] This strict requirement means, perhaps unfortunately, that a single region cannot accommodate the change ayc:xyz unless the two y's are not aligned to each other. It could be relaxed, however, to a prior or an initialization or learning bias.

[5] The two boundary characters #, numbered 0 and max (max=6 in our experiments), are neither changes nor identities.

to learn to label the ge insertion with a 1 and vowel change with a 3, skipping region 1 in the examples where the ge insertion is not present (see section 2.9.1.2, Analysis).

In the next section we describe features over these enriched alignment strings.

2.4 Features

One of the simplest ways of scoring a string is an n-gram model. In our log-linear model (2.1), we include n-gram features $f_i(A)$, each of which counts the occurrences in A of a particular n-gram of alignment characters. An example for such an n-gram feature is shown under ngram in Figure 2.2, box (a), on Page 22; it fires on the alignment character trigram (a:ϵ)(k:k)(ϵ:e). The log-linear framework lets us include n-gram features of different lengths, a form of backoff smoothing (Wu and Khudanpur, 2000).[6]

In addition to features that fire on specific n-grams, we include backoff features; such features can be used to capture phonological, morphological, and orthographic generalizations. In particular, we describe several classes of backoff features:

1. vc-ngram (see Figure 2.2, box b): Suppose the model is choosing whether to add s or es to pluralize a Spanish noun. It is linguistically significant whether the original noun ends in a consonant. While we could learn weights for individual bigram features which fire on words ending in specific consonants, this would overfit the training data, missing the generalization captured by a single backoff feature that fires on *any* final consonant.[7] The vc-ngram features match vowel and consonant character classes in the input and output dimensions.

2. id/subst ngram (see Figure 2.2, box d): In the id/subst ngram features, we have a similar abstraction. Character classes are here defined over input/output pairs, to match insertions, deletions, identities (matches), and substitutions (see ins, del, id, and subst in the figure). These features are useful in tasks where the input and output alphabets, Σ_x and Σ_y, are the same, like in morphological inflection and lemmatizations tasks (Sections 2.9.1 and 2.9.2), or in spelling correction, name matching or other tasks.[8] In such tasks, identities between input and output are observed very frequently; consider an English lemmatization example like *walked/walk*.

3. target LM (see Figure 2.2, box c): In string transduction tasks, it is helpful to include a language model of the target. While this can be done by mixing the transduction

[6] In effect, the higher-order weights serve as corrections to the lower-order weights. Since regularized training (see Equation (2.4) on page 34) exerts pressure to keep all weights small, these higher-order additive corrections will be close to 0 except where really necessary to explain the data. In particular, they will be close to 0 for rare n-grams, giving a backoff effect.

[7] The regularization in Equation 2.4 prefers economical explanations, which have fewer features. Therefore, it will prefer making the generalization if the model includes the appropriate backoff feature.

[8] But not in transliteration (Section 2.9.3 on page 49).

model with a separate language model, it is desirable to include a target language model directly within the transduction model. We accomplish this by adding features that ignore the input dimension.

We also include features which mirror features (a)-(d), see the corresponding boxes in Figure 2.2, but ignore the latent classes and/or regions (e.g., features (e)–(h)). For all features, versions of different lengths are included. While features that ignore latent classes and/or regions (or the output dimension) can be described as ignoring certain *rows* in the depiction of Figure 2.2, shorter features can be described as ignoring certain *columns*—a shorter feature ignores the leftmost columns of a long feature.

Notice that our choice of Σ only permits monotonic, 1-to-1 alignments, following Chen (2003). We may nonetheless favor the 2-to-1 alignment (ea:o) with bigram features such as (e:o)(a:ϵ). A "collapsed" version of a feature will back off from the specific alignment of the characters within a window: thus, (ea:o) is itself a feature. Collapsed versions of target language model features ignore epsilons introduced by deletions in the alignment, so that collapsed ok fires in a window that contains oϵk.

We have now described the central components that characterize our finite-state-based log-linear string pair model. Before we move on to more peripheral aspects like feature selection and pruning (Section 2.6), particular ways of training (Section 2.7) and avoiding divergence (Section 2.8), we relate our model to previous work in computational sequence (pair) modeling.

2.5 Related Work

2.5.1 Computational Sequence Modeling

Our model is different from conventional CRFs and hidden-state CRFs due to the use of unbounded structure, as discussed above (Section 2.2.2 on page 21).

In Section 2.2 on page 17 we also mentioned that our model is different from other string pair models (Sherif and Kondrak, 2007; Zhao et al., 2007; Hong, Kim, Lee, and Rim, 2009) that are inspired by phrase-based machine translation (Koehn et al., 2007), *segmenting* the input string into substrings (called chunks or phrases) that are transduced *independently*, ignoring context. We feel that such independence is inappropriate. By analogy, it would be a poor idea for a language model to score a string highly if it could be *segmented* into independently frequent n-grams. Rather, language models use overlapping n-grams, and indeed, it is the language model that rescues phrase-based machine translation (MT) from producing disjointed translations. We believe phrase-based machine translation avoids overlapping phrases in the *channel* model only because these would complicate the

modeling of reordering. But in the problems of Section 2.1, letter reordering is rare and we may assume it is local to a window.[9]

There are other string-based models, which use overlapping n-gram models over alignments. The simplest such model is string edit distance, famously introduced by Levenshtein (1966) as a distance measure based on deletions, substitutions, and insertions. Later such models have successfully been trained in generative frameworks (e.g., Ristad and Yianilos (1998), Bilenko and Mooney (2003)). However, string edit distance does not make use of contextual information; it is a unigram model.

Deligne, Yvon, and Bimbot (1995), Galescu and Allen (2001), Demberg, Schmid, and Möhler (2007), Bisani and Ney (2002), Bisani and Ney (2008) and others use higher-order generative joint n-gram models. Galescu and Allen (2001) first train an alignment model using Expectation Maximization (EM; Dempster, Laird, and Rubin, 1977) and proceed by obtaining the Viterbi alignment of the training data and training a conventional n-gram model on the alignment. Jansche and Sproat (2009) follow a similar approach. Chen (2003) uses a local log-linear grapheme-to-phoneme model, which is trained on aligned training string pairs; this alignment is repeatedly reestimated during training. Clark (2001) uses a mixture of pair HMMs for morphology. Freitag and Khadivi (2007) perform sequence alignment based on the averaged perceptron (Collins, 2002), but the features look separately at input or output of an alignment. Jiampojamarn, Kondrak, and Sherif (2007) present a many-to-many alignment algorithm and learn a discriminative linear model using overlapping features. These differ from our model in that they do not consider the alignment a hidden variable in the objective function, they use less flexible features, and do not add other latent variables.

The alignment is a hidden variable in the hidden CRF-based models (Quattoni et al., 2007) of Do, Gross, and Batzoglou (2006) and McCallum, Bellare, and Pereira (2005). In their Contralign model, Do et al. (2006) use discriminatively trained CRFs for protein sequence alignment. These models are related to ours in that they model string pairs and their hidden alignments in a globally normalized log-linear framework. Differences from our model are that they do not use any other latent variables, their finite-state topology typically consists of less than ten states and they do not have to sum over output strings, since input and output are always given.

McCallum et al. (2005) also define a globally normalized log-linear model over (hidden) alignments. Their model has a structure similar to ours, including the use of a binary *class*, like our conjugation classes. A major difference in our work is that we use hidden classes, which we marginalize over in decoding, whereas classes in McCallum et al. (2005) are observed and their assignment is part of the task.

The fact that in both these models the input and output strings are always given makes them fundamentally different from ours, since the output string (and therefore the alignment) in our model is unbounded, as described above in Section 2.2.2 on page 21.

[9] In the context of machine translation, Mariño, Banchs, Crego, de Gispert, Lambert, Fonollosa, and Ruiz (2005) use a sequence model over phrase pairs, similar to related string transduction work described below, where all reordering is local within the phrase pairs; see also Casacuberta (2000).

Our latent conjugation classes and latent regions are novel. Demberg et al. (2007) also add some extra dimensions to the input–output string pair in a (generative) joint n-gram model, but their added dimensions are supervised, not latent.

Rumelhart and McClelland (1986) used a neural network trained with the perceptron algorithm (Rosenblatt, 1962) to learn to predict English past-tense forms from observed present-tense forms. They used trigram features to describe input and output forms, like most other methods that we have described so far (including ours). In their case, the trigrams are built over phonemes, not the orthographic forms; this is an approach that we are interested in as well, see our remarks on future work in Section 2.10 on page 51. From the standpoint of feature design and statistical modeling, our approach is quite different from Rumelhart & McClelland's model: All our features are based on an (implicitly learned) one-to-one alignment of the two strings; we can make more effective generalizations by backing off to identities versus substitutions, vowels and consonants and the target language model; we model conjugation classes, and features may refer to the latent regions, which makes them position-dependent. Rumelhart and McClelland (1986) have been criticized for rejecting linguistic rules in favor of network units and agnostic features (Pinker and Prince, 1988). We take a middle ground and argue that linguistic knowledge and any linguistic representation may be added to our model and be picked up by the learning algorithms; the conjugations classes, vowels and consonants and others are a good start, and other linguistically inspired features may turn out to be useful as well (see our discussion in Section 2.10 on page 51). From the standpoint of statistical modeling, our model is different from Rumelhart and McClelland (1986) in that it defines a proper probability distribution, which makes it suitable as a model component in other, larger joint probability models. A further difference is that we do not make any claims about being able to model human brain functionality and human language learning using n-gram features, which is a claim that Pinker and Prince (1988) vehemently criticized.

2.5.2 Morphological Theory

Our work is also anchored in part of the literature on **inflectional morphology**, where much work has been done on finding morpheme boundaries. Harris (e.g., Harris (1955)), ahead of his time, seeks to identify boundaries at positions where the predictability of the following letter is low. De Marcken (1996) segments boundaryless text into lexemes, using a Minimum Description Length (MDL) framework. Goldsmith (1997, 2001a) describes an algorithm for the unsupervised learning of stem/suffix boundaries. He identifies sets of suffix paradigms that he calls *signatures*. Examples for such sets in English are {NULL,ed,ing}, {e,ed,ing}, or {e,ed,es,ing}. Wicentowski (2002) defines prefix, stem, and suffix regions, together with their points of transition, into which word pairs of various languages can be aligned.[10] (See also Section 4.7 on page 114 for more discussion of related work in computational morphology.) The work in this chapter extends this line of

[10] We refer to his work again in Section 2.9 on page 38.

previous research by modeling word regions and their boundaries as hidden information in a unified log-linear framework.

Our work is in many aspects close to the work of Wilson and Hayes (2008), who present a computational model for phonotactics, learning to describe the permissible combinations of phonemes from data. Their model determines the well-formedness of any phonetic string by the use of log-linear, or maximum-entropy, constraints, much like the way strings are scored in our model. In their case, the different sounds of a language are described using overlapping sets of weighted features. In addition, non-local phenomena, such as vowel harmony, are modeled using a projection that transforms the phonetic string into its vowels and consonants. This is comparable to our vowel/consonant features, or could also be modeled by a string-valued variable in our graphical models over strings, described in the next chapter (Chapter 3). Wilson and Hayes (2008) also introduce a projection that functions like a metrical grid, to evaluate stress patterns.

While Wilson and Hayes (2008) is a log-linear model over single strings, we score an input and an output form by applying a log-linear finite-state transducer. This transducer is an *intersection* of several smaller transducers, each of which encodes a subset of the overall constraints. Each of these constraints can examine either form (input or output) in isolation or both forms at once, on any alignment between the two. It is important to note that all constraints are applied in parallel; they are soft constraints.

This string-pair setup is related to other approaches in morphology that apply constraints in *parallel*, as opposed to applying a *series* of ordered rewrite rules. The following paragraphs briefly contrast these two approaches[11] in relation to our work.

Ordered rewrite rules have been applied in early approaches to computational morphology (Kay and Kaplan, 1981). In this sequential approach to morphology, an input form is modified by a series of ordered rewrite rules, each of which rewrites the input string into an intermediate representation, with the last rule application resulting in the output form. Each rule operates on the output of the previous rule. In this way, one rule can create the context to which another rule then applies (feeding). Each single rewrite rule can be expressed as a finite-state transducer; all transducers are applied one after another in the given order. But it is also possible to *compose* all of the transducers to obtain a single transducer that applies the whole sequence of changes in one step (Schützenberger, 1961). Note that the composition result still applies an ordered sequence of changes; the transducers need to be composed in the order in which they apply.

In contrast, two-level morphology (Koskenniemi, 1984) operates directly on the input and output forms (given an alignment of the two), without making use of intermediate representations. Similar to our model, it is not a sequential approach; constraints are unordered and directly applied in parallel. Like in the cascaded rewrite-rule approach, the various constraints can be compiled into separate finite-state transducers, but here all of these are then *intersected*, not composed, to build one big finite-state transducer that represents and can *simultaneously* apply all constraints to the input/output alignment. Kay (1987) stresses that

[11] See also Karttunen and Beesley (2005) for an overview.

two-level, finite-state approaches are ideal for modeling non-concatenative morphology. The approaches described so far use *hard* constraints, i.e. the transducers are unweighted and constraints may not be violated.

Optimality Theory (OT; Prince and Smolensky, 1993) removes this restriction: Here, not all constraints are equal, but there is a *ranking* among them; lower-ranked constraints may be violated by higher-ranked ones. For example, if a lower-order constraint postulates that the input should be similar to the output (*faithfulness*), it may be violated by a higher-ranked one that, for example, prefers the output to have a certain suffix that the input does not have. In our model, this would correspond to the *identity* features versus the *target language* features (Section 2.4), which have been trained on data to have different weights. OT does not use weights, but operates based on rankings alone. Higher-order constraints may violate lower-order ones, but not vice versa: No number of matching lower-ranked constraints can outweigh the violation of a single higher-ranked constraint.

In a log-linear model like the one we use, where each constraint (feature) is assigned a numerical weight (Equation 2.1), the OT ranking scheme could be expressed using the following weights: Given n constraints $f_{i...n}$ sorted from lower to higher ranking, the weight w_i of each constraint f_i is 2^i. This ensures that $\forall i < j, (\sum_{k=i}^{j} w_k) < w_{j+1}$. However, this works only if each constraint is *binary* over the whole sequence, so it cannot be counted multiple times, like in log-linear models like ours. Ranking versus weighting schemes are further discussed in (Hayes and Wilson, 2008).

An important connection to our work is that the set of constraints is not applied in any particular order, but in parallel. In OT and in the other approaches described, many constraints can be expressed as finite-state transducers (Eisner, 2002a). OT models are often trained discriminatively, using an online learning approach that repeatedly modifies the constraint ranking based on the performance of constraints on training data examples (Tesar and Smolensky, 2000). OT has no intermediate representations to work on, but see sympathy theory (McCarthy, 2002).

This concludes the comparison of our work to previous work in computational morphology. To summarize, we chose an approach that is based on two-level morphology and optimality theory (OT) in that transducers are intersected to form a single scoring machine $U_{\boldsymbol{\theta}}$, which applies constraints in parallel, not sequentially. We note that it would also be possible to learn rule-based *sequential* morphology in a log-linear finite-state setting like ours. One possibility would be to use transformation-based learning (TBL; Brill, 1992): At each iteration of training, a set of rewrite rule candidates is evaluated and the best one is applied. Evaluation in TBL is done in an error-driven way; the rule is chosen that minimizes the error of the current forms compared to the desired output forms.[12] Another possibility would be to define a generic transducer that, given a particular weight vector, can be specified to perform a certain rewrite rule. At the beginning of training, one specifies the number n of serial rewrites and composes n copies of the generic transducer. The weights

[12] Jongejan and Dalianis (2009) describe a related approach, which extracts rewrite rules from the training pairs and applies them in certain order to an inflected word form. Each rewrite rule may be interpreted as a small finite-state transducer.

in these composed transducers are then optimized jointly, to maximize the probability of observed inputs and outputs. Our approach to model inflectional paradigms in Chapter 3 (Page 58) resembles such an approach: Several transducers are learned jointly, although we do not work with completely hidden forms, and we do not restrict our models to be a linear chain leading from an input string to an output string; instead we are interested in predicting more than two strings.

In this chapter, so far we have described the central parts of our string transduction method and put them in relation to other work in this area. We will now describe several more practical aspects before turning to the experiments in Section 2.9 on page 38.

2.6 Feature Selection and Pruning

2.6.1 Motivation

This section describes a simple feature selection method. In theory, *all* features up to a certain length could be used in the model, but that would typically be too expensive. To understand why, consider the topology of the finite-state machine $U_{\boldsymbol{\theta}}$, which relates input to output strings and assigns scores according to the features that fire on possible alignments between them. All our features have n-gram structure, as just described in Section 2.4, so $U_{\boldsymbol{\theta}}$ has the well-known n-gram topology (Allauzen, Mohri, and Roark, 2003; Roark, Saraclar, Collins, and Johnson, 2004b), where each state represents an n-gram history $h = c_{i-n+1}, ..., c_{i-1}$ and has one outgoing arc per symbol c_i, representing the n-gram hc. In our case, each c is an alignment character (in the literature, n-grams are typically over *words* instead of alignment characters). The size of $U_{\boldsymbol{\theta}}$ is determined by n, the size of the input and output alphabets, the number of latent regions and the number of latent conjugation classes. Assume, for simplicity, that the input and the output alphabets, Σ_x and Σ_y, have the same size V. We need V^{2+n-1} states to represent all histories for an n-gram model, if there are no latent annotations. With r latent regions, we need $O((V^2 r)^{n-1})$ states, times a constant factor for any latent conjugation classes. Typically, V is about 30 (the number of letters in a language), so for a trigram model using 6 latent regions the number of states would be in the order of 10 million and the number of arcs in the order of 100 billion.

We will now present a simple feature selection and pruning method that will result in smaller finite-state machines to represent our model. The method relies on count cutoffs on an initial alignment of the training input and output sequences. We are also interested in other feature selection methods, including the use of L_1 regularization (Tsuruoka, Tsujii, and Ananiadou, 2009) or grafting (Perkins, Lacker, and Theiler, 2003), which we regard as opportunity for future work (see Section 2.10).

2.6.2 Pruning of Alignment Characters and Backing off Histories

The simple solution adopted in (Dreyer et al., 2008) and in this thesis reduces the number of states and arcs in the finite-state machine $U_{\boldsymbol{\theta}}$ and the number of features in the model.[13] It is motivated by the fact that

1. many character-to-character alignments can reasonably be ruled out and pruned away (e.g., aligning a 'k' in the input with an 'e' in the output or many other alignments in an English lemmatization task), and

2. many longer n-gram features may be left out in favor of shorter ones, e.g., the trigram feature abc may not add much information compared to just using ab and bc.

These two steps are now described.

First, we select a set of plausible alignment characters, based on an initial alignment of the training data without latent annotations. This selection is done after the first stage of training, see more details in the section about our staged training process (Section 2.7 on page 34). N-grams that contain alignment characters not in that set are excluded (pruned) from the model, i.e., their weights are effectively fixed to $-\infty$. In an English lemmatization task, for example, we selected about 50 plausible alignment characters, which excluded implausible ones like (k:e) or (y:f), and others. This pruning step greatly reduces the number of states and arcs in $U_{\boldsymbol{\theta}}$.

Second, we select a set $N = N_0 \cup N_1$ of ngram features to include in our model. N_0 contains all n-grams up to a certain length[14] that we find by aligning the training data in all possible ways—under the constraint of the pruned alignment alphabet.[15] Many of these ngram features are expected to be important and receive high weights during training.[16] N_1 contains other n-grams that can be included relatively cheaply; these are all n-grams that share the same history as the n-grams in N_0. Recall that $U_{\boldsymbol{\theta}}$ contains one state per n-gram history, where the history h of an n-gram is the sequence of elements leading up to the last element in the n-gram. Once an n-gram with history h is already included, all other possible n-grams with the same history can be included as well without adding any additional state, by adding arcs to the state that represents h. Note that N_1 by definition also contains all alignment character unigrams from the set of pruned alignment characters, since they all share the empty history.

The previously described, second, step is not a pruning step. All n-grams that are not in the set N will still be matched—using shorter n-grams (unless they contain one of the

[13] Discussion of feature selection was omitted in (Dreyer et al., 2008).

[14] This length is set to 3, unless otherwise noted.

[15] In addition, the alignments of the training data can be pruned using standard posterior pruning, according to the scores that the initial, first-stage model assigns to the alignments. This was done to speed up the transliteration experiments, see Section 2.9.3 on page 49. Such a pruning step is also described in Section 2.6.4.

[16] Not all of these features turn out to receive high weights because typically, a small number of alignments between an input x and output y will outweigh all others.

alignment characters already ruled out in the first step). For example, if a bigram ab is not in N, then the sequence ab will be matched using the unigrams a and b.

For each ngram feature, we also include all backoff features from the vc-ngram, target LM and other feature classes. Note that adding backoff features will typically add more states and arcs to $U_{\boldsymbol{\theta}}$. To see why, suppose the specific alignment character bigram (a:x) (b:y) is included, on which we fire the bigram target LM feature xy. If, however, the alignment character bigram (c:x) (d:y), for example, is not included, then the sequence (c:x) (d:y) would be matched using unigram transitions—but on unigram transitions there cannot be a bigram target LM feature xy, which, once it is included in the model, would need to fire on that sequence too. Therefore, an additional state needs to be added that remembers the history necessary to assign the xy feature to (c:x) (d:y) and all other alignment character sequences whose output dimension is xy. Such states can be added using intersection of different finite-state machines, each of which assigns features for different backoff feature classes. This is described in Section 2.6.3.

2.6.3 Finite-state Implementation

The procedure to create $U_{\boldsymbol{\theta}}$ closely follows Allauzen, Mohri, and Roark (2005): First we use a count transducer to create a count lattice containing all n-grams from all alignments of all training examples (without latent annotations). We then convert this count lattice to an n-gram model using the algorithm MakeModel (Allauzen et al., 2005), which adds ϕ arcs[17] and bends n-gram-final arcs back to lead to the next history.[18] We then perform phi removal, which, at each state, replaces all ϕ arcs with the arcs that ϕ stands for, given the state and its other outgoing arcs, so that we will be able to include different features for these n-grams, and remove the weights (i.e., the counts). We also insert all latent annotations, if any. We call this acceptor U_0.

For each class of backoff features vc-ngram, target LM, etc., we create a copy of U_0 and map the alignment characters to their backoff versions. This maps, for example, (a:x) to (?:x) for target LM or to (V:C) for vc-ngrams. If latent annotations are present they are unchanged, e.g., mapping (a:x:1:2) to (V:C:1:2). We then insert features into this copy and map the characters back to specific characters—for example, (V:C:1:2) to (a:c:1:2), (a:d:1:2), ..., (u:z:1:2). Then, after inserting the ngram features into U_0 as well, we intersect all FSAs to create $U_{\boldsymbol{\theta}}$. Each of the finite-state acceptors stores its features on its arcs, using the expectation semiring (Eisner, 2001, 2002b; Li and Eisner, 2009b), so the arcs in the intersection contain the combined feature vectors of the matching arcs of the single acceptors.

[17] A ϕ arc matches any symbol that cannot be matched by other arcs leading out of the same state. When it is traversed, no symbol is consumed (in that sense, it behaves like ϵ). In n-gram machines, ϕ arcs are used to back off to states that represent a smaller history if there is no match at the current state, see Allauzen et al. (2005).

[18] For example, if the count lattice contains n-grams up to length 3, then the z arc of a trigram xyz is directed back to the state that represents the history yz.

We used the OpenFst library (Allauzen, Riley, Schalkwyk, Skut, and Mohri, 2007) to implement all finite-state operations.

2.6.4 Variation

In addition to the method described above, we experimented with a slight variation that turned out to produce smaller models while retaining similar accuracies.

In this variation, we also use n-grams that we observe on initial alignments of the training data (N_0) and additional n-grams that can be added without increasing the state space of $U_{\boldsymbol{\theta}}$ (N_1), and we also prune the space of permitted alignment characters. But there are some modifications: (1) When we select n-grams for N_0, we do not yet use any pruning of alignment characters. Instead, we just use posterior pruning on the alignments of the training input/output pairs under the initial, first-stage model. The n-grams in N_1 then are, as before, added under the constraint that implausible alignment characters are not permitted. (2) To obtain plausible alignment characters we use a stricter criterion than the one described in Section 2.6.2: We again obtain unigram counts of all alignment characters in the initial alignment of the training data (i.e., after stage 1 in Section 2.7 on the next page). But now we do not just accept every alignment character with non-zero count. Instead we compute the conditional probabilities of each alignment character $(\mathtt{a} : \mathtt{x})$ given its input symbol $\mathtt{a}$, and for each $a \in \Sigma_x$ we only keep enough alignment characters to cover 99% of the conditional probability mass.[19] For example, if we observe the alignment characters (i:x), (i:y) and (i:z) with respective counts of 90, 10, and 1 on the initial alignment of training data, we prune the alignment character (i:z) from the model because its probability is less than 1 percent. In other words, more than 99% of the time, i is aligned to x or y.

We used this variation in all experiments in Section 2.9.2 on page 41 (lemmatization), since we found that otherwise (i.e., using the method in Section 2.6.2), training was too time-consuming on some of the lemmatization datasets. Especially on noisy datasets, that method results in big models since there will be many very infrequently used alignment characters due to misalignments.[20] The variation described in this subsection alleviates this problem by being more selective when deciding about permissible alignment characters.

[19] This method is asymmetric and, therefore, makes sense only for conditional models. One could symmetrize it by adding frequent alignment characters per output symbol as well.

[20] An example for a noisy training example would be (hereinkam, kommen) instead if (kam, kommen), where the verb prefix *herein* was left out in the second form. This results in many deletion characters that appear nowhere else on training data and may reasonably be ruled out as implausible alignment characters.

2.7 Training and decoding

We train $\boldsymbol{\theta}$ to maximize the regularized[21] conditional log-likelihood[22]

$$\sum_{(x,y^*)\in\mathcal{C}} \log p_{\boldsymbol{\theta}}(y^* \mid x) - ||\boldsymbol{\theta}||^2/2\sigma^2, \tag{2.4}$$

where $\mathcal{C}$ is a supervised training corpus consisting of string pairs. To maximize (2.4) during training, we apply the gradient-based optimization method L-BFGS (Liu and Nocedal, 1989b). In Section 2.8 on the next page we note that our training objective may diverge, which makes it difficult to train. We explain in Section 2.8.2 on page 37 how L-BFGS can be used to train such potentially divergent objective functions.

To decode a test example x, we wish to find $\hat{y} = \text{argmax}_{y\in\Sigma_y^*} p_{\boldsymbol{\theta}}(y \mid x)$. Constructively, $\hat{y}$ is the highest-probability string in the WFSA $T[x]$, where $T = \pi_x^{-1} \circ U_{\boldsymbol{\theta}} \circ \pi_y$ is the trained transducer that maps x nondeterministically to y. Alas, it is NP-hard to find the highest-probability string in a WFSA, even an acyclic one (Casacuberta and Higuera, 2000). The problem is that the probability of each string y is a sum over many paths in $T[x]$ that reflect different alignments of y to x. Although it is straightforward to use a determinization construction (Mohri, 1997)[23] to collapse these down to a single path per y (so that $\hat{y}$ is easily read off the single best path), determinization can increase the WFSA's size exponentially. We approximate by pruning $T[x]$ back to its 1000-best paths before we determinize.[24]

Since the alignments, classes and regions are not observed in $\mathcal{C}$, we do not enjoy the convex objective function of fully-supervised log-linear models. Training Equation 2.4, therefore, converges only to some *local* maximum that depends on the starting point in parameter space. To find a good starting point we employ **staged training**, a technique in which several models of ascending complexity are trained consecutively. The parameters of each more complex model are initialized with the trained parameters of the previous simpler model.

Our training is done in four **stages**. All weights are initialized to zero.

1. We first train only features that fire on unigrams of alignment characters, ignoring features that examine the latent strings or backed-off versions of the alignment characters (such as vowel/consonant or target language model features). The resulting model is equivalent to weighted edit distance (Ristad and Yianilos, 1998).

[21] The variance σ^2 of the L_2 prior is chosen by optimizing on development data.

[22] Alternatives would include faster error-driven methods, such as perceptron (Rosenblatt, 1962) or MIRA (Margin Infused Relaxed Algorithm; Crammer and Singer, 2003), and the slower Maximum margin Markov (M^3) networks (Taskar, Guestrin, and Koller, 2004).

[23] Weighted determinization is not always possible, but it is in our case because our limit to k consecutive insertions guarantees that $T[x]$ is acyclic.

[24] This value is high enough; we see no degradations in performance if we use only 100 or even 10 best paths. Below that, performance starts to drop slightly.

2. Next,[25] we add higher-order n-grams of alignment characters, but no backed-off features or features that refer to latent strings.

3. Next, we add backed-off versions (vowel/consonant, target LM, etc.) of the features from stage 2 as well as all collapsed features.

4. Finally, we add all features that look at the latent classes or regions. In our experiments, we permitted latent classes `1-2` and regions `0-6`.

Staged training helps finding better maxima in training, which is illustrated in Figure 2.3. The grey line shows the objective function during training with all features immediately active after stage 1; the black line shows the objective function during our 4-stage training process. It converges to a better maximum.

For some experiments (namely all lemmatization experiments in Section 2.9.2), we reverse the order of stages 3 and 4. We observed that this sped up training. Latent variables take many iterations to train (we typically cut off after 100 iterations), and each iteration is faster if backoff features are not present yet (see Section 2.6.2). The last stage then, which adds the backoff features and is most expensive to run, can be run for fewer iterations (we cut off after 50 iterations) because they are supervised and less numerous.

2.8 Avoiding Divergence of the Probability Model

We noted in Sections 2.2.2 and 2.5 that an important difference of our model from the models of (Lafferty et al., 2001b) and others is that it handles sequences of arbitrary lengths and not just fixed-size vectors of input/output variables. This section deals with a crucial implication of this modeling domain.

Since the possible output strings y' in the denominator in Equation 2.1 may have arbitrary length, the summation over alignment strings may be *infinite*. Thus, for some values of $\boldsymbol{\theta}$, the sum in the denominator diverges and the probability distribution is undefined. This is the case when $\boldsymbol{\theta}$ does not penalize or even rewards forming an output string y by inserting more and more characters into the input x, i.e. the WFSA $U_{\boldsymbol{\theta}}(x, y)$ contains one or more divergent cycles. To model $p_{\boldsymbol{\theta}}(y \mid x)$, we have to introduce a mechanism to avoid and disallow any $\boldsymbol{\theta}$ that would give rise to such undefined probability distributions.

This problem can occur in all prediction tasks in natural language processing that involve generating structured output, such as sequences or trees, whose length or depth are not known in advance. In machine translation, we wish to find the best translation, in pronunciation modeling the best phoneme sequence, in parsing the best tree structure. In all these cases, we often wish to to allow *insertion* of structure, since the desired outputs

[25] When unclamping a feature at the start of stages (2)–(4), we initialize it to a random value from $[-0.01, 0.01]$.

may be longer or more complex than the input. But structure insertion can be problematic; if structure is allowed to be inserted repeatedly, infinite loops may occur. In finite-state machines, these may be *path cycles*; in parsing, these might be *unary rule cycles*.

Two subsections follow; the first (Section 2.8.1) describes a solution that forces the objective function to converge by imposing a constraint on certain insertions. When a hard constraint is not used, the divergence problem may occur during training: Some of the various $\boldsymbol{\theta}$ that are explored may cause the objective function to diverge on training inputs; such cases need to be detected safely (without following the infinite loops that these $\boldsymbol{\theta}$ favor).

In addition, we describe in Appendix A (Page 128) a problem where divergent parameters may be detected only at decoding time. An objective function like Equation 2.1 that was optimized on training data may still diverge for previously unseen x. We will define a modified objective function that, once optimized, is guaranteed to converge on *any* x. This is not a practical concern for the experiments in this thesis because the chosen topology of the scoring finite-state machine $U_{\boldsymbol{\theta}}$ guarantees that the output will be finite on any decoding input x.[26]

2.8.1 Insertion Limits

Authors often deal with divergence by disallowing repeated structure insertion altogether or bounding it. Finkel, Kleeman, and Manning (2008a) restrict their parsing model by adding a hard constraint that disallows multiple unary rule applications over the same span of the input sentence. In machine translation, typically an insertion that is not anchored at any input substring is disallowed as well, which may result in loss of generalization.

Similar to Finkel et al. (2008a), in Dreyer et al. (2008) we imposed a maximum on consecutive inserted letters in our finite-state machines to prevent any infinite denominator sum. This results in slightly bigger finite-state machines, depending on the actual limit imposed,[27] since the machines have to keep track of consecutive insertions. It also makes it impossible to delete an input string completely and insert a new output string, e.g., in modeling pairs like *go* and *went*.

In addition, we will see in later chapters that a simple limit on consecutive insertions will not work for some crucial cases: Sometimes the input string x is not given directly, but is rather defined as a possibly infinite distribution $p(x)$ over inputs—a character language model, which, just like regular language models, is naturally not length-bounded itself; in such a case, limits on consecutive insertions will still result in possibly infinite output strings, and the probability distribution diverges for some values of the parameter vector $\boldsymbol{\theta}$.

[26] In brief, due to the n-gram structure of $U_{\boldsymbol{\theta}}$, any arc labeled with ϵ on the input side is reachable from the start state and will be traversed during training. The problem occurs if arcs with ϵ on the input side are not explicitly traversed during training and may end up as negative- or zero-weight arcs due to their features. For details, see Appendix A.

[27] Dreyer et al. (2008) experimented with limits of 1–3, depending on the training data of the task at hand.

2.8.2 Detecting Divergence

We wish to compute $L_{\boldsymbol{\theta}}(x, y)$ for any $\boldsymbol{\theta}$; this includes cases where $L_{\boldsymbol{\theta}}(x, y)$ diverges. Recognizing these cases can be a challenge. Some standard pathsum algorithms (Mohri, 2002) do not converge in this case when we try to compute $Z_{\boldsymbol{\theta}}$; they simply loop around cycles until (after a very long time) the computation numerically overflows. Algorithms like Bellman-Ford, which can detect negative-cost cycles, are not the solution: When we compute the pathsum, even zero-cost cycles are harmful and will be traversed infinitely many times. And, more important, consider a finite-state machine with a single state and two self-loops each with probability $\frac{1}{2}$. The individual cycles would give convergent geometric series sums if they did not interact. But in fact so many new paths are formed by their interaction that the pathsum diverges.

To detect divergence, we propose a spectral solution, which is based on eigenvalues: We compute the largest eigenvalue of the transition matrix of the finite-state machine and report divergence iff the largest eigenvalue is 1 or greater. To compute the largest eigenvalue we use the power iteration method (Golub and Loan, 1996; Page, Brin, Motwani, and Winograd, 1999). Reporting divergence means returning $L_{\boldsymbol{\theta}}(x, y) = -\infty$ since in that case the probability of the training data is zero.

After detecting the divergent normalization pathsum the question is what the maximizer should do in such a case. Here we explore the case of the popular second-order method L-BFGS (Liu and Nocedal, 1989a). It requires a finite objective function. We ensure this by using a modified objective function,

$$L' = \frac{L}{L+1}, \tag{2.5}$$

which is always between -1 and 0, instead of $-\infty$ and 0.

Fortunately, no other changes are required for running L-BFGS with divergent objective functions: Whenever this maximization algorithm encounters the minimum value of the objective function (i.e., the divergent case) during line search it will avoid this bad region of the parameter space in the future.

Practically, we do impose an insertion limit in Stages 1 and 2 of training (see Section 2.7 on page 34), for better stability, and lift it in the following stages. Typically, fewer iterations of L-BFGS training are needed when an insertion limit is in place.

With *online* methods, such as stochastic gradient descent or MIRA, a potentially divergent objective function is harder to optimize than with L-BFGS. With these methods, a small change is made to the parameter vector after processing each training example (or small batch of training examples). Each of these small updates may cause the overall training objective to diverge, but this may not immediately be obvious. An expensive test on *all* training data needs to be run after each update to find out if the objective function still converges on all training examples. This would defeat the very purpose of doing online optimization. A solution can be to just recognize whenever the objective function does not converge on a given example, which might have been caused by some update several exam-

ples earlier. Then, manually adjust some parameter(s)[28] to force the objective function to converge on this example and continue. However, we have found that this method is very unstable, and once the objective function has been divergent, it will often become divergent again; the learner does not learn to stay away from such bad regions of the parameter space. More research is needed to find methods to reliably apply online optimization to a potentially divergent objective function.

2.9 Experiments

We evaluate our model on two tasks of morphology generation and a transliteration task. Predicting morphological forms has been shown to be useful for machine translation and other tasks.[29] Here we describe two sets of morphological experiments: an inflectional morphology task in which models are trained to transduce verbs from one form into another (Section 2.9.1), and a lemmatization task (Section 2.9.2), in which *any* inflected verb is to be reduced to its root form. The transliteration task is from the Named Entities Workshop (NEWS) 2009 shared task (see Section 2.9.3 on page 49).

2.9.1 Inflectional Morphology

We conducted several experiments on the CELEX morphological database (Baayen, Piepenbrock, and Gulikers, 1995). We arbitrarily considered mapping the following German verb forms:[30] 13SIA $\rightarrow$ 13SKE, 2PIE $\rightarrow$ 13PKE, 2PKE $\rightarrow$ z, and rP $\rightarrow$ pA.[31] We refer to these tasks as 13SIA, 2PIE, 2PKE and rP. Table 2.3 on page 40 shows some examples of regular and irregular forms. Common phenomena include stem changes (`ei:ie`), prefixes inserted after other morphemes (*abz**u**brechen*) and circumfixes (***ge**rieb**en***).

We compile lists of form pairs from CELEX. For each task, we sample 2500 data pairs without replacement, of which 500 are used for training, 1000 as development and the remaining 1000 as test data. We train and evaluate models on this data. The whole process consisting of sampling new data, splitting, training and evaluating is repeated 5 times. All results are averaged over these 5 runs.

[28] For example, increase a length penalty on each arc of the finite-state machine.

[29] E.g., Toutanova, Suzuki, and Ruopp (2008) improve MT performance by *selecting* correct morphological forms from a knowledge source. We instead focus on generalizing from observed forms and *generating* new forms (but see *with rootlist* in Table 2.5).

[30] From the available languages in CELEX (German, Dutch, and English), we selected German as the language with the most interesting morphological phenomena, leaving the multilingual comparison for the lemmatization task (Section 2.9.2), where there were previous results to compare with. The 4 German datasets were picked arbitrarily.

[31] A key to these names: 13SIA=*1st/3rd sg. ind. past*; 13SKE=*1st/3rd sg. subjunct. pres.*; 2PIE=*2nd pl. ind. pres.*; 13PKE=*1st/3rd pl. subjunct. pres.*; 2PKE=*2nd pl. subjunct. pres.*; z=*infinitive*; rP=*imperative pl.*; pA=*past part.*

	Features							Task			
	ng	vc	tlm	tlm-coll	id	lat.cl.	lat.reg.	13SIA	2PIE	2PKE	rP
ngrams	x							82.3 (.23)	88.6 (.11)	74.1 (.52)	70.1 (.66)
ngrams+x	x				x			82.8 (.21)	88.9 (.11)	74.3 (.52)	70.0 (.68)
	x		x					82.0 (.23)	88.7 (.11)	74.8 (.50)	69.8 (.67)
	x		x		x			82.5 (.22)	88.6 (.11)	74.9 (.50)	70.0 (.67)
	x		x	x				81.2 (.24)	88.7 (.11)	74.5 (.50)	68.6 (.69)
	x		x	x	x			82.5 (.22)	88.8 (.11)	74.5 (.50)	69.2 (.69)
	x	x						82.4 (.22)	88.9 (.11)	74.8 (.51)	69.9 (.68)
	x	x			x			83.0 (.21)	88.9 (.11)	74.9 (.50)	70.3 (.67)
	x	x	x					82.2 (.22)	88.8 (.11)	74.8 (.50)	70.0 (.67)
	x	x	x		x			82.9 (.21)	88.6 (.11)	75.2 (.50)	69.7 (.68)
	x	x	x	x				81.9 (.23)	88.6 (.11)	74.4 (.51)	69.1 (.68)
	x	x	x	x	x			82.8 (.21)	88.7 (.11)	74.7 (.50)	69.9 (.67)
ngrams+x +latent	x	x	x	x	x	x		84.8 (.19)	**93.6 (.06)**	75.7 (.48)	81.8 (.43)
	x	x	x	x	x		x	**87.4 (.16)**	**93.8 (.06)**	**88.0 (.28)**	83.7 (.42)
	x	x	x	x	x	x	x	**87.5 (.16)**	93.4 (.07)	**87.4 (.28)**	**84.9 (.39)**
Moses3								73.9 (.40)	92.0 (.09)	67.1 (.70)	67.6 (.77)
Moses9								85.0 (.21)	**94.0 (.06)**	82.3 (.31)	70.8 (.67)
Moses15								85.3 (.21)	**94.0 (.06)**	82.8 (.30)	70.8 (.67)

Table 2.2: Exact-match accuracy (average edit distance) versus the correct answer on the German inflection task, using different feature sets. See page 39 for more explanations.

Table 2.2 and Figure 2.4 on page 57 report results after stages ②, ③, and ④ of training, which include successively larger feature sets. These are respectively labeled *ngrams*, *ngrams+x*, and *ngrams+x+latent*. In Table 2.2, the last row in each section shows the full feature set at that stage (cf. Figure 2.4), while earlier rows test feature subsets. The highest n-gram order used is 3, except for *Moses9* and *Moses15* which examine windows of up to 9 and 15 characters, respectively, see below. We mark in the table in **bold** the best result for each dataset, along with all results that are statistically indistinguishable (paired permutation test, $p < 0.05$).

Our baseline is **Moses** (Koehn et al., 2007), the popular toolkit for statistical machine translation (SMT), run over letter strings rather than word strings. It is trained (on the same data splits) to find substring-to-substring phrase pairs and translate from one form into another (with phrase reordering turned off). Results reported as *moses3* are obtained from Moses runs that are constrained to the same context windows that our models use, so the maximum phrase length and the order of the target language model were set to 3. We also report results using much larger windows, labeled *moses9* and *moses15*.

13SIA	lieb**te**, pick**te**, red**ete**, r**ieb**, tr**ieb**, zu**zog**
13SKE	lieb**e**, pick**e**, red**e**, r**eibe**, tr**eibe**, zu**ziehe**
2PIE	lieb**t**, pick**t**, red**et**, reib**t**, treib**t**, zuzieh**t**
13PKE	lieb**en**, pick**en**, red**en**, reib**en**, treib**en**, zuzieh**en**
2PKE	abbreche**t**, entgegentrete**t**, zuziehe**t**
z	ab**zu**breche**n**, entgegen**zu**trete**n**, zu**zu**zieh**en**
rP	redet, r**eibt**, tr**eibt**, verb**i**ndet, überfischt
pA	**ge**redet, **ge**r**ieben**, **ge**tr**ieben**, verb**u**nd**en**, überfischt

Table 2.3: CELEX forms used in our experiments. Changes from one form to the other are in bold (information not given in training). The changes from rP to pA are very complex. Note also the differing positions of `zu` in z. See also the complete table of forms on page 141.

2.9.1.1 Results

The results in Table 2.2 on the preceding page show that including *latent classes and/or regions* improves the results dramatically. Comparing the last line in *ngrams+x* to the last line in *ngrams+x+latent*, the accuracy numbers improve from 82.8 to 87.5 (13SIA), from 88.7 to 93.4 (2PIE), from 74.7 to 87.4 (2PKE), and from 69.9 to 84.9 (rP).[32] This shows that error reductions between 27% and 50% were reached.

On 3 of 4 tasks, even our simplest *ngrams* method beats the *moses3* method that looks at the same amount of context.[33] With our full model, in particular using latent features, we always outperform *moses3*—and even outperform *moses15* on 3 of the 4 datasets, reducing the error rate by up to 48.3% (rP). On the fourth task (2PIE), our method and *moses15* are statistically tied. *Moses15* has access to context windows of five times the size than we allowed our methods in our experiments.

While the gains from *backoff features* in Table 2.2 were modest (significant gains only on 13SIA), the learning curve in Figure 2.4 suggests that they were helpful for smaller training sets on 2PKE (see *ngrams* vs *ngrams+x* on 50 and 100) and helped consistently over different amounts of training data for 13SIA.

2.9.1.2 Analysis

The types of errors that our system (and the moses baseline) make differ from task to task. We will describe the errors on the complex rP task, as this is the most interesting and diverse one. Here, most errors come from wrongly copying the input to the output, without making a change (40-50% of the errors in all models, except for our model with latent classes and no regions, where it accounts for only 30% of the errors). This is so common

[32] All claims in the text are statistically significant under a paired permutation test ($p < .05$).

[33] This bears out our contention in Section 2.5 that a "segmenting" channel model is damaging. Moses cannot fully recover by using overlapping windows in the *language* model.

because about half of the training examples contain identical inputs and outputs (as in the imperative *berechnet* and the participle *(ihr habt) berechnet*). Another common error is to wrongly assume a regular conjugation (just insert the prefix *ge-* at the beginning). Interestingly, this error by simplification is more common in the Moses models (44% of moses3 errors, down to 40% for moses15) than in our models, where it accounts for 37% of the errors of our *ngrams* model and only 19% if latent classes *or* latent regions are used; however, it goes up to 27% if both latent classes and regions are used.[34] All models for rP contain errors where wrong analogies to observed words are made (*verschweisst/verschwissen* in analogy to the observed *durchweicht/durchwichen*, or *bebt/geboben* in analogy to *hebt/gehoben*). In the 2PKE task, most errors result from inserting the *zu* morpheme at a wrong place or inserting two of them, which is always wrong. This error type was greatly reduced by latent regions, which can discover different parameters for different positions, making it easier to identify where to insert the *zu*.

Analysis of the 2 latent classes (when used) shows that a *split into regular and irregular conjugations* has been learned. For the rP task we compute, for each data pair in development data, the posterior probabilities of membership in one or the other class. 98% of the regular forms, in which the past participle is built with *ge- ... -t*, fall into one class, which in turn consists nearly exclusively (96%) of these forms. Different irregular forms are lumped into the other class.

The learned regions are consistent across different pairs. On development data for the rP task, 94.3% of all regions that are labeled 1 are the insertion sequence (ϵ:ge), region 3 consists of vowel changes 93.7% of the time; region 5 represents the typical suffixes (t : en), (et : en), (t : n) (92.7%). In the *2PKE* task, region 0 contains different prefixes (e.g., *entgegen* in *entgegenzutreten*), regions 1 and 2 are empty, region 3 contains the *zu* affix, region 4 the stem, and region 5 contains the suffix.

The pruned alignment alphabet excluded a few gold standard outputs so that the model contains paths for 98.9%–99.9% of the test examples.

Note that the results and analysis in this section (2.9.1) are taken from (Dreyer et al., 2008), where a limit on consecutive insertions was used.[35] We reran the experiments without such a limit and obtained better final results in 3 of the inflection tasks (accuracy 94.7% versus 93.4% for 2PIE, 88.8% versus 87.4% for 2PKE, 87.0% versus 84.9% for rP) and a worse result on one task (86.0% versus 87.5% for 13SIA).

2.9.2 Lemmatization

Next, we apply our models to the task of lemmatization, where the goal is to generate the lemma given an inflected word form as input. We compare our model to a state-of-the-art supervised approach, Wicentowski (2002, chapter 3). Wicentowski's *Base* model

[34] We suspect that training of the models that use classes and regions together was hurt by the increased non-convexity; annealing or better initialization might help.

[35] The limit was set to 3.

simply learns how to replace an arbitrarily long suffix string of an input word, choosing some previously observed suffix → suffix replacement based on the input word's final n characters (interpolating across different values of n). His *Affix* model essentially applies the Base model after stripping canonical prefixes and suffixes (given by a user-supplied list) from the input and output. Finally, his *WFAffix* uses similar methods to also learn substring replacements for a stem vowel cluster and other linguistically significant regions in the form (identified by a *deterministic* alignment and segmentation of training pairs). This approach could be characterized as supervised change regions combined with region-independent phrase pairs similar to Moses' approach (Koehn et al., 2007).

We compare against all three models. Note that *Affix* and *WFAffix* have an advantage that our models do not, namely, user-supplied lists of canonical affixes for each language. It is interesting to see how our models with their more non-committal trigram structure compare to this. We obtained the datasets for 15 languages used in (Wicentowski, 2002) and report results in Table 2.5 on page 45.[36] Following Wicentowski, 10-fold cross-validation was used. The columns *n+l* and *n+l+x* mean *ngram and latent variables* and *ngram and latent variables and backoff features*, respectively.[37] As latent variables, we include 2 word classes and 6 change regions.[38]

For completeness, we also evaluate our models on a *selection* (rather than a generation) task and compare to Wicentowski (2002), see Table 2.6. Here, at test time, the lemma is not generated, but selected from a list of allowed lemmas for the given language. This suppresses nonsensical forms that may otherwise be proposed by the model, improving the performance of the model. The list of allowed lemmas simply consists of the lemmas observed during training. Note that this selection trick is possible for lemmatization (but not for most other string transduction tasks) since many inflected forms map to the same lemma. So for a given lemma, most inflected forms have typically been seen in training and the remaining ones occur as test inputs, so that their lemma indeed occurs in training data, if with other inflected input forms.

2.9.2.1 Results

On the supervised generation task *without root list*, our models outperform Wicentowski (2002) on 14 of the 15 languages, often by a very large margin (Table 2.5). On 10 of the 15 languages, our full model (column *n+l+x*), which uses backoff features, latent classes and regions, reduces the number of errors (number of output forms that contain one or more incorrect characters) by more than 50%, compared to Wicentowski's best model. The biggest single improvement is on Irish, where we reduce the error by 92.1%. Our full model has an accuracy of 90% or greater on 11 languages, whereas for Wicentowski's models, this is the case only for 3 of the 15 languages.

[36] (Wicentowski, 2002) contains results on more languages. Since our models are expensive to train we chose to train and decode on a representative subset of 15 languages only.

[37] The order of training stages for these experiments was explained on page 35.

[38] The insertion limit k in the early stages of training was set to 3 for German and 2 for the other languages.

Language	Size
Arabic	3296
Basque	5843
Czech	23818
Dutch	5768
English	79909
French	63559
German	14120
Greek	201
Irish	2160
Latin	26818
Norwegian	2489
Polish	23731
Russian	3076
Spanish	57224
Tagalog	10099

Table 2.4: Numbers of available inflection-root pairs for the various languages.

Our performance on Greek is an outlier. Here, Wicentowski's best model has an accuracy of 85.9%, whereas the performances of our models is just below 60%. This may be due to the very small data size (only 200 Greek forms overall, see Table 2.4). Presumably the user-supplied affix lists that Wicentowski's WFAffix model uses adds valuable information that is not easily or reliably extractable from the small training data. This would also explain why Wicentowski's Base model here has unusually low accuracy (14%).

Language	Model	All	Regular	Semi-Regular	Irregular	Obsolete/Other
Dutch	n	88.9	93.7	88.1	67.6	-
	n+l	91.4	95.6	91.4	71.7	-
	n+l+x	91.4	95.7	90.4	73.4	-
English	n	91.5	94.5	81.8	18.6	100.0
	n+l	95.0	96.9	92.3	23.3	100.0
	n+l+x	95.8	97.7	93.9	20.9	100.0
German	n	84.9	85.9	92.9	75.7	77.8
	n+l	94.0	95.4	97.3	85.0	81.5
	n+l+x	96.7	97.9	96.9	89.7	96.3
Irish	n	94.4	97.1	94.7	69.7	-
	n+l	97.4	99.2	98.3	72.7	-
	n+l+x	97.7	99.2	98.6	75.8	-
Latin	n	83.7	89.7	80.7	86.7	7.1

(continued on the next page)

Table 2.7: (continued)

	n+l	93.6	94.8	78.5	89.4	93.4
	n+l+x	94.4	95.4	86.6	90.0	93.6
Spanish	n	91.3	92.5	90.3	75.2	89.4
	n+l	94.0	95.1	92.4	79.9	92.3
	n+l+x	94.3	95.3	92.3	82.3	93.7

Table 2.7: Lemmatization (generation task, no root list). Results are listed in whole-word accuracy, for the languages where information about regular and irregular verbs was available. Comparative results are not available in (Wicentowski, 2002), except for the *All* column; the numbers in *All* are similar to the results in Table 2.5. For the corresponding *selection* task results, see Table 2.8.

Comparing our own models, it is noteworthy that the model that uses latent classes and regions (*n*+*l*) often gives much higher accuracy than our basic model (*n*) that just uses n-gram features. Our basic model has a performance greater than 90% on 5 of 15 languages, whereas the models with latent variables perform that high on 11 languages, as noted above. Often, the accuracy increases by more than 5 absolute percentage points. This is the case on Arabic, Basque, French, German, Latin, Russian and Tagalog. On some languages, on the other hand, the latent variables did not help much (Czech, Norwegian) or even hurt accuracy (but only on Greek, discussed above). The backoff features (target LM, id/subst ngram) always improved the performance; in 8 of the 15 languages this improvement was significant. Overall, the average edit distance numbers (in Table 2.5 as well) show the same trends as the accuracy numbers.

We also measure how well our models perform on regular versus irregular words (Table 2.7). The classification of (inflection, root) pairs into regular, irregular, semi-regular and other is taken from (Wicentowski, 2002).[39] We expected and observe here that, as we add latent variables as well as backoff features, the accuracy on irregular verbs improves, sometimes quite dramatically, while the performance on regular verbs often increases only slightly. For German and Latin, however, the performance increases dramatically even for regular verbs. This is due to the fact that in these languages, even regular conjugation is hard and benefits from the additional features. In German, for example, some affixes are in general confusable with stem parts so that the model sometimes incorrectly removes stem

[39]Of the 15 languages we picked for this thesis, 6 were annotated with regularity information; these are shown in the table. There is no comparison with Wicentowski's numbers in this table since they are not available, except for the *All* column, which corresponds to information in Table 2.5.

Language	Wicentowski (2002) Base	Af.	WFA.	This work n	n+l	n+l+x	% err. red.
Arabic	-	-	-	65.9 (0.45)	**79.4 (0.30)**	**79.7 (0.30)**	-
Basque	85.3	81.2	80.1	88.3 (0.26)	**94.8 (0.12)**	**95.8 (0.10)**	71.4
Czech	72.3	85.1	85.2	**95.4 (0.07)**	95.4 (0.09)	95.4 (0.09)	68.9
Dutch	79.7	74.2	58.2	88.9 (0.20)	**91.4 (0.17)**	**91.4 (0.17)**	57.6
English	91.0	94.7	93.1	91.5 (0.10)	95.0 (0.07)	**95.8 (0.06)**	20.8
French	85.8	91.9	90.4	87.0 (0.20)	95.9 (0.07)	**97.7 (0.04)**	71.6
German	87.7	-	84.6	84.9 (0.19)	94.0 (0.09)	**96.7 (0.06)**	73.2
Greek	14.1	-	**85.9**	59.2 (0.65)	57.1 (0.83)	58.2 (0.81)	-196.5
Irish	43.3	-	70.8	94.4 (0.09)	**97.4 (0.06)**	**97.7 (0.04)**	92.1
Latin	78.0	-	69.4	83.7 (0.29)	93.6 (0.08)	**94.4 (0.08)**	74.5
Norwegian	82.5	-	80.4	84.5 (0.22)	**85.9 (0.20)**	**85.8 (0.21)**	18.9
Polish	93.3	-	92.3	93.9 (0.08)	96.7 (0.05)	**97.2 (0.04)**	58.2
Russian	77.9	-	67.3	83.8 (0.23)	**89.5 (0.18)**	**89.8 (0.18)**	53.8
Spanish	89.9	89.3	86.5	91.3 (0.12)	94.0 (0.09)	**94.3 (0.08)**	43.6
Tagalog	0.3	80.3	81.7	82.1 (0.31)	94.7 (0.09)	**95.5 (0.07)**	75.4

Table 2.5: Lemmatization (generation task, no rootlist): Exact-match accuracy and average edit distance (the latter in parentheses) on 15 languages. The numbers from Wicentowski (2002) are for his Base, Affix and WFAffix models. The numbers for our models are for ngram features (n), ngram features with latent annotations (n+l) and added backoff features (n+l+x). The best result per language is in **bold** (as are statistically indistinguishable results when we can do the comparison, i.e., for our own models). Last column: relative error reduction of our n+l+x versus Wicentowski's best model.

Language	Wicentowski (2002) Base	Af.	WFA.	This work n	n+l	n+l+x	% err. red.
Arabic	-	-	-	**72.3 (1.29)**	72.4 (1.36)	72.3 (1.37)	-
Basque	94.5	94.0	**95.0**	90.7 (0.31)	90.8 (0.35)	90.8 (0.35)	-84.0
Czech	78.7	**98.2**	**98.2**	88.5 (0.27)	88.0 (0.39)	88.0 (0.39)	-566.7
Dutch	86.4	93.8	79.2	**97.8 (0.08)**	97.5 (0.10)	97.3 (0.10)	56.5
English	98.3	98.7	98.5	**98.8 (0.03)**	98.3 (0.07)	98.3 (0.07)	-30.8
French	99.0	99.3	98.7	99.7 (0.01)	99.8 (0.01)	**99.9 (0.00)**	85.7
German	92.0	92.1	94.7	**99.0 (0.03)**	**99.4 (0.03)**	**99.5 (0.03)**	90.6
Greek	15.6	**99.5**	91.2	93.9 (0.31)	92.9 (0.36)	92.9 (0.36)	-1320.0
Irish	43.9	-	89.1	**99.7 (0.02)**	**99.9 (0.01)**	**99.9 (0.01)**	99.1
Latin	88.5	88.5	83.9	99.7 (0.01)	**99.6 (0.01)**	**99.7 (0.01)**	97.4
Norwegian	93.7	-	95.5	97.9 (0.09)	**98.2 (0.08)**	**98.2 (0.08)**	60.0
Polish	97.2	97.2	97.1	**99.5 (0.01)**	**99.6 (0.01)**	**99.6 (0.01)**	85.7
Russian	85.8	-	85.2	94.8 (0.12)	**96.5 (0.08)**	95.9 (0.10)	71.1
Spanish	94.6	96.5	95.3	**99.8 (0.00)**	99.8 (0.01)	99.8 (0.01)	94.3
Tagalog	0.8	91.8	96.0	97.0 (0.07)	**98.0 (0.05)**	**98.1 (0.05)**	52.5

Table 2.6: Lemmatization (selection task, with root list) on 15 languages. The only difference to Table 2.5 on the previous page is the use of a root list.

parts in order to generate a lemma. A prediction error on the input *geben* '(we) give' illustrates this; it was transduced to *ben* 'to give' instead of the identity transduction *geben* 'to give' because typically *ge-* at the beginning of a word would be a past participle prefix. In Latin, it is also surprising that the performance on verbs classified as *Obsolete/Other* is abysmal with our basic model, but very high when we add latent variables. This is the case because most verbs in this class build their lemma using the passive form (e.g., *exsequor*) but the basic model just assigns the usual active form. These errors are fixed when conjugation classes are used.

Language	Model	All	Regular	Semi-Regular	Irregular	Obsolete/Other
Dutch	Base	86.4	85.2	95.2	69.7	-
	Affix	93.8	92.9	98.8	85.7	-
	n	97.8	98.5	99.1	91.0	-
	n+l	97.5	98.6	98.4	90.3	-
	n+l+x	97.3	98.5	97.9	90.7	-
English	Base	98.3	99.0	98.7	28.1	100.0
	Affix	98.7	99.3	99.2	32.3	100.0
	n	98.8	99.5	98.4	69.8	100.0
	n+l	98.3	99.5	98.1	44.2	100.0

(continued on the next page)

Table 2.8: (continued)

	n+l+x	98.3	99.5	98.1	44.2	100.0
German	Base	92.0	93.4	97.5	81.9	90.4
	Affix	92.1	N/A	N/A	N/A	N/A
	n	99.0	99.4	99.3	96.6	100.0
	n+l	99.4	99.8	99.1	97.4	100.0
	n+l+x	99.5	99.8	99.1	98.0	100.0
Irish	Base	43.9	95.4	20.5	0.0	-
	Affix	N/A	N/A	N/A	N/A	-
	n	99.7	100.0	100.0	93.9	-
	n+l	99.9	100.0	100.0	97.0	-
	n+l+x	99.9	100.0	100.0	97.0	-
Latin	Base	88.5	94.9	60.3	62.8	69.5
	Affix	88.5	N/A	N/A	N/A	N/A
	n	99.7	99.9	99.4	99.0	98.4
	n+l	99.6	99.8	100.0	98.9	98.4
	n+l+x	99.7	99.9	100.0	98.8	98.5
Spanish	Base	94.6	96.0	89.3	78.8	93.8
	Affix	96.5	96.9	95.0	91.0	100.0
	n	99.8	99.9	99.4	98.6	100.0
	n+l	99.8	99.9	99.3	98.3	100.0
	n+l+x	99.8	99.9	99.2	98.2	100.0

Table 2.8: Lemmatization (selection task, with root list). The only difference from Table 2.7 is that here, a root list is used. *Base* and *Affix* show results from (Wicentowski, 2002) for the same data sets.

While tables 2.5 and 2.7 show results for the *generation* task, tables 2.6 and 2.8 show results for the *selection* task on the same data, where the task is to pick the correct lemma from a specified list of known lemmas. Compared to Wicentowski (2002), our model also improves the results for most languages on this task, here on 12 out of 15 languages, comparing, for each language, our best model to Wicentowski's best model. If we always pick our *full* model as our final result then we improve over Wicentowski's results on 11 languages (see error reduction column, Table 2.6). On the languages for which we improve upon Wicentowski, the relative error reduction (compared to Wicentowski's models) is between 50% and 99%.

For all models, the performance on the selection task is higher than on the generation task, since many candidates with small spelling errors are eliminated; they are not contained

in the list of known lemmas. Our basic models (column *n*) have accuracy of 90% or greater on 13 of the 15 languages. The latent variables and backoff features did not help that much on this task due to this already-high accuracy of the basic models. The latent variable and backoff models are especially good at fixing small errors—a job that the provided list of known lemmas already does on this selection task. Interestingly, Arabic has a relatively bad edit distance result on this task, compared to the generation task in Table 2.5, even where the accuracy is actually improved. This can happen when the provided root list often does not contain the correct lemma, so the model is forced to pick a completely different lemma instead, which has a high edit distance to the truth.

For the selection task (as opposed to the generation task), Wicentowski (2002) reports separate results for regular and irregular verbs, so we can compare our models on the same data (see Table 2.8).[40] An interesting pattern arises: While our model outperforms Wicentowski's models on regular as well as on irregular verbs, the performance differences are much greater on the irregular verbs. Here, our models reach 90% or higher on 5 out of the 6 listed languages (97% or higher on 4 languages), while Wicentowski's models reach 91% only on one language (Spanish).

2.9.2.2 Error Analysis and Learned Classes

We have seen that conjugation classes helped. This is natural since the input forms for the lemmatization task can be any inflected form, so training examples by definition fall into different classes that are unannotated. Analysis of the trained model in English shows that the learner was able to recover such latent classes. We computed class membership probabilities for development data examples, like in Section 2.9.1.2. Class 2 contains 99% of all pairs whose input word ends in *ing*, such as *zipping - zip*. Class 1, on the other hand, contains most of the pairs (77.2%) whose input is equal to its output (*wish - wish*), as well as most of those whose input ends in *s* (78%), such as *yields/yield*. Other forms do not clearly belong into one or the other class. Regular past tense examples (*arm/armed*) are equally likely under each class. This analysis shows that meaningful classes can be learned and suggests that more classes might be useful.

We examined the classes learned on English lemmatization by our *ngrams+x+latent* model. For each of the input/output pairs in development data, we found the most probable latent class. For the most part, the 2 classes are separated based on whether or not the correct output ends in e. This use of latent classes helped address many errors like *wronging / wronge* or *owed / ow*). Such missing or surplus final e's account for about 70% of the errors for the basic model *n* but only about 30% of the errors for the full model *n+l+x*.

[40] As with Table 2.7, we have regularity information for 6 of the 15 languages.

2.9.2.3 Analysis: What Does the Model Learn?

We now investigate what features and rules our model learns from data. It will be interesting to see if the knowledge that our model extracts from lemmatization data is similar to how a linguist would describe the lemmatization process.

To show this, we list the highest-ranked features learned in the training process for several languages, see the tables in Appendix B on Page 135.

In each table, the first feature column lists the alignment n-gram (possibly backed-off)[41], the second column lists the same feature in more readable format by removing the specific alignment (see Table B.1 on page 136, for example). For each feature, we also list a few examples from training data; these are the training pairs on which that feature has the highest expected feature count under the model, summed over all alignments of that training pair. Therefore, they are the most representative examples for a given feature.

Many of these learned, high-ranked features look like natural rewrite rules that linguists would jot down as well. In English, for example, the rules *ied*$\rightarrow$ *y*, *ies*$\rightarrow$ *y*, *ing*$\rightarrow$ ϵ, *ed#* $\rightarrow$ ϵ*#*, *es#* $\rightarrow$ *e#* and others are among the highest-ranked features (see Table B.3 on page 138). There are also some high-ranked features that cover irregular cases, such as *ew* $\rightarrow$ *ow*. That specific rule helps generate the correct lemmas for *grew*, *knew*, *threw* and similar verbs. One reason why it is ranked so high is that it is in conflict with the second-best rule overall, which determines that an *e* in the input needs to remain an *e* in the output. Together with other rules, the *ew* $\rightarrow$ *ow* rule can overwrite such default behavior, so that *threw* ends up being correctly rewritten as *throw* and not as *threw*.

In English, note also Features 5 and 19; they are conjugation class features. The examples listed for Feature 19 indicate that conjugation class 2 prefers certain irregular verbs, like *would* and *been*. For a discussion of learned conjugation classes see also Section 2.9.1.2 on page 41 and Section 2.9.2.2 on the preceding page above.

Our model also stores some of the irregular pairs in almost complete form as features. Feature 18, for example, states that *wer* should be rewritten as *b*, which applies only to the training pair *were*$\rightarrow$*be*. If we had allowed features longer than trigrams then the complete training pair could have been stored as a feature. This would be useful and closer to what a linguist would do in such extremely exceptional cases; we list longer-span features as future work (see Section 2.10 on page 51).

2.9.3 Transliteration

To show that our string-to-string model is sufficiently general and is useful beyond morphological tasks, we also report results on a transliteration task. We train and evaluate on the English-Russian data from the NEWS 2009 shared task (Li, Kumaran, Zhang, and Pervouchine, 2009a).

[41] The backoff features tend not to be among the highest-ranked features in these training runs; the reason for this is likely that we added the backoff features in the last stage, for better training efficiency on the lemmatization datasets (see Section 2.7 on page 35).

2.9.3.1 Setup

No changes to the model structure are necessary. We use the pruning and backoff strategies described in Section 2.6.2 on page 31, including the described optional posterior pruning step, where an alignment path is pruned if its probability is less than 0.1 times the probability of the best alignment path, before n-grams N_0 are extracted.

In staged training, we add bi- and trigrams in stage 2. We then run a stage (2b), where we reestimate the pruned Σ and add n-grams up to order 5. Then we run a stage (2c) similar to (2b), but with n-grams up to order 6. These settings are determined by experimentation on the provided development set. For efficiency, we do not add any backoff features or latent variables.

2.9.3.2 Results

We obtain final results on the provided test set, shown in Table 2.9 on the next page. The highest accuracy result among all submissions reached an accuracy of 61.3%, while the median result was 54.8%. With an accuracy of 60.0%, our result is among the few highest-scoring ones and ranks 4th out of 15 submissions. We believe that even higher accuracy results can be achieved when backoff or latent features are added.

We add a list of our result in terms of all scores that the NEWS shared task uses for evaluation, compared to the workshop submissions, for details see Li et al. (2009a):

- ACC: 0.600 (4th rank)
- F-score: 0.921 (5th rank)
- MRR: 0.694 (4th rank)
- MAP_{ref}: 0.600 (4th rank)
- MAP_{10}: 0.295 (1st rank)
- MAP_{sys}: 0.295 (2nd rank)

It is interesting that our system ranks first in terms of MAP_{10}; this means that the 10-best lists produced by our system are of high quality; when there are multiple references they can more often be found in our system's 10-best lists than in other systems' 10-best lists. This property of producing good distributions where different most plausible candidates rank high, as opposed to optimizing for the one-best solution, makes our approach especially suitable to be part of a larger graphical model (Chapter 3) because there, inference is based on combining and negotiating agreement between the distributions from several different string-to-string models (see Section 3.4 on page 64).

Team ID	ACC	Organization
7	0.613	University of Alberta
22	0.609	SRI International (post-evaluation)
6	0.605	NICT
-	**0.600**	**Our results**
17	0.597	*N/A*
24	0.566	SRA
8	0.564	*N/A*
31	0.548	IIIT
23	0.545	IBM Cairo TDC
3	0.531	University of Tokyo
10	0.506	Johns Hopkins University
4	0.504	University of Illinois, Urbana-Champaign
9	0.500	*N/A*
22	0.364	SRI International
27	0.354	*N/A*

Table 2.9: Standard runs for the English to Russian transliteration results, compared to the NEWS 2009 shared task submission results.

2.10 Remaining Challenges and Future Work

This work opens up interesting venues for future research. One might want to identify additional features, latent variables, and training methods that port well across languages and string-transduction tasks.

Accuracy could be improved by using features that look at wide context on the *input* side (Jiampojamarn et al., 2007), which is inexpensive. Latent variables one might consider are an increased number of word classes; more flexible regions (see Petrov, Pauls, and Klein (2007) on learning a state transition diagram for *acoustic* regions in phone recognition); and phonological features and syllable boundaries. Indeed, our local log-linear features over several aligned latent strings closely resemble the soft constraints used by phonologists (Eisner, 1997), as described in Section 2.5.

All features should in future work be evaluated on more tasks than the inflection task, the lemmatization task and the transliteration task that we have presented. For most tasks in general (including the ones we have presented), it would be interesting to work on linguistic word representations other than the orthographic forms. One might want to consider various phonological forms as well, either as an alternative to the orthographic forms or in conjunction with them, as additional layers or tiers, see our multiple-tier discussion below.

In contrast to more language- and task-independent features, it would also be useful to develop features that cover *specific* morphological phenomena that we find in some languages and devise language-specific features. Although many non-concatenative phenomena can be captured by the features we present in this chapter—we can model any gen-

eral change at any position in the words and are not restricted to just handling prefixes or suffixes—there are some specific phenomena that could better be handled by specifically designed features. For the following phenomena, it would be interesting to add specific features and other mechanisms (e.g., more latent variables) that may work better than the more general methods we have presented so far:

- Infixation

  Example: *palit-p**um**alit* (Tagalog)[42]

  We have not presented any features that specifically describe infixes or points of infixation. However, we present good prediction results on languages that use infixation (e.g., Irish and Tagalog). This is possible because infixes can in many cases also be handled by general n-gram features. In the example above (*palit-p**um**alit*), a regular n-gram feature can certainly describe and reward an insertion of *-um-*, by itself or in certain contexts. We analyze a learned Tagalog lemmatization model and find that indeed among the highest-scoring features are features like (h:h)(u:ϵ)(m:ϵ) or (p:p)(u:ϵ)(m:ϵ), which means separate features are learned that reward the deletion of the *um* affix after an *h*, *p*, or other specific consonants.[43] Similarly, there are high-scoring features that reward the deletion of *um* before certain vowels, e.g., (u:ϵ)(m:ϵ)(o:o). Two simple changes to our feature set would enable our model to capture infixation better: (1) an n-gram window of size 4 would have captured this affix as an infix, since in that case context on the left and on the right of the short infix could be part of the feature, as in a feature (h:h)(u:ϵ)(m:ϵ)(o:o).[44] (2) Features that mix different character classes would have provided better generalization. For example, if we had features that combine both letters and vowel/consonant classes, we could capture the deletion of *um* after a consonant in general: (C:C)(u:ϵ)(m:ϵ). Note that these features would be useful in capturing short infixes even though they are general features not designed for infixes specifically. Long infixes would still pose a problem. Here, the latent change regions (Section 2.3) in our model help. A specific change region number can be learned that is dedicated to inserting an infix. In our error analysis for German (page 41), which shows phenomena similar to infixation (*entgegentreten-entgegen**zu**treten*), we observed that an infix region was learned. German and Tagalog both belong to the languages where the gains from the latent variables in general were exceptionally high (see Table 2.5 on page 45 and text on page 44).

[42] As a curiosity, note that Tagalog has borrowed the English word *to graduate* and builds the form *I graduated* as gr**um**aduate (Zuraw, 2007).

[43] Similar features are also shown in Table B.5 on page 140.

[44] We used window size 3 for all large batches of experiments since larger windows would currently pose efficiency problems for many datasets. (Runtime improvements are listed below as another opportunity for future work.)

- Vowel harmony

  Example: *el→elin*, but *kız→kızın* (Turkish)

  We are currently not handling vowel harmony in general. Only if vowel harmony occurs within a short window can it be captured by our standard n-gram-based features, e.g., *ız→ızı* and *el→eli*. However, often vowels are assimilated over a longer distance. Our latent conjugation classes can help in this case; classes can be learned that prefer certain similar vowels throughout the whole strings. One would need to allow more conjugation classes than we did in this chapter, to be able to learn different classes for different vowels. One could also manually design different classes, one for each vowel. Vowel harmony is related to templatic morphology because in both cases, vowels must agree over long distances, so the following considerations about templatic morphology are relevant to modeling vowel harmony as well.

- Templatic morphology

  Example: *ktb→kateb*, *ktb→kattab*, *ktb→kuttib* (Arabic)

  In templatic morphology, verb roots are consonant patterns, which do not change during inflection; what changes are vowels in between the fixed (*shape-invariant*) consonants. This is related to vowel harmony in that vowels in a verb need to agree over a potentially long distance. The feature set we present in this thesis does not cover this phenomenon specifically. Our noncommittal n-gram features can capture vowel changes only if they happen within small local windows. The examples above suggest that much of the phenomenon could be captured by windows of size 4 or wider, which could certainly be done. In that case, a feature like *atta→utti* may receive a high weight during training. In addition, one would want to abstract away from the consonants in such windows, firing a more general feature like *aCCa→uCCi* (see our discussion of infixation above), where *CC* describes any two consonants in the middle.

  McCarthy (1981) proposes to model Arabic templatic morphology using separate tiers: One tier contains just the consonantal root, another tier contains the vowel-consonant pattern, and a third one contains the vowels, similar to approaches described by Wilson and Hayes (2008).[45] These tiers are modeled using a multi-tape finite-state machine. A problem with multi-tape finite-state machines is that they can become very large, since multiple tapes and their alignments are represented. McCarthy works around this problem by using the vowel-consonant patterns to restrict the alignments. But when multiple strings need to be modeled (for example, one input and one output string, with three tapes each), a single multi-tape finite-state machine can become unmanageably big. In this thesis, we offer an alternative: We present graphical models over strings in the next chapter (Chapter 3), which can be described as a decomposition of multi-tape finite-state machines (see Section 3.6.1).

[45] See also our discussion on page 28.

These would be very appropriate to model phenomena like vowel harmony and templatic morphology; this is left for future work.

- Reduplication

  Example: *currit-**cu**currit* (Latin)

  In reduplication, word material gets read and copied. Such phenomena are notoriously hard to handle as a productive process in a finite-state approach (Roark and Sproat, 2007). Specific rules can be learned using alignment n-gram features (e.g., *cu*→*cucu*), but it is harder to learn general rules (e.g., *read a vowel and a consonant and output them twice*). If we added a postprocessing step to our model, we could in a first step insert a simple duplication marker, which then gets interpreted by a postprocessor.

- General vowel change

  Example: *lesen*→*las* (German)

  We model different vowel changes by using different alignment n-gram features, e.g., *e*→*a* or *les*→*las*. It would be desirable (and easy) to add a more general feature that fires whenever some vowel V changes into another, V', where $V \neq V'$. This would be only a small change from a feature that we do have, i.e. one that fires whenever a vowel is aligned to a vowel in general.

- Agglutinative morphology

  Example: *epäjärjestelmällisyys* 'unsystematicalness' (Finnish)

  Currently, we do not handle agglutinative morphology in a productive way; but see the discussion in Section 4.8 on page 118.

In addition to feature design and efforts to capture specific morphological phenomena, there are also engineering aspects that could advance our work and improve inference speed and usability of our method. In particular, our implementation used for the experiments in this chapter builds up a complete finite-state scoring machine $U_{\boldsymbol{\theta}}$ that encodes all features in advance, which can then be composed with observations at training or test time. This generic implementation is very general and allows for composition with whole *distributions* over strings as input or output, making this approach work well as a component in other models and suitable for the re-use in Chapters 3 and 4. However, this is very memory-intensive; such a machine can use several gigabytes of memory. We experimented with an alternative approach, where the complete scoring machine is not built in advance, but several smaller machines, each one encoding a subset of the features (e.g., all consonant/vowel features), are kept separate and are lazily intersected when presented with some input and/or output. These frequent lazy intersection operations, however, turned out to be too slow. One could devise a more efficient implementation where given inputs and outputs are scored using hand-tailored code rather than general finite-state code. The challenge

would be to make this generic enough to be pluggable into a graphical model like the ones in Chapter 3. We also experimented with using the special ρ and σ arcs, which have the semantics of *other*, summarizing set of other arcs at a state that do not explicitly have to be enumerated. However, we did not identify a satisfying solution for their use in transducers or alignment-string acceptors that practically saves memory and runtime.

Finally, it would be useful to further investigate language-specific backoff patterns and adaptively sized n-gram windows, which we successfully used in Section 2.9.3 on page 49. One could use the related method of (Lavergne, Cappé, and Yvon, 2010), where L_1 regularization is employed to focus on the most helpful n-grams, which are then extended to longer n-grams in subsequent training stages. Or one could consider the similar method of *grafting* (Perkins et al., 2003), where at the end of each training stage the gradients of new candidate features are computed, and the feature with the highest absolute gradient value is added to the model, which is then repeatedly retrained in this fashion.

2.11 Summary

The modeling framework we have presented here is, we believe, an attractive solution to most string transduction problems in natural language processing. Rather than learn the topology of an arbitrary WFST, one specifies the topology using a small set of feature templates, and simply trains the weights.

We evaluated on two morphology generation tasks and a transliteration task. When inflecting German verbs, even with the simplest features we outperform the *moses3* baseline on 3 out of 4 tasks, which uses the same amount of context as our models. Introducing more sophisticated features that have access to latent classes and regions improves our results dramatically, even on small training data sizes. Using these we outperform *moses9* and *moses15*, which use long context windows, reducing error rates by up to 48%. On the lemmatization task we were able to improve the results reported in Wicentowski (2002) on 14 out of 15 tested languages and reduce the error rates by up to 92%. The model's errors are often reasonable misgeneralizations (e.g., assume regular conjugation where irregular would have been correct), and it is able to use even a small number of latent variables (including the latent alignment) to capture useful linguistic properties. When applied to a NEWS transliteration task, our system ranks among the highest-scoring ones and proves to have higher-quality 10-best lists than all other systems (MAP_{10} score). This makes it especially useful for tasks where distributions over strings are important, which leads us to an extension of our work described in the next chapter.

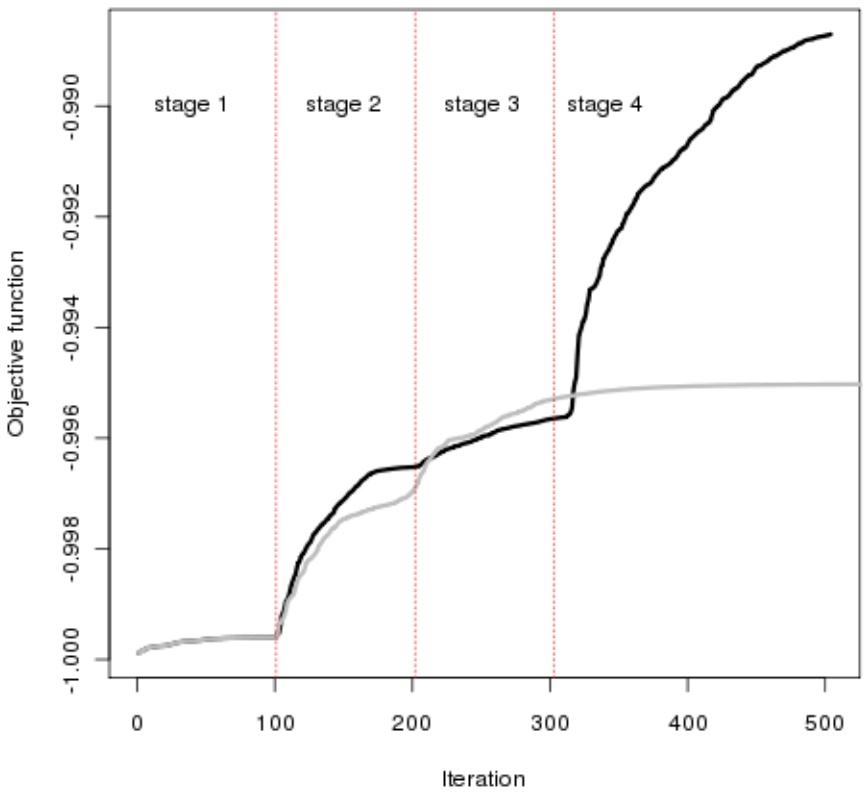

Figure 2.3: The training objective Equation 2.5 during the four stages of training (black). This is one of the 5 training runs for the rP-pA inflection task. Stages 2 (addition of 2- and 3-gram features over alignment characters) and 4 (addition of latent annotations, see Section 2.3) dramatically increase the objective function. The gray line shows what happens if all features are immediately active after stage 1; the objective function converges to a lower optimum.

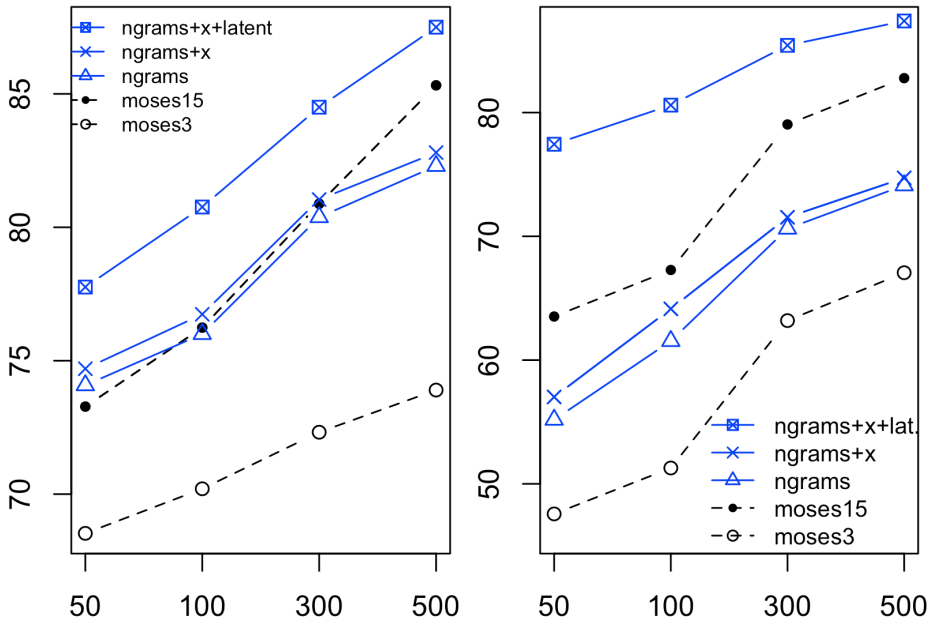

Figure 2.4: Learning curves for German reinflection tasks, 13SIA (left) and 2PKE (right), as a function of the number of training pairs. *ngrams+x* means all backoff features were used; *ngrams+x+latent* means all latent features were used in addition. *Moses15* examines windows of up to 15 characters.

Chapter 3

Graphical Models over Multiple Strings

3.1 Overview

In this chapter, we extend the string-based models we have presented so far. While the approach of the previous chapter handles one or two strings, we will here develop more complex joint models over *multiple* strings. These can be used to learn and predict whole inflectional paradigms.

We will see that this is not a trivial extension of the models presented so far, but requires factorizing the joint probability into several parts that need to negotiate with each other during inference. Such string-based factored models, or *graphical models over strings*, and the associated inference method are part of the novel contributions made in this thesis.

Graphical models have become popular in machine learning as a principled way to work with collections of interrelated random variables. The models over multiple strings that we present in this chapter are graphical models in which the variables range over *strings* of unbounded length, rather than over the typical *finite* domains such as booleans, words, or tags. Variables that are connected in the graphical model are related by some weighted finite-state transduction, as presented in the previous chapter.

Graphical models are most often used as follows:

1. **Build:** Manually specify the n variables of interest; their domains; and the possible direct interactions among them.

2. **Train:** Train this model's parameters θ to obtain a specific joint probability distribution $p(V_1, \ldots, V_n)$ over the n variables.

3. **Infer:** Use this joint distribution to predict the values of various unobserved variables from observed ones.

Note that 1. requires intuitions about the domain; 2. requires some choice of training procedure; and 3. requires a choice of exact or approximate inference algorithm.

Our graphical models over strings are natural objects to investigate. We motivate them with some additional applications in computational linguistics and other fields (Section 3.2). We then give our formalism: a Markov Random Field whose potential functions are rational weighted languages and relations (Section 3.3). Next, we point out that inference is in general undecidable, and explain how to do approximate inference using message-passing algorithms such as belief propagation (Section 3.4). The messages are represented as weighted finite-state machines.

Finally, we report on some initial experiments using these methods (Section 3.7). We use incomplete data to train a joint model of morphological paradigms, then use the trained model to complete the data by predicting unseen forms.

3.2 Motivation

As laid out in the introduction (Chapter 1), we are interested in learning the inflectional morphology of a language, by learning a model of its inflectional paradigms. The inflected forms of a (possibly irregular) verb are placed in the cells of such inflectional paradigms and can influence and reinforce one another (see Page 2). The Markov Random Field approach presented in this chapter models such behavior of multiple morphologically related words.

Although our model is developed with morphology in mind, it is generally applicable to *any* problem of mapping between multiple different forms and representations of strings:

- mapping an English word to its foreign *transliteration* may be easier when one considers the orthographic *and* phonological forms of both words (see Figure 3.1 on the next page);
- in *spelling correction*, one might want to consider how the misspelling is pronounced in order to find the correct spelling; consider the example in Figure 3.2 on page 61;
- similar *cognates* in multiple languages are naturally described together, in orthographic or phonological representations, or both;
- modern and ancestral word forms form a phylogenetic tree in *historical linguistics*;
- in *bioinformatics* and in *system combination*, multiple sequences need to be aligned in order to identify regions of similarity.

We propose a unified model for multiple strings that is suitable for all the problems mentioned above. It is robust and configurable and can make use of task-specific overlapping features. It learns from observed and unobserved, or latent, information, making it useful in supervised, semi-supervised, and unsupervised settings.

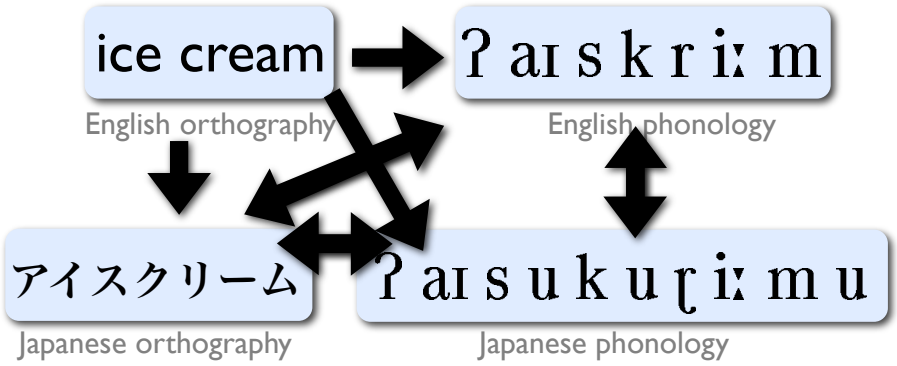

Figure 3.1: Transliteration using graphical models over strings: The orthographic forms in Japanese and English constrain each other; hidden phonological forms give additional useful constraints. This generalization of (Knight and Graehl, 1997) is one of many possible applications of graphical models over strings.

3.3 Formal Modeling Approach

3.3.1 Variables

A **Markov Random Field** (MRF) is a joint model of a set of random variables, $\mathcal{V} = \{V_1, \ldots, V_n\}$. We assume that all variables are string-valued, i.e., the value of V_i may be any string $\in \Sigma_i^*$, where Σ_i is some finite alphabet.

We may use meaningful names for the integers i, such as $V_{2\text{SA}}$ for the ***2nd singular past*** form of a verb.

The assumption that *all* variables are string-valued is not crucial; it merely simplifies our presentation. It is, however, sufficient for many practical purposes, since most other discrete objects can be easily encoded as strings. For example, if V_1 is a part of speech tag, it may be encoded as a length-1 string over the finite alphabet $\Sigma_1 \stackrel{\text{def}}{=} \{\textsf{Noun}, \textsf{Verb}, \ldots\}$.

3.3.2 Factors

A Markov Random Field defines a probability for each assignment $\mathcal{A}$ of values to the variables in $\mathcal{V}$:

$$p(\mathcal{A}) \stackrel{\text{def}}{=} \frac{1}{Z} \prod_{j=1}^{m} F_j(\mathcal{A}) \tag{3.1}$$

This distribution over assignments is specified by the collection of **factors** $F_j : \mathcal{A} \mapsto \mathbb{R}_{\geq 0}$. Each factor (or **potential function**) is a function that depends on only a *subset* of $\mathcal{A}$.

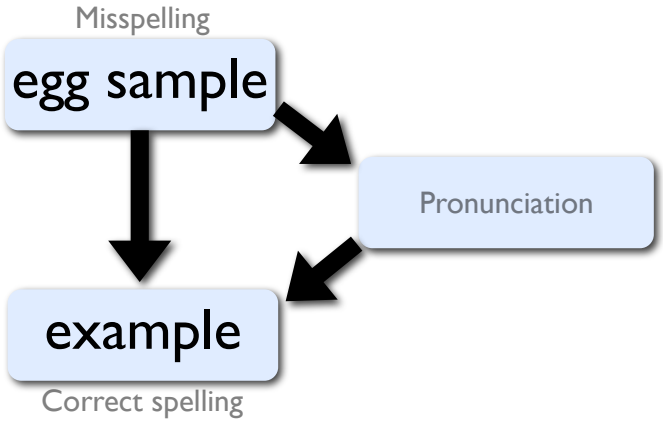

Figure 3.2: Spelling correction using graphical models over strings: In the example, the misspelling and the correct spelling have the same pronunciation. Therefore, modeling the pronunciation (as a hidden variable) can help find the correction; this would make spelling correction a multiple-string problem.

Figure 3.3 displays an undirected **factor graph**, in which each factor is connected to the variables that it depends on. F_1, F_3, F_5 in this example are **unary** factors because each one scores the value of a single variable, while F_2, F_4, F_6 are **binary** factors.

In our setting, we will assume that each unary factor is specified by a **weighted finite-state automaton** (WFSA) whose weights fall in the semiring $(\mathbb{R}_{\geq 0}, +, \times)$. Thus, since F_3 is unary, the score $F_3(\ldots, V_{2SA} = x, \ldots)$ is the total weight of all paths in F_3's WFSA that accept the string $x \in \Sigma^*_{2SA}$. Each path's weight is the product of its component arcs' weights, which are non-negative.

Similarly, we assume that each binary factor is specified by a **weighted finite-state transducer** (WFST). As mentioned before, we use the finite-state transducer models developed in Chapter 2.[1]

Formally, a WFST is an automaton that resembles a weighted FSA, but it nondeterministically reads *two* strings x, y in parallel from left to right. The score of (x, y) is given by the total weight of all accepting paths in the WFST that map x to y. For example, different paths may consider various monotonic alignments of x with y, and we sum over these mutually exclusive possibilities.[2]

[1] We will use a simplified version; conjugation classes and latent regions are not used in this chapter.

[2] Each string is said to be on a different "tape," which has its own "read head," allowing the WFSM to maintain a separate position in each string. Thus, a path in a WFST may consume any number of characters from x before consuming the next character from y.

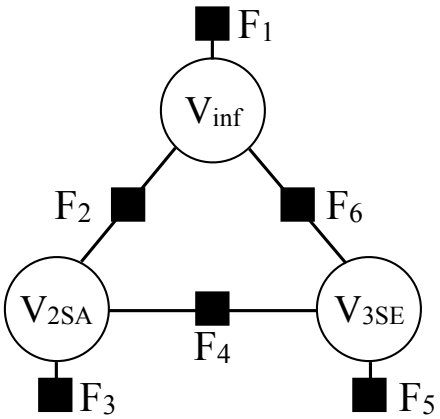

Figure 3.3: Example of a factor graph. Black boxes represent factors, circles represent variables (infinitive, 2nd past, and 3rd present-tense forms of the same verb; different samples from the MRF correspond to different verbs). Binary factors evaluate how well one string can be transduced into another, summing over all transducer paths (i.e., alignments, which are not observed in training).

A factor might depend on $k > 2$ variables. This requires a k-tape **weighted finite-state machine** (WFSM), an obvious generalization where each path reads k strings in some alignment.[3]

To ensure that Z is finite in Equation 3.1, we can require each factor to be a **"proper" WFSM**, i.e., its accepting paths have finite total weight (even if the WFSM is cyclic, with infinitely many paths).

Global versus local normalization. Note that, while Markov Random Fields generally allow global normalization and are therefore undirected graphical models, they allow *local normalization* of the factors as a special case, so they subsume directed graphical models. In such a case, Z in Equation (3.1) on page 60 becomes a product of smaller local factors. Directed graphical models are sometimes preferred for efficiency reasons, since they avoid computation of a global normalization constant (see, for example, 4.6.1.0.2, Page 105).

3.3.3 Parameters

The probability model has trainable parameters: a vector of **feature weights** $\theta \in \mathbb{R}$. In general, each factor F_j in the factor graph evaluates to a non-negative value, according to the values of the variables it is connected to. This non-negative value is the sum of weighted feature values in log-linear space, $\exp \sum \theta \boldsymbol{f}_j(\boldsymbol{\mathcal{A}})$. In our setting, each of these values is computed using a weighted finite-state machine. Each arc in each of the WFSMs has a

[3]Weighted acceptors and transducers are the cases $k = 1$ and $k = 2$, which are said to define **rational languages** and **rational relations**.

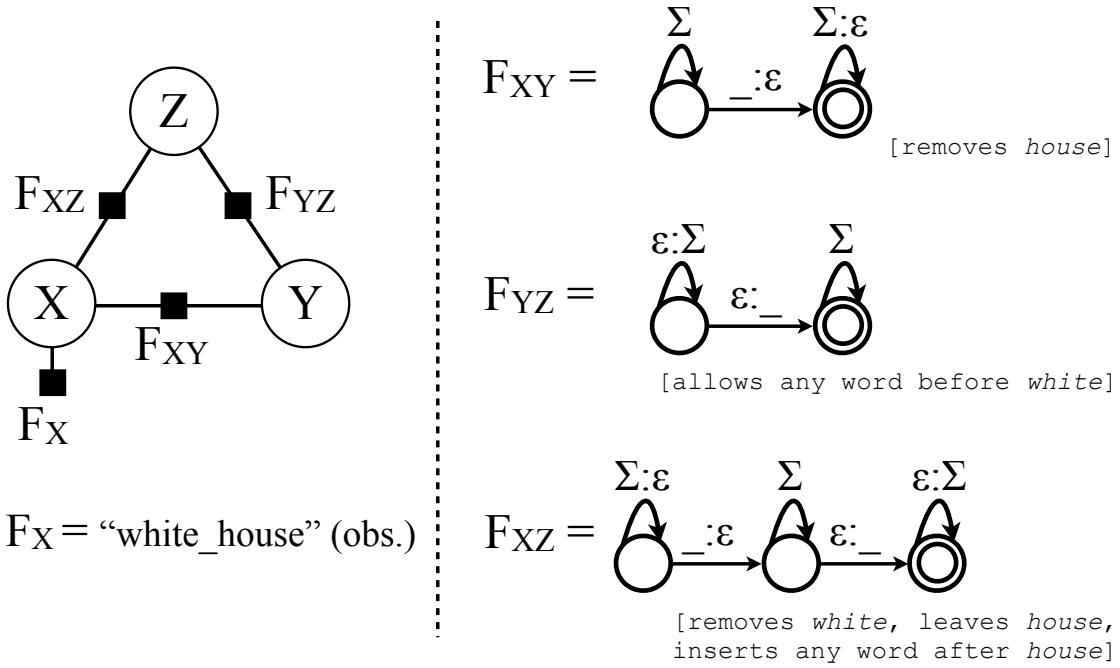

Figure 3.4: Swapping arbitrary word pairs using a graphical model over strings. The variables X, Y, and Z are input, auxiliary variable and output, respectively. As an example, X is assigned the string 11-character string *white_house*. Given the specified factors, the model assigns Y the string value *white* with probability 1, and Z the string value *house_white*. A single finite-state transducer cannot perform such arbitrary swap operations.

real-valued weight that depends on θ. Thus, tuning θ during training will change the arc weights, hence the path weights, the factor functions, and the whole probability distribution $p(\mathcal{A})$.

Designing the probability model includes specifying the topology and weights of each WFSM. In the previous chapter, we explained how to specify and train such **parameterized WFSMs** (see also Eisner (2002b)). Typically, the weight of an arc in log-linear space is a simple sum like $\theta_{12} + \theta_{55} + \theta_{72}$, where θ_{12} is included on all arcs that share feature 12. However, more interesting parameterizations arise if the WFSM is constructed by operations such as transducer composition, or from a weighted regular expression.

3.3.4 Power of the Presented Formalism

Factored finite-state string models (3.1) were originally suggested in Kempe, Champarnaud, and Eisner (2004). That paper showed that even in the unweighted case, such models

could be used to encode relations that could not be recognized by any k-tape FSM. We offer a more linguistic example.

Figure 3.4 specifies a factored model, consisting of three FSTs, that assigns positive probability to just those triples of character strings (x, y, z) that have the form (*red_ball*, *red*, *ball_red*), (*white_house*, *white*, *house_white*, *white*), etc. This uses the auxiliary variable Y to help encode a relation between X and Z that swaps words of unbounded length. By contrast, no finite-state machine can accomplish such unbounded swapping, even with 3 or more tapes.

Such extra power might be linguistically useful. Troublingly, however, Kempe et al. (2004) also observed that the framework is powerful enough to express computationally *undecidable* problems.[4] This implies that to work with arbitrary models, we will need approximate methods.[5] Fortunately, the graphical models community has already developed many such methods, to deal with the computational intractability (if not undecidability) of exact inference.

3.4 Approximate Inference

We will now focus on how **belief propagation** (BP)—a simple well-known method for approximate inference in MRFs (Bishop, 2006)—can be used in our setting. BP in its general form has not yet been widely used in the NLP community.[6] However, it is just a generalization to *arbitrary* factor graphs of the familiar forward-backward algorithm (which operates only on chain-structured factor graphs). The algorithm becomes approximate (and may not even converge) when the factor graphs have cycles. (In that case it is more properly called "loopy belief propagation.")

3.4.1 Belief Propagation

We first sketch how BP works in general. Each variable V in the graphical model maintains a **belief** about its value, in the form of a marginal distribution $\tilde{p}_V$ over the possible values of V. The final beliefs are the output of the algorithm.

Beliefs arise from **messages** that are sent between the variables and factors along the edges of the factor graph. Variable V sends factor F a message $\mu_{V \to F}$, which is an (unnor-

[4] Consider a simple model with two variables and two binary factors: $p(V_1, V_2) \stackrel{\text{def}}{=} \frac{1}{Z} \cdot F_1(V_1, V_2) \cdot F_2(V_1, V_2)$. Suppose F_1 is 1 or 0 according to whether its arguments are equal. Under this model, $p(\epsilon) < 1$ iff there exists a string $x \neq \epsilon$ that can be transduced to itself by the unweighted transducer F_2. This question can be used to encode any instance of Post's Correspondence Problem, so is undecidable.

[5] Notice that the simplest approximation to cure undecidability would be to impose an arbitrary maximum on string length, so that the random variables have a finite domain, just as in most discrete graphical models.

[6] Notable exceptions are Sutton, Rohanimanesh, and McCallum (2004) for chunking and tagging, Sutton and McCallum (2004) for information extraction, Smith and Eisner (2008) for dependency parsing, and Cromierès and Kurohashi (2009) for alignment.

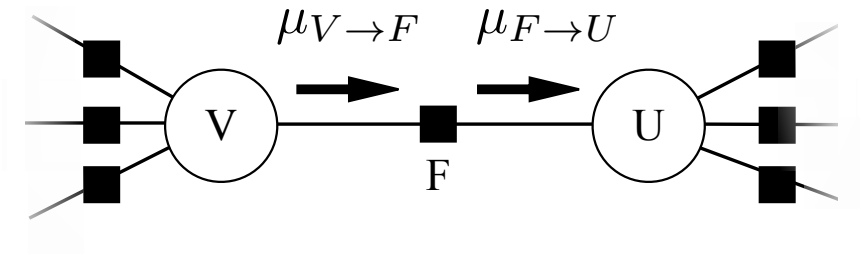

Figure 3.5: Illustration of messages being passed from variable to factor and factor to variable. Each message is represented by a finite-state acceptor.

malized) probability distribution over V's values v, computed by

$$\mu_{V \to F}(v) \ := \prod_{F' \in \mathcal{N}(V), F' \neq F} \mu_{F' \to V}(v) \tag{3.2}$$

where $\mathcal{N}$ is the set of neighbors of V in the graphical model. This message represents a consensus of V's *other* neighboring factors concerning V's value. It is how V tells F what its belief $\tilde{p}_V$ would be if F were absent. Informally, it communicates to F: *Here is what my value would be if it were up to my other neighboring factors F' to determine.*

The factor F can then collect such incoming messages from neighboring variables and send its own message on to another neighbor U. Such a message $\mu_{F \to U}$ suggests good values for U, in the form of an (unnormalized) distribution over U's values u, computed by

$$\mu_{F \to U}(u) \ := \sum_{\mathcal{A} \ s.t. \mathcal{A}[U]=u} F(\mathcal{A}) \prod_{U' \in \mathcal{N}(F), U' \neq U} \mu_{U' \to F}(\mathcal{A}[U']) \tag{3.3}$$

where $\mathcal{A}$ is an assignment to all variables, and $\mathcal{A}[U]$ is the value of variable U in that assignment. This message represents F's prediction of U's value based on its *other* neighboring variables U'. Informally, via this message, F tells U: *Here is what I would like your value to be, based on the messages that my other neighboring variables have sent me about their values, and how I would prefer you to relate to them.*

Thus, each edge of the factor graph maintains two messages $\mu_{V \to F}, \mu_{F \to V}$. All messages are updated repeatedly, in some order, using the two equations above, until some stopping criterion is reached.[7] The *beliefs* are then computed:

$$\tilde{p}_V(v) \stackrel{\text{def}}{=} \prod_{F \in \mathcal{N}(V)} \mu_{F \to V}(v) \tag{3.4}$$

If variable V is *observed*, then the right-hand sides of equations (3.2) and (3.4) are modified to tell V that it *must* have the observed value v. This is done by multiplying in an

[7] Preferably when the beliefs converge to some fixed point (a local minimum of the Bethe free energy). However, convergence is not guaranteed.

extra message $\mu_{\text{obs}\to V}$ that puts probability 1 on v[8] and 0 on other values. That affects *other* messages and beliefs. The final belief at each variable estimates its *posterior* marginal under the MRF (3.1), *given all observations*.

3.4.2 Finite-state Messages in Belief Propagation

Both $\mu_{V\to F}$ and $\mu_{F\to V}$ are unnormalized distributions over the possible values of V—in our case, strings. A distribution over strings is naturally represented by a WFSA. Thus, belief propagation translates to our setting as follows:

- Each message is a WFSA.
- Messages are typically initialized to a one-state WFSA that accepts all strings in Σ^*, each with weight 1.[9]
- Taking a pointwise product of messages to V in Equation 3.2 corresponds to WFSA intersection.
- If F in Equation 3.3 is binary,[10] then there is only one U'. Then the outgoing message $\mu_{F\to U}$, a WFSA, is computed as $\text{domain}(F \circ \mu_{U'\to F})$.

  Here $\circ$ composes the factor WFST with the incoming message WFSA, yielding a WFST that gives a joint distribution over (U, U'). The domain operator projects this WFST onto the U side to obtain a WFSA, which corresponds to marginalizing to obtain a distribution over U.
- In general, F is a k-tape WFSM. Equation 3.3 "composes" $k-1$ of its tapes with $k-1$ incoming messages $\mu_{U'\to F}$, to construct a joint distribution over the k variables in $\mathcal{N}(F)$, then projects onto the kth tape to marginalize over the $k-1$ U' variables and get a distribution over U. All this can be accomplished by the WFSM generalized composition operator $\boxtimes$ (Kempe et al., 2004).

After projecting, it is desirable to determinize the WFSA. Otherwise, the summation in Equation (3.3) on the preceding page is only implicit—the summands remain as distinct paths in the WFSA[11]—and thus the WFSAs would get larger and larger as BP proceeds. Unfortunately, determinizing a WFSA still does not guarantee a small result. In fact it can lead to an exponential blowup, or even infinite blowup.[12] Thus, in practice we recommend

[8] More generally, on *all* possible observed values.

[9] This is an (improper) uniform distribution over Σ^*. Although is *not* a proper WFSA (see Section 3.3.2), there is an upper bound on the weights it assigns to strings. That guarantees that all the messages and beliefs computed by (3.2)–(3.4) will be proper FSMs, provided that all the factors are proper WFSMs.

[10] If it is unary, (3.3) trivially reduces to $\mu_{F\to U} = F$.

[11] The usual implementation of projection does not change the topology of the WFST, but only deletes the U' part of its arc labels. Thus, multiple paths that accept the same value of U remain distinct according to the distinct values of U' that they were paired with before projection.

[12] If there is no deterministic equivalent (Mohri, 1997).

against determinizing the messages, which may be inherently complex. To shrink a message, it is safer to *approximate* it with a small deterministic WFSA, as discussed in the next section.

3.4.3 Approximation of Messages

In our domain, it is possible for the finite-state messages to *grow unboundedly in size* as they flow around a cycle. After all, our messages are not just multinomial distributions over a fixed finite set; they are distributions over the infinite set Σ^*. A WFSA represents this in finite space, but more complex distributions require bigger WFSAs, with more distinct states and arc weights.

Facing the same problem for distributions over the infinite set $\mathbb{R}$, Sudderth, Ihler, Ihler, Freeman, and Willsky (2002) simplified each message $\mu_{V\to F}$, approximating a complex Gaussian mixture by using fewer components.

We could act similarly, variationally approximating a large WFSA P with a smaller one, Q. Choose a family of message approximations (such as bigram models) by specifying the topology for a (small) deterministic WFSA Q. Then choose Q's edge weights to minimize the KL divergence $\mathrm{KL}(P \,\|\, Q)$. This can be done in closed form.[13]

A variant—used in the experiments of this thesis—approximates $\mu_{V\to F}$ by pruning it back to a finite set of most plausible strings.[14] Such a set can be represented as a determinized, minimized, acyclic WFSA. Equation 3.2 requests an intersection of several WFSAs, e.g., $\mu_{F_1\to V} \cap \mu_{F_2\to V} \cap \cdots$. Now approximate that intersection as $Q = ((Q_0 \cap \mu_{F_1\to V}) \cap \mu_{F_2\to V}) \cap \cdots$. The WFSA Q_0 is the topology of the message. It represents the uniform distribution over the set of strings we would like the message to contain. The intersection with the incoming messages reweights it, which is an efficient operation.

How can we construct Q_0? If the factors $\mu_{F\to V}$ are already trained and therefore give reasonable estimates, a straightforward way is to use and combine their predictions to obtain Q_0: List the k-best string predictions from all factors $\mu_{F_i\to V}$, unionize them, minimize and determinize, and remove all weights. This Q_0 can then be used to perform the weighted intersection as described above. If some factors are not trained to give good predictions (yet), they can be left out of that union (see Section 3.5 on the next page). Of course there must be at least one factor $\mu_{F\to V}$, however, that can be used to provide the message topology Q_0.

[13] See Li, Eisner, and Khudanpur (2009b, Footnote 9) for a sketch of the construction, which finds locally normalized edge weights. Or if Q is large but parameterized by some compact parameter vector ϕ, so we are only allowed to control its edge weights via ϕ, then Li and Eisner (2009b, section 6) explain how to minimize $\mathrm{KL}(P \,\|\, Q)$ by gradient descent. In both cases Q must be deterministic.

We remark that if a factor F were specified by a synchronous grammar rather than a WFSM, then its outgoing messages would be weighted context-free languages. Exact intersection of these is undecidable, but they too can be approximated variationally by WFSAs, with the same methods.

[14] In future work, we would like to explore other ways of adaptively choosing the topology of WFSA approximations at runtime, particularly in conjunction with expectation propagation.

3.5 Training the Model Parameters

Any standard training method for MRFs will transfer naturally to our setting. In all cases we draw on Eisner (2002b), who showed how to train the parameters θ of a *single* WFST, F, to (locally) maximize the joint or conditional probability of fully or partially observed training data. This involves computing the gradient of that likelihood function with respect to θ.[15]

We must generalize this to jointly train a *product* of WFSMs. Typically, training data for an MRF (3.1) consists of some fully or partially observed independent and identically distributed (i.i.d.) samples of the joint distribution $p(V_1, \ldots V_n)$. It is well-known how to tune an MRF's parameters θ by stochastic gradient descent to locally maximize the probability of this training set, even though both the probability and its gradient are, in general, intractable to compute in an MRF. The gradient is a sum of quantities, one for each factor F_j. While the summand for F_j cannot be computed exactly in a loopy factor graph, it can be estimated using the BP messages to F_j. In semi-supervised training, where some variable values are unobserved, the gradient for F_j is computed much as in supervised training (see above), but treating any message $\mu_{V_i \to F_j}$ as an uncertain observation of V_i—a form of noisy supervision.[16] We show results from supervised and semi-supervised joint training in the experimental part of this chapter (Section 3.7 on page 70).

3.6 Comparison With Other Approaches

3.6.1 Multi-tape WFSMs

In principle, one could use a 100-tape WFSM to jointly model the 100 distinct forms of a typical Polish verb. In other words, the WFSM would describe the distribution of a random variable $\boldsymbol{V} = \langle V_1, \ldots, V_{100} \rangle$, where each V_i is a string. One would train the parameters of the WFSM on a sample of $\boldsymbol{V}$, each sample being a fully or partially observed paradigm for some Polish verb. The resulting distribution could be used to infer missing forms for these or other verbs.

As a simple example, either a morphological generator or a morphological analyzer might need the probability that *krzyczałoby* is the neuter third-person singular conditional imperfective of *krzyczeć*, despite never having observed it in training. The model determines this probability based on other observed and hypothesized forms of *krzyczeć*, using its knowledge of how neuter third-person singular conditional imperfectives are related to these other forms in other verbs.

Unfortunately, such a 100-tape WFSM would be huge, with an astronomical number of arcs (each representing a possible 100-way edit operation). Our approach is to factor the

[15] The likelihood is usually non-convex; even when the two strings are observed (supervised training), their accepting path through the WFST, i.e., the string alignment, may be ambiguous and unobserved.

[16] See Bishop (2006), or consult Smith and Eisner (2008) for notation close to the notation used here.

problem into a number of (e.g.) pairwise relationships among the verb forms. Using a factored distribution has several benefits over the k-tape WFSM: (1) a smaller representation in memory, (2) a small number of parameters to learn, (3) efficient approximate computation that takes advantage of the factored structure, (4) the ability to reuse WFSAs and WFSTs previously developed for smaller problems, (5) additional modeling power.

3.6.2 Simpler Graphical Models on Strings

Some previous researchers have used factored joint models of several strings. To our knowledge, they have all chosen *acyclic*, *directed* graphical models. The acyclicity meant that exact inference was at least possible for them, if not necessarily efficient. The factors in these past models have been WFSTs (though typically simpler than the ones we will use).

Many papers have used cascades of probabilistic finite-state transducers. Such a cascade may be regarded as a directed graphical model with a linear-chain structure. Pereira and Riley (1997) built a speech recognizer in this way, relating acoustic to phonetic to lexical strings. Similarly, Knight and Graehl (1997) presented a generative cascade using 4 variables and 5 factors: $p(w, e, j, k, o) \stackrel{\text{def}}{=} p(w) \cdot p(e \mid w) \cdot p(j \mid e) \cdot p(k \mid j)$ $\cdot p(o \mid k)$ where e is an English word sequence, w its pronunciation, j a Japanese version of the pronunciation, k a katakana rendering of the Japanese pronunciation, and o a version of the katakana that is corrupted by OCR (optical character recognition). Knight and Graehl used finite-state operations to perform inference at test time, observing o and recovering the most likely w, while marginalizing out e, j, and k.

Bouchard-Côté, Griffiths, and Klein (2009) reconstructed ancient word forms given modern equivalents. They used a directed graphical model, whose tree structure reflected the evolutionary development of the modern languages, and which included latent variables for historical intermediate forms that were never observed in training data. They used Gibbs sampling rather than an exact solution (possible on trees) or a variational approximation (like our BP).

Our work seeks to be general in terms of the graphical model structures used, as well as efficient through the use of BP with approximate messages. We also work with globally normalized models in this chapter.[17]

3.6.3 Unbounded Objects in Graphical Models

We distinguish our work from "dynamic" graphical models such as Dynamic Bayesian Networks and Conditional Random Fields, where the string *brechen* would be represented by creating 7 letter-valued variables. Those methods can represent strings (or paths) of any length—but the length for each training or test string must be specified in advance, not inferred. Furthermore, it is awkward and costly to model unknown alignments, since the

[17] In the next chapter, we will use locally normalized models for efficiency, see Page 105.

variables are position-specific, and any position in *brechen* could in principle align with any position in *brichst*. WFSTs are a much more natural and flexible model of string pairs.

We also distinguish our work from current non-parametric Bayesian models, which sometimes generate unbounded strings, trees, or grammars. If they generate two unbounded objects, they model their relationship by a single synchronous generation process (akin to Section 3.6.1), rather than by a globally normalized product of overlapping factors.

Conditional Neural Fields (CNF, Peng, Bo, and Xu (2009)) are another variant of Markov Random Fields that is worth mentioning. It also adds additional, though not unbounded, structure. The log-linear formulation of factors, scoring the input and a particular output label at time t, $\phi(\boldsymbol{x}, y_t) = \boldsymbol{w}\boldsymbol{f}(\boldsymbol{x}, y)$, that is typically found in Markov Random Fields, is replaced by a formulation that adds a hidden neural-network layer. However, all variable domains are finite, and the added structure is enumerable, whereas in our model the factors relate variables with unbounded structure, using dynamic programming.

3.7 Experiments

In this section, we present experiments in which we model multiple strings with Markov Random Fields. We show a data-driven method to induce factor graphs; we train the factor graphs jointly using belief propagation; and we present empirical results in terms using various evaluation measures. We report results on a supervised and a semi-supervised case; in the latter, most or all of the training examples are incomplete.

3.7.1 Model Structure Induction

Our experiments contrast separate prediction, where each form in the paradigm is predicted separately from the lemma, with joint prediction in a factor graph, where some forms may influence each other as they are being predicted.

What factor graphs do we use?

We run on different factor graph topologies, all of which are obtained in a data-driven way. We use *average edit distance* as the main criterion to decide if two forms should be connected in the factor graph: In each training set, we first consider all possible n^2 form pairs. For each form pair, we measure the average edit distance between the observed instances. We then build a complete and connected undirected graph G with all morphological forms as vertices and their pairwise connections as edges. Using this graph G directly as a factor graph would be very costly, and loopy belief propagation is known to perform poorly when run on graph structures with many short loops.

Instead, we find a minimum-cost spanning tree T, using the pairwise edit distance scores as costs. A spanning tree is a connected, undirected graph containing all n vertices, but only a subset of the edges, such that a tree is formed that spans every vertex. This excludes cycles. To find the minimum-cost spanning tree, we use Kruskal's algorithm (Kruskal Jr, 1956).

The approach of using a minimum-cost spanning tree over the variables as model structure is not new in the graphical models literature. Chow and Liu (1968) constructed a minimum-cost spanning tree using the *mutual information* between pairs of observed variables as cost.

In addition to the unconstrained spanning tree, we also find an approximate Hamiltonian path P, which is a degree-constrained spanning tree, where each vertex has only one or two neighbors. Graphical model topologies based on paths are often called *chain-structured* models.

3.7.2 Training and Inference in the Experiments

After constructing a factor graph using the method just shown, we train the corresponding probability model from incomplete or complete training data, $\mathcal{A}_1, \ldots, \mathcal{A}_n$, where each training example is an assignment to the variables. In the experiments in this chapter, the lemma in each assignment, $\mathcal{A}_i[V_{\text{lemma}}]$, is observed; the lemma characterizes and identifies a paradigm. In training, we wish to find

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathcal{A}_i \mid \mathcal{A}_i[V_{\text{lemma}}]) \tag{3.5}$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log\left(\frac{1}{Z_{\mathcal{A}_i[\text{lemma}]}} \prod_{j=1}^{m} F_{\boldsymbol{\theta},j}(\mathcal{A}_i)\right) \tag{3.6}$$

Since each factor $F_{\boldsymbol{\theta},j}$ in p_θ has log-linear form (see Section 3.3.3 on page 62), the gradients of the objective function with respect to the features is computed as the difference of the expected feature counts given the full training data minus the expected feature counts given the input lemmas only, similar to Equation (2.2) on page 21 and to Conditional Random Fields (Lafferty et al., 2001a).

We can follow the gradient using first-order methods like Stochastic Gradient Descent (SGD) or RProp (resilient backpropagation; Riedmiller and Braun, 1992), or, in some cases, second-order methods like L-BFGS (Limited-Memory Broyden–Fletcher–Goldfarb–Shanno; Liu and Nocedal, 1989b). Note that L-BFGS cannot be used if the factor graph is loopy; in these cases we cannot compute the objective function exactly since loopy belief propagation obtains only approximate marginals. L-BFGS would need the exact function value and corresponding exact gradients, otherwise its line search steps will fail.

Since the training objective is highly non-convex, we chose to run training in stages, as we did in Section 2.7, see Page 34.

We first initialize all weights to zero. Then we train all binary factors that connect the lemma form with an output form, until convergence, ignoring all other factors in the model. We then repeatedly add k more factors (drawn uniformly at random from the specified

factor graph to train),[18] and train until convergence again. Whenever we add more factors, we keep the weights trained in the previous step fixed, in order to reduce the search space in the maximization steps. At the end, when all factors are trained in this staged fashion and the current weights $\boldsymbol{\theta}$ are presumably in a good region of the search space, an additional training stage can be run, where all weights may be changed (as opposed to earlier stages, where weights from previous stages were fixed).

This training procedure is similar to Cascade-Correlation (CC; Fahlman and Lebiere, 1990).

The featurized finite-state machines that act as factors in our factor graphs are similar to the finite-state machines described in Chapter 2. They use trigram-based n-gram and backoff features, as described in Section 2.4 on page 24. Due to the increased runtime of training and decoding in a graphical model as compared to just modeling two strings, we do not include latent classes and regions in our experiments. The fact that the finite-state machines do not necessarily model $\Sigma*$ but instead work with a reduced alignment alphabet (see Section 2.6.2 on page 31), means that the intersection in Equation (3.2) on page 65 may be empty. To prevent this, we mix each factor F with a length-decaying n-gram model that models $\Sigma * \times \Sigma *$. We choose a simple zerogram model and give it a very small mixture weight, 10^{-12}. So we have as a factor the union $F' = F \cup 10^{-12}(0.999\Sigma \times \Sigma)$, where the factor 0.999 prevents the zerogram model from generating infinite-length strings.

3.7.3 Supervised Experiment

3.7.3.1 Data

In the following experiment, we use a supervised data set: The training data consist of a number of lemmas with their correct, complete paradigms. A complete paradigm consists of the lemma and 21 inflected forms. At test time, we are given previously unseen lemmas and predict their complete paradigms. The predicted paradigms are then evaluated using whole-word accuracy and edit distance to the correct answers for the various forms to be predicted.

To create the training data, we first sample 200 lemmas and their complete paradigms from the German CELEX morphological database. The remaining 5,415 lemmas and their paradigms in the German CELEX morphological database are used as test data.

From the 200 training instances, we create additional, smaller training sets of sizes 100 and 50 by subsampling. All these training sets are relatively small in the number of paradigms; this is a realistic condition for most of the world's languages (see, for example, Janecki, 2000). Note that each paradigm in itself contains 22 forms, so 100 observed paradigms contain 2,200 observations. From each training set, we use 10% as development data. We repeat this process 10 times, which results in 10 different training/development/test splits of the data. All reported results are averaged over these 10 splits.

[18] $k = 2$ in our experiments.

The same setup will be used in the next chapter (Chapter 4) as well, along with an additional text corpus.

The German CELEX morphological database was filtered to remove verbs with detachable prefix, such as `abheben`/`hebt ab`, which create many redundancies in the data. This is not to make the problem in this chapter easier—after all, predicting `abhebt`, which is easy to generate, instead of the reordered `hebt ab` is in general correct as well, and which form is used would depend on the context (a main clause would use *er hebt ab* versus a subclause . . . *dass er abhebt*). Rather, we are making the data here similar to the data in the next chapter, in which we will identify verbs in a text corpus but have no way of identifying free-floating prefixes as verb parts (see the discussion in Section 4.8.4 on page 120).

3.7.3.2 Results

We evaluate the trained models on the 5,415 test paradigms; the task is to predict all inflectional forms, given the lemma. Tables 3.1 and 3.2 show the results.

A baseline that simply inflects each morphological form according to the basic regular German inflection pattern, reaches an accuracy of 84.5% and an average edit distance to the correct forms of 0.27. This baseline applies simple rules like *in observed lemma form, replace word-final* `-en` *by* `-t` *and add* `ge-` *at the front*, which is correct for regular words like `machen`, `gemacht`, but not for others, e.g., `brechen`, `gebrochen`.

Table 3.1 on the next page shows that the used Hamilton-path factor graph increases whole-word accuracy from 89.4% to 90.4%, and the spanning-tree topology increases accuracy to 90.9% (error reduction of 8.1%), if 50 training paradigms are used (i.e. 1,100 observed morphological forms). If twice the number of training paradigms is used, the baseline accuracy increases by about 1 point, as does the spanning-tree result. The Hamilton path result increases by 0.5%.

	50			100		
Form	**Separate**	**Path**	**Tree**	**Separate**	**Path**	**Tree**
13PIA	74.81	**82.55**	81.68	81.39	85.07	**85.58**
13PIE	99.98	99.98	99.98	99.98	99.98	99.98
13PKA	74.67	**82.55**	81.99	81.15	85.39	**85.76**
13PKE	99.92	99.92	99.92	99.93	99.93	99.93
13SIA	**84.24**	82.81	83.96	**85.82**	84.97	85.45
13SKA	83.48	83.95	**84.47**	86.19	85.85	**86.40**
13SKE	99.75	99.76	**99.78**	99.68	99.69	**99.70**
1SIE	99.64	99.64	99.64	99.49	**99.52**	99.28
2PIA	83.98	83.03	**84.22**	**85.87**	85.24	85.63
2PIE	**98.12**	97.84	97.67	98.14	**98.45**	98.38
2PKA	83.03	83.73	**84.51**	85.15	85.62	**86.19**
2PKE	99.88	99.88	99.88	99.90	99.90	99.90

(continued on the next page)

Table 3.1: (continued)

2SIA	83.75	83.04	**84.34**	**85.78**	85.14	85.68
2SIE	91.12	**91.33**	91.24	94.20	**94.21**	94.09
2SKA	82.23	83.36	**84.22**	84.98	85.91	**86.10**
2SKE	99.90	99.90	99.90	99.91	99.91	99.91
3SIE	**93.88**	93.79	93.77	**94.41**	94.19	94.13
pA	59.76	58.33	**61.45**	63.39	63.66	**64.18**
pE	**99.38**	99.32	**99.38**	**99.39**	99.38	**99.39**
rP	**97.81**	97.53	97.36	98.14	**98.45**	98.36
rS	98.67	98.68	**98.72**	98.71	**98.72**	98.63
all	89.89	90.51	**90.85**	91.49	91.85	**92.02**

Table 3.1: Whole-word accuracy of separate versus jointly trained models on test data. *Path* and *Tree* are the shapes of the factor graphs used: *Path* is an approximate Hamiltonian path based on average edit-distance scores between pairs in training data, whereas *Tree* is a minimum-spanning tree based on these scores. `13PIA`, `13PIE`, ..., are the forms to be predicted, see Table C.1 on page 141.

	50			100		
Form	**Separate**	**Path**	**Tree**	**Separate**	**Path**	**Tree**
13PIA	0.42	**0.32**	0.34	0.34	0.30	**0.29**
13PIE	0.00	0.00	0.00	0.00	0.00	0.00
13PKA	0.43	**0.33**	0.34	0.35	**0.30**	**0.30**
13PKE	0.00	0.00	0.00	0.00	0.00	0.00
13SIA	0.43	0.44	**0.42**	**0.40**	0.41	**0.40**
13SKA	0.34	0.32	**0.31**	0.30	**0.29**	**0.29**
13SKE	0.00	0.00	0.00	0.00	0.00	0.00
1SIE	0.01	0.01	0.01	0.01	0.01	0.01
2PIA	0.39	0.41	**0.38**	**0.36**	0.37	**0.36**
2PIE	0.02	0.02	0.02	0.02	0.02	0.02
2PKA	0.33	0.32	**0.31**	0.31	0.30	**0.29**
2PKE	0.00	0.00	0.00	0.00	0.00	0.00
2SIA	0.39	0.40	**0.38**	**0.36**	0.37	**0.36**
2SIE	0.10	0.10	0.10	0.07	0.07	0.07
2SKA	0.34	**0.32**	**0.32**	0.31	**0.29**	**0.29**

(continued on the next page)

Table 3.2: (continued)

2SKE	0.00	0.00	0.00	0.00	0.00	0.00
3SIE	0.07	0.07	0.07	0.07	0.07	0.07
pA	0.91	0.95	**0.88**	0.84	0.83	**0.82**
pE	0.01	0.01	0.01	0.01	0.01	0.01
rP	**0.02**	**0.02**	0.03	0.02	0.02	0.02
rS	0.02	0.02	0.02	0.02	0.02	0.02
all	0.20	**0.19**	**0.19**	0.18	0.18	**0.17**

Table 3.2: Average edit distance of separate versus jointly trained models on test data. *Path* and *Tree* are the shapes of the factor graphs used: *Path* is an approximate Hamiltonian path based on average edit-distance scores between pairs in training data, whereas *Tree* is a minimum-spanning tree based on these scores.

Loopy factor graphs. In addition to evaluating factor graphs that are chain- and tree-structured, we test what happens if we add loops to the factor graph. Note that in that case, the inference procedure that we run during training—belief proagation—becomes approximate (see Page 64). We add loops in a sequential fashion, using a greedy procedure: Starting from the tree-structured factor graph, we first add the one additional factor that reduces the likelihood of the training data most. This requires testing the likelihood reduction of all $O(n^2)$ potential factors to add, which is an expensive procedure. We make this more manageable by just testing the likelihood reduction after one gradient step instead of full training with each factor. After that, we add a second factor in the same fashion, and so forth. Since this procedure is expensive, we stop after adding 8 factors to the spanning tree. We also evaluate on 5 of the 10 datasets only, so the *sep* (separate prediction of non-lemma forms) and *span* (spanning tree) baselines differ slighlty from Table 3.1 on the preceding page, where the results were averaged over 10 splits.

Note that the greedy approach of adding factors one by one and the approximate likelihood computation make this approach less optimal.

In addition, there is a trade-off: Adding these additional factors adds information to the model, but at the same time it makes training and decoding less exact, due to the loopiness of the graph.[19] We run loopy belief propagation for several iterations. As convergence criterion, we use the Jensen-Shannon (JS) divergence (Lin, 1991);[20] we stop if the L_2

[19]For a possible remedy, see ()Stoyanov, Ropson, and Eisner, 2011 (to appear).

[20]Unlike the Kullback–Leibler (KL) divergence (Kullback and Leibler, 1951), the Jensen-Shannon divergence is symmetric.

	50 paradigms	100 paradigms
sep	*89.70*	*91.25*
span	90.42	91.56
span + 1	89.77	91.70
span + 2	90.05	91.28
span + 3	90.21	91.32
span + 4	90.40	91.34
span + 5	90.48	91.31
span + 6	90.60	91.33
span + 7	90.65	91.39
span + 8	90.68	N/A

Table 3.3: Whole-word accuracies for models where 1,2,... loops were added to the spanning tree factor graph. This is averaged over only 5 of the 10 data splits.

norm of the JS divergences between messages in the previous iteration and messages in the current iteration falls below 10^{-6}. The messages converge quickly; it typically runs for 3 or 4 iterations.

The results are shown in Table 3.3. The findings differ somewhat between the runs with 50 paradigms and with 100 paradigms. With 50 paradigms, we see an initial drop in accuracy when the first loopy factor is added. After that, the accuracy increases steadily as more factors are added, making the graph more and more loopy and adding more and more informative features. With 100 paradigms, we see an inital accuracy gain when the first loopy factor is added; then the performance drops, but increases again in a somewhat unsteady line. However, it does not reach the performance of the first loopy factor again.

It is possible that our greedy procedure hurts accuracy, and a factor that is added early may hurt performance later. It would be useful to have a mechanism to re-evaluate previously added factors after more factors are added. In summary, although the best overall performances (with 50 or with 100 paradigms) can be observed on a loopy graph, the results are somewhat unstable. In future work, we would like to test further construction procedures for loopy factor graphs (see, e.g., (Lee, Ganapathi, and Koller, 2006)).

Token-based Evaluation. In addition to the type-based evaluations described above we now conduct a token-based evaluation of the separate, the chain-structured and the spanning-tree factor graphs. These results will be directly comparable to Table 3.1 on page 74. Here the goal is to find out how well our different models predict inflected forms with varying degrees of frequency in a text corpus. This is especially interesting because the more frequent an inflection is in text the more likely it is to be irregular (Jurafsky, Martin, Kehler, Vander Linden, and Ward, 2000, p. 49). As an example, among the most

Bin	Frequency	# Verb Forms
1	0-9	116,776
2	10-99	4,623
3	100-999	1,048
4	1,000-9,999	95
5	10,000-	10
all	*any*	122,552

Table 3.4: The inflected verb forms from 5,615 inflectional paradigms, split into 5 token frequency bins. The frequencies are based on a 10-million word corpus.

frequent verb forms in English are *is*, *was*, *are*, *has*, *be*, *said*, all of which are irregular.[21] Less frequent forms tend to be more regular. Therefore, this evaluation is similar to our analysis of regular versus irregular forms in Chapter 2 (see page 44).

We conduct this token-based evaluation on the same dataset and the same predictions we evaluated above in this section; but here we split these predictions into different frequency bins and report separate results, thereby answering the question how well our models perform on verb forms that have frequency counts of varying degrees in a large corpus, ranging from *very low*, *low*, *medium*, *high*, to *very high* frequencies. In numbers, these five frequency bins contain the verb forms with the following frequency counts in a 10-million words corpus: *0-9*; *10-99*; *100-999*; *1000-9999*; *10,000 or higher*. Note that most verb forms have *very low* frequency, while there is a very small number of forms with *very high* frequency (see Table 3.4).

As corpus for these frequency counts we used the first 10 million words from the WaCky corpus (Baroni, Bernardini, Ferraresi, and Zanchetta, 2009); the same 10 million words will later be used in Chapter 4, as an unannotated resource for learning (see Page 105). This corpus does not contain any morphological annotations; therefore we cannot directly count how often each form (e.g., the third person singular indicative of *gehen*) occurs. However, given our supervised morphological paradigms, there are only one or a few possible morphological forms per spelling; it is easy to roughly disambiguate these using a simple model (see Appendix D).[22]

Table 3.5 shows the results; the whole-word accuracy results of our models are shown separately for each frequency bin. We observe that the spanning tree model always performs best on Frequency Bins 1 and 2, which contain less frequent and more regular verb inflectins. There is a great number of different inflections in these bins (see Table 3.4),

[21] We indeed found these to be the most frequent word forms in a quick check on a subset of the English Gigaword corpus (Graff and Cieri, 2003).

[22] As described above, the morphological paradigms that we predict are taken from the CELEX morphological database. These forms in those paradigms do have some frequency count attached, but they are not useful here since they are just spelling frequencies. If various morphological forms have the same spelling they all get the same count.

	50			100		
Bin	**Separate**	**Path**	**Tree**	**Separate**	**Path**	**Tree**
1	90.50	91.17	**91.48**	92.09	92.46	**92.63**
2	78.07	77.70	**78.70**	80.17	80.26	**80.38**
3	71.64	70.95	**71.83**	**73.29**	73.16	73.28
4	**57.35**	56.86	57.11	**57.35**	56.62	56.25
5	**20.73**	**20.73**	**20.73**	**20.73**	**20.73**	**20.73**
all	89.89	90.51	**90.85**	91.49	91.85	**92.02**

Table 3.5: Whole-word acuracy on inflected verb forms in different token frequency classes.

which results in overall best scores for that model. Apparently, the more complex factor graph helps learn the regularities of the language better. On the more frequent inflections, separate prediction tends to be better, with a tie of all models on the very frequent forms, which include forms of *sein* 'to be' and similar. As we have pointed out, the more frequently an inflection occurs in text the more irregular it tends to be. Therefore, it is expected that the prediction numbers decrease, as can be seen in the table. Note that the number of training paradigms we use is very small (50 and 100), and if certain irregular verbs are not included there is no way for the model to obtain knowledge about how it is correctly inflected. In Chapter 4, we will conduct the same token-based analysis again (see Page 110), but on models that can learn from corpora, in addition to the seed paradigms. Large corpora, even without any explicit annotation, can provide information on irregularities and exceptions and give clues that lead to better performance on the more frequent forms.

3.7.4 Semi-supervised Experiment

3.7.4.1 Data and Setup

We will now demonstrate that our string-based MRF model can be used to learn from incomplete data as well. We ask the following question: If we just observe the spellings of common (i.e., frequently occurring) morphological forms, can our model learn and predict how all other morphological forms in the language are spelled?

Here we use a morphologically annotated corpus to obtain the frequencies of the various inflected verb forms. It is only of moderate size, which made it unsuitable for the token-based evaluation above (see Page 76),[23] but here it can be useful. In this experiment, the

[23]Only 5,915 of the 63,778 different spellings contained in the CELEX paradigms we used above can be found in the Tiger corpus. The 10 million words from the WaCky corpus, on the other hand, contain 13,216

more frequently occurring morphological forms are the ones that are observed in this text corpus of moderate size, the other forms will be predicted by our model.

We gather information from the German Tiger corpus for training. In that corpus, each verb token is tagged with its correct morphological form in the sentence. The corpus contains 50,474 sentences with 888,579 tokens. We use the given morphological forms of the verb tokens to place each verb token in an inflectional paradigm. This results in 4,284 incomplete paradigms, in which only 2.4 of 20 forms are observed on average. The number of possible forms per paradigm is given by the union of forms found for any lexeme in the corpus.

We use these 4,284 given incomplete paradigms as training data (1) to find a suitable factor graph and (2) to estimate the model parameters of that factor graph. After training, we run inference in these trained factor graphs in order to predict the most likely values for the unoccupied cells in the paradigms. That is, we predict how the more uncommon morphological forms of the language are spelled. In the evaluation, we measure how often those forms were predicted correctly (whole-word accuracy) and how similar the predicted forms were to the correct forms in terms of edit distance.

How does evaluation work, given the fact that the Tiger corpus does not actually give the correct answers for the unoccupied cells, which are the ones we predict? We use information from the CELEX morphological database. We find 3,178 of the 4,284 paradigms in the CELEX database; here they are complete, so we can use them to evaulate the model predictions for the unoccupied cells. Naturally, all other paradigms, for which we do not have correct answers, are discarded.

We use the same model structure induction method (described in Section 3.7.1 on page 70) as in the previous, supervised experiment. The method requires us to determine the average edit distance between each form pair. In the supervised case, every training paradigm contains observations for all form pairs, whereas here in the semi-supervised case, most or all paradigms are missing some of the form pairs. However, for each form pair we can still compile a list of all examples that we do have. For some form pairs, that list is very short (see Table 3.6 on the following page), which means that we cannot determine the their average edit distance reliably, which may lead to suboptimal factor graphs. In future work, we are interested in finding ways to induce good factor graphs even under these adverse conditions. One could for example train a model on the potentially suboptimal factor graph first, then use the model to predict values for the unoccupied cells, re-estimate edit distance using the given values and the predicted values, create a better factor graph, and so forth.

We train the factor graph using the same staged training method as in the supervised case, described in Section 3.7.2 on page 71. The only difference from the supervised case is that here, we take expectations wherever a form is unobserved.

of the spellings we were interested in. A concatenation of both contains hardly any more of the paradigm spellings (13,572).

3.7.4.2 Results

The results can be found in Table 3.6, which contains the whole-word accuracy results for the various morphological forms that were to be predicted and that we evaluate on. The table is sorted by the number of overall observations that the different morphological forms have in the Tiger corpus and the derived incomplete paradigms. The column labeled *separate* contains results from simply predicting each form separately from the lemma form; there is no joint training or joint decoding in that case. The column labeled *tree* contains results from joint training and joint prediction using a tree-shaped factor graph, induced by the edit-distance method (Section 3.7.1).

Form	**# Obs.**	**Separate**	**Tree**
pA	2917	51.85	**53.39**
3SIE	1900	89.93	**90.95**
13PIE	1424	**99.47**	**99.47**
13SIA	1293	82.02	**84.08**
13PIA	874	77.04	**78.23**
z	613	**82.39**	82.12
13SKE	533	99.53	**99.64**
13PKA	204	58.32	**65.23**
1SIE	149	99.18	**99.44**
13SKA	101	53.15	**60.89**
2SIE	33	72.68	**72.72**
rP	31	**54.32**	58.03
2PIE	28	95.57	**96.20**
rS	25	41.29	**41.68**
13PKE	20	65.64	**74.57**
2SIA	3	1.42	**1.70**
pE	2	**99.59**	**99.59**
2SKE	2	0.00	0.00
2PIA	2	0.00	0.00
2SKA	1	0.22	**0.38**
all	10155	58.13	**60.01**

Table 3.6: Whole-word accuracy results on the various morphological forms to be predicted in the semi-supervised experiment.

We observe that joint prediction in the tree-shaped factor graph outperforms separate prediction on almost all forms, often by a large margin. A few forms have been observed three times or less; these are second-person forms and the present participle (pE). On these, the performance of separate prediction and joint prediction are almost equally bad because the few observed pairs contained irregular forms of the verb *sein* 'to be'. The present participle form is an exception; here the few observed forms are regular, and prediction is easy.

3.8 Summary

In this chapter, we have proposed that one can jointly model multiple related strings by using Markov Random Fields. We described this formally as an undirected graphical model with string-valued variables whose factors (potential functions) are defined by weighted finite-state transducers. Each factor evaluates some subset of the strings. The topology of this graphical model may be chain-structured, tree-structured or loopy.

Approximate inference can be done by loopy belief propagation. The messages take the form of weighted finite-state acceptors, and are constructed by standard operations. We explained why the messages might become large, and gave methods for approximating them with smaller messages. We also discussed training methods.

We presented experiments on the task of jointly predicting multiple missing verb forms in morphological paradigms. We demonstrated that we can learn from complete or incomplete inflectional paradigms, and that the factor graphs outperform separate-prediction models.

The factors were simplified versions of statistical finite-state models for supervised morphology. Our MRF for this task might be used not only to conjugate verbs (e.g., in MT), but to guide further learning of morphology—either active learning from a human or semi-supervised learning from the distributional properties of a raw text corpus.

Our modeling approach is potentially applicable to a wide range of other tasks, including transliteration, phonology, cognate modeling, multiple-sequence alignment and system combination.

The work in this chapter ties into a broader vision of using algorithms like belief propagation to coordinate the work of several NLP models and algorithms. Each individual factor considers some portion of a joint problem, using classical statistical NLP methods (weighted grammars, transducers, dynamic programming). The factors coordinate their work by passing marginal probabilities. Smith and Eisner (2008) reported complementary work in this vein.

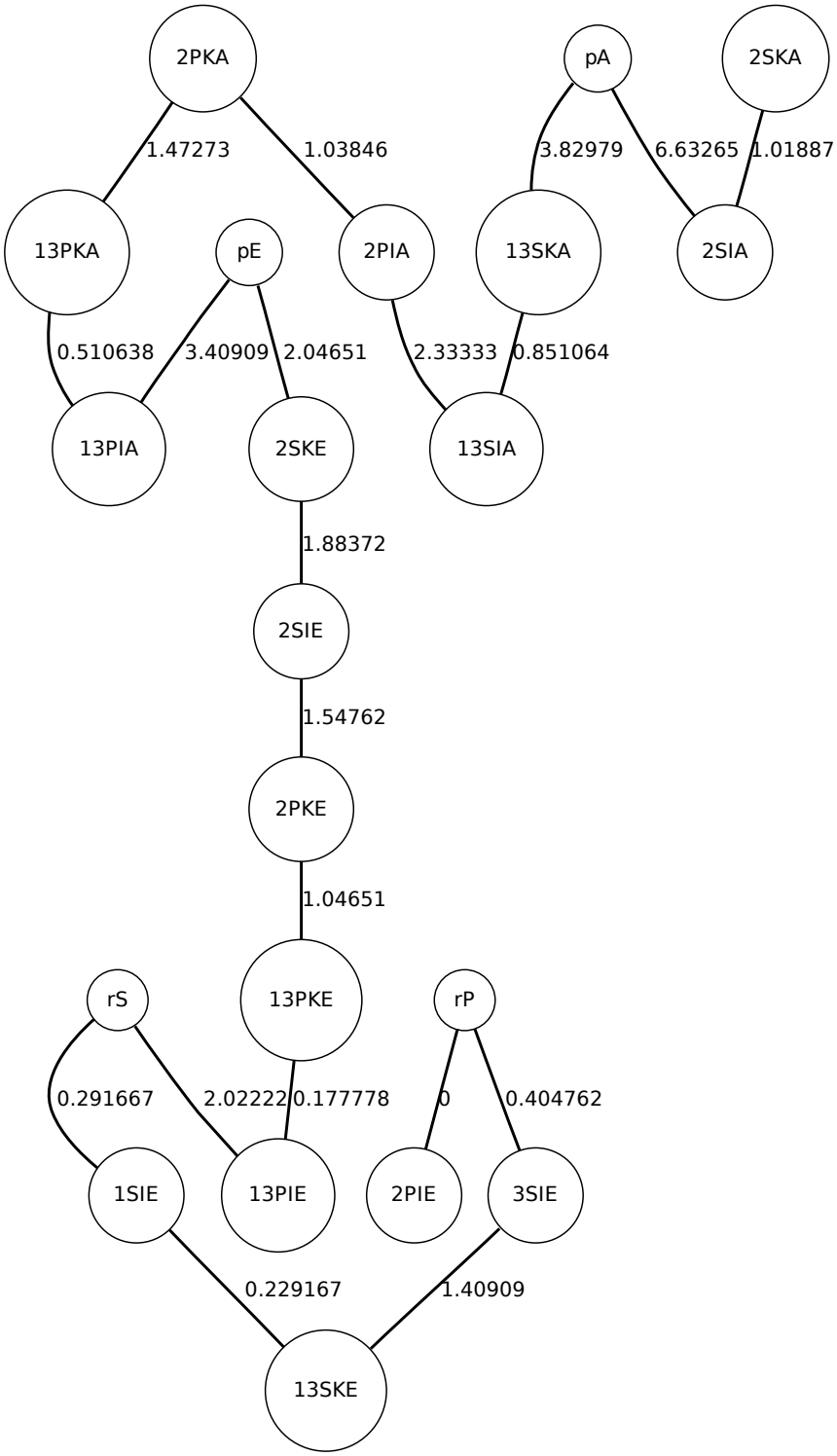

Figure 3.6: An approximate Hamiltonian path through all German CELEX forms in the supervised experiment; the distances between form pairs are measured in edit distance.

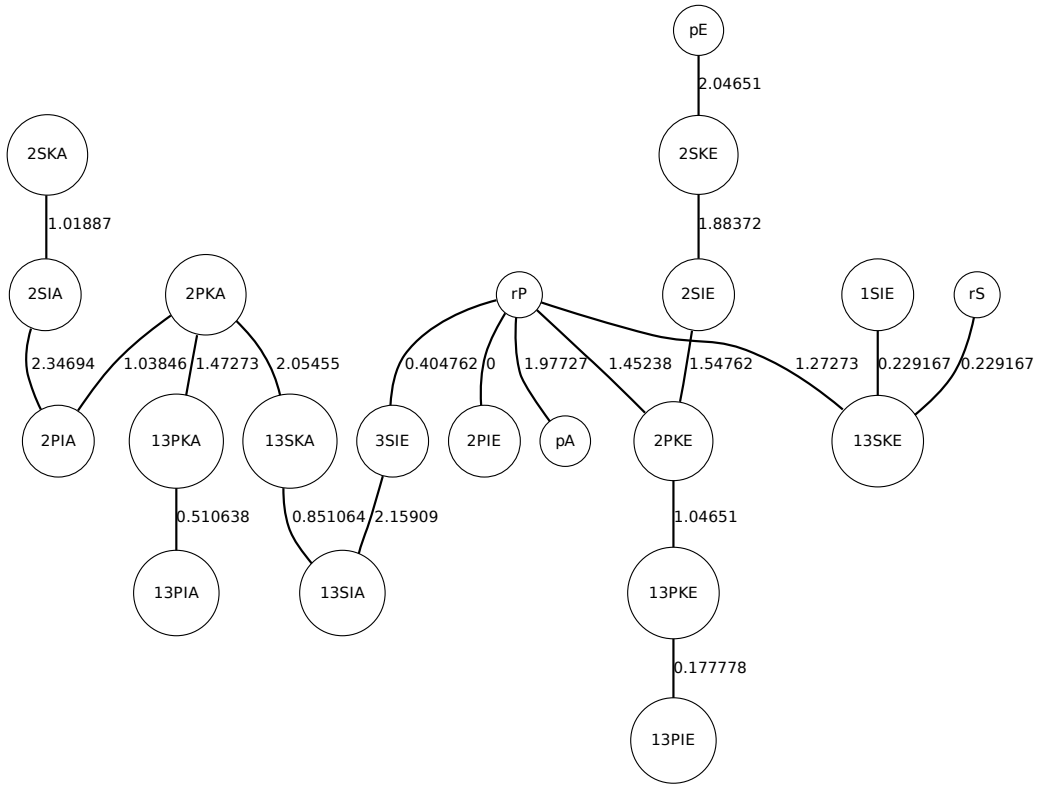

Figure 3.7: Minimum spanning tree over all German CELEX forms in the supervised experiment; the distances between form pairs are measured in edit distance.

Chapter 4

Discovering Morphological Paradigms from Plain Text using Graphical Models

4.1 Introduction

4.1.1 Outline

This chapter further extends our approach to learning the morphology of a language. We will make use of the models developed in the previous chapter but add something very natural: The ability to learn the morphology of a language—with its rules, exceptions and oddities—from reading and analyzing large amounts of text that contains no prior morphological annotation. For this task, we require no morphological supervision except a small set of example paradigms to get started.

We will present a well-defined generative probability model along with a statistical inference procedure for this task and give a detailed, formal definition in the next sections of this chapter. Here we provide an overview and more intuitive explanation.

The probability model is based on the following ordinary linguistic notions:

1. There exist infinitely many lexemes in a language.
2. A lexeme's paradigm contains systematically related spellings.
3. To generate a word, pick a lexeme and inflection, then look up the spelling in the paradigm.

The inference procedure simultaneously *analyzes* text and *constructs* inflectional paradigms accordingly. The analysis consists of assigning a part-of-speech (POS) tag (unless that is given by an external POS tagger), a lexeme and an inflection. In the next subsection, we give an example that illustrates the process and gives an intuition of how our approach utilizes unannotated text data and and learns morphological information from it.

The example is similar to the one given in Section 1.3.3.1 on page 11; here we emphasize more what kinds of errors can potentially be fixed in our corpus-based inference procedure.

4.1.2 Example

Suppose you want to learn German verb morphology, and you know that the 3rd person past indicative is usually built by appending **-te** to the stem, as in `sag`**`te`**, `hör`**`te`** or `lieb`**`te`**. But upon reading actual German text you also find verb forms that are very similar but slightly different: `red`**`ete`**, `atm`**`ete`**, `zeichn`**`ete`**. These end in **-ete**. You may (correctly) hypothesize that this is a variation on the same 3rd person past-tense form that would usually be built with just **-te**. You may think so especially since the otherwise expected forms `red`**`te`**, `atm`**`te`** and `zeichn`**`te`** never occur and the ones you observed instead are so similar. And what else would `red`**`ete`** be? According to your knowledge it certainly looks closest to a 3rd person past indicative. You (preliminarily) assume that that this past-tense form can be built with **-te** or **-ete**, and with this assumption in mind you can search and analyze other forms in the text. You may even try to generalize and hypothesize that it is **-ete** only after the consonants **d**, **m** or **n**, like in the observed `re`**`dete`**, `at`**`mete`** and `zeich`**`nete`**. But as you read more text you will find exceptions like `wan`**`dte`**, `käm`**`mte`** or `krön`**`te`** (which should have been `wan`**`dete`**, `käm`**`mete`** and `krön`**`ete`** according to that new rule). These observed exceptions will force you to relativize and further refine your constantly evolving morphological knowledge of that language. You will often want to go back to previously analyzed words and re-analyze them, according to newly acquired knowledge.

These are the kinds of hypotheses, generalizations and decisions that we will make during inference under our model.

Although the example above presented them almost as a thought process that sounds like an agglomerate of heuristics and arbitrary rules of thumb, we will later see that all decisions are made under one **clean, well-defined joint model of morphological knowledge** that captures all the phenomena described above, making use of general and linguistically motivated features in a well-understood statistical inference procedure.

Whenever a word in the corpus is preliminarily analyzed (e.g., `red`**`ete`** as 3rd person past indicative of `reden`) it is placed in an inflectional paradigm; this could be a previously created paradigm that already contains slots filled with forms like `reden` and `redetest`, or a new, otherwise empty one. And in any paradigm that still has unfilled slots because we have not encountered the corresponding forms in the corpus, we maintain a probability distribution over possible morphological forms that we would expect in that slot, given the spellings in the occupied cells so far. Such a process of filling an inflectional paradigm and maintaining uncertainty over so-far unoccupied slots is depicted in Figure 1.1 on page 5.

4.1.3 Summary

To summarize our approach in more formal detail, our goal is to jointly reconstruct both token and type information about a language:

First, we will tag each word token in a corpus with (1) a *part-of-speech tag* (unless given), (2) an *inflection*, and (3) a *lexeme*. E.g., a token of `broken` might be tagged in context as a VERB and more specifically as the past participle inflection of the abstract lexeme *break*.

Reconstructing the latent lexemes and inflections allows the features of other statistical models to consider them. A parser may care that `broken` is a past participle; a search engine or question answering system may care that it is a form of *break*; and a translation system may care about both facts.

Second, in carrying out the above, we will reconstruct morphological paradigms of the language. A paradigm is a grid of all the inflected forms of some lexeme, as illustrated in Table 1.1 on page 2. Our reconstructed paradigms will include our predictions of inflected forms that were never observed in the corpus. This tabular information about the types (rather than the tokens) of the language may be separately useful, for example in translation and other generation tasks, and we will evaluate its accuracy.

Third, in the course of the above, we will estimate **hyperparameters** that describe more general patterns in the language. Among others, we recover the weights of finite-state transducers that serve to relate words within the same paradigm. These constitute a morphological grammar of the language, which can be used to analyze and reinflect novel words.

The model that we will describe in this chapter uses the models we have described in previous chapters as ingredients. While the models from Chapter 2 are just factors in the graphical model from Chapter 3, that graphical model now becomes a factor in the joint probability model we are about to describe.

4.2 Random Variables and Their Values

4.2.1 Value Types

Our probabilistic model considers the following types of mathematical objects. (We use certain lowercase letters to name non-random variables with these types, and certain fonts for constants of these types.)

A **word** w, such as `broken`, is a finite string of any length, over some finite, given alphabet Σ.

A **part-of-speech tag** t, such as VERB, is an element of a certain finite set $\mathcal{T}$, which we typically assume to be given.

w	word (observed)
s	inflection
ℓ	lexeme
Π	inflectional paradigm
D	distribution over inflectional paradigms
θ	parameters defining D
σ^2	prior for θ
H	lexeme-specific distribution over inflections (Dirichlet)
H_0	base distribution for H: general distribution over inflections
ϕ	parameters defining H_0
σ^2	prior for ϕ
α'	Dirichlet concentration parameter for H
G	distribution over lexemes (Dirichlet)
G_0	base distribution for G: uniform distribution over lexemes
α	Dirichlet concentration parameter for G

Figure 4.1: Variables in Figure 4.2 on the next page

An **inflection** s,[1] such as past participle, is an element of a finite set $\mathcal{S}_t$. A token's part-of-speech tag $t \in \mathcal{T}$ determines its set $\mathcal{S}_t$ of possible inflections. For tags that do not inflect, $|\mathcal{S}_t| = 1$. The sets $\mathcal{S}_t$ are language-specific, and we assume here that they are given by a linguist rather than learned. A linguist also specifies features of the inflections: the grid layout in Table 1.1 on page 2 shows that 4 of the 12 inflections in $\mathcal{S}_{\text{VERB}}$ share the "3rd-person" feature.

A **paradigm** for $t \in \mathcal{T}$ is a mapping $\pi : \mathcal{S}_t \to \Sigma^*$, specifying a **spelling** for each inflection in $\mathcal{S}_t$. Table 1.1 on page 2 shows one VERB paradigm.

A **lexeme** ℓ is an abstract element of some lexical space $\mathcal{L}$. In this paper, lexemes have no internal semantic structure: the only question we can ask about a lexeme is whether it is equal to some other lexeme. For present purposes, we will formally take $\mathcal{L}$ to be simply the unit real interval $[0, 1]$. Thus *break* is merely a suggestive nickname for a lexeme such as 0.2538159.

4.2.2 Random Variables

Our generative model of the corpus is a joint probability distribution over the following random variables (denoted by uppercase letters):

- The corpus is represented by a sequence of words $W_1, \ldots, W_N \in \Sigma^*$. In our setting, these are observed, as is the integer N.

[1] We denote inflections by s because they represent "slots" in paradigms (or, in the metaphor of Section 4.3.3, "seats" at tables in a Chinese restaurant). These slots (or seats) are filled by words.

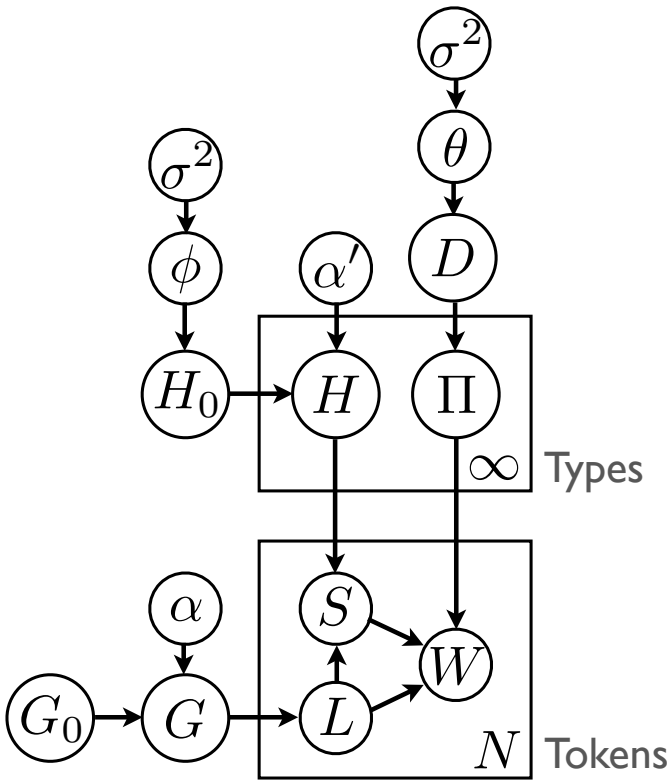

Figure 4.2: Graphical depiction of our probability model. For simplicity, we left out the random variable T that denotes part-of-speech tags. Assume all N tokens have the same part-of-speech tag here. For explanations of the variables in this figure, see Table 4.1 on the previous page.

- Our goal is to recover the corresponding part-of-speech tags $T_1, \ldots, T_N \in \mathcal{T}$ (unless observed), lexemes $L_1, \ldots, L_N \in \mathcal{L}$, and inflections $S_1, \ldots, S_N$, where $(\forall i) S_i \in \mathcal{S}_{T_i}$. These are *token* variables that describe the corpus.

- For each $\ell \in \mathcal{L}$, and each $t \in \mathcal{T}$, the paradigm $\Pi_{t,\ell}$ is a random function from $\mathcal{S}_t \rightarrow \Sigma^*$. For example, Table 1.1 on page 2 shows a possible value for $\Pi_{\text{VERB},break}$. Thus, the various spellings in the paradigm, such as $\Pi_{\text{VERB},break}(\text{1st-person sing. pres.})$=`breche`, are string-valued random variables that are correlated with one another. The paradigms are *type* variables that describe the language.

Note that we have just defined uncountably many paradigms, since $\mathcal{L} = [0, 1]$. Fortunately, at most N of these paradigms were involved in generating the corpus, namely Π_{T_i,L_i} for $1 \leq i \leq N$. As a result, our inference method will be able to integrate out these uncountably many random variables.

What can we say about paradigms that were not used in the corpus? Might they still be part of the language? Our generative model does define a posterior probability distribution over the full language. For each tag $t \in \mathcal{T}$, define the set of **possible lexemes** $\mathcal{L}_t$ as those that have positive probability given that tag. The language's **lexicon** comprises just the paradigms of positive probability, i.e., the values of $\Pi_{t,\ell}$ for all $t \in \mathcal{T}, \ell \in \mathcal{L}_t$. It turns out that our model will generate only languages with countably infinite lexicons (see Section 4.3.2).

4.3 A Dirichlet Process Mixture Model

We focus on the following directed graphical model, sketched in Figure 4.2 on the previous page:

$$\begin{aligned} p(\boldsymbol{W} = \boldsymbol{w}, \boldsymbol{S} = \boldsymbol{s}, \boldsymbol{L} = \boldsymbol{\ell}, \boldsymbol{T} = \boldsymbol{t}, \overrightarrow{\Pi} = \boldsymbol{\pi} \mid \boldsymbol{\tau}, \boldsymbol{\alpha}, \boldsymbol{\phi}, \boldsymbol{\alpha}', \boldsymbol{\theta}) \\ \stackrel{\text{def}}{=} (\boldsymbol{t} \mid \boldsymbol{\tau}) \cdot p(\boldsymbol{\ell} \mid \boldsymbol{t}, \boldsymbol{\alpha}) \cdot p(\boldsymbol{s} \mid \boldsymbol{\ell}, \boldsymbol{t}, \boldsymbol{\phi}, \boldsymbol{\alpha}') \\ \cdot \prod_{t \in \mathcal{T}, \ell \in \mathcal{L}} p(\Pi_{t,\ell} \mid \boldsymbol{\theta}) \cdot \prod_{i=1}^{N} p(w_i \mid \pi_{t_i,l_i}(s_i)) \end{aligned} \quad (4.1)$$

Why this model? Because it corresponds to a simple and plausible generative story for the corpus:

1. Generate the part-of-speech tags,
2. then select an abstract lexeme at each tag,
3. then decide how to inflect each lexeme;
4. finally (in the last factor), obtain the words simply by looking up the spellings of the inflected lexemes in the lexicon of the language (which was generated by the next-to-last factor).

We now define each factor of Equation 4.1 in more detail, and then explain how to view this model as a variant on the Dirichlet process mixture model (Antoniak, 1974).

A key property of our model is that there are only finitely many hyperparameters $\boldsymbol{\tau}, \boldsymbol{\alpha}, \boldsymbol{\phi}, \boldsymbol{\alpha}', \boldsymbol{\theta}$. We can therefore reasonably optimize these to maximize their posterior probability given the finite corpus, i.e., we perform Empirical Bayes (McAuliffe, Blei, and Jordan, 2006) for these hyperparameters. What about all of the parameters (paradigms and probabilities) that describe the behavior of each of the infinitely many lexemes? The model is carefully designed so that we can integrate out these infinitely many parameters. Thus, given a setting of the finitely many hyperparameters, we will be able to do collapsed Gibbs sampling directly over the $O(N)$ token variables $\boldsymbol{T}, \boldsymbol{L}, \boldsymbol{S}$.

In the following subsections, we will explain the different factors in Equation 4.1. See also Figure 4.3 on page 92, which shows how $\boldsymbol{W}, \boldsymbol{T}, \boldsymbol{L}$, and $\boldsymbol{S}$ interact in the Chinese Restaurant process view of the Dirichlet process mixture model.

4.3.1 Part-of-speech Tag Sequence

We use a standard trigram model for the part-of-speech sequence $\boldsymbol{T}$, where the parameter vector $\boldsymbol{\tau}$ specifies the transition probabilities:

$$p(\boldsymbol{t} \mid \boldsymbol{\tau}) \stackrel{\text{def}}{=} \prod_{i=1}^{N+1} p(t_i \mid t_{i-2}, t_{i-1}, \boldsymbol{\tau}) \tag{4.2}$$

Notice that this model is a proper probability distribution over the set of all possible $\boldsymbol{T}$ (of any length), so it implicitly generates $N = |\boldsymbol{T}|$ along with $\boldsymbol{T}$.

4.3.2 Lexeme Sequence

For each part-of-speech tag type $\in \mathcal{T}$, we define a corresponding distribution over lexemes, G_T, so that there is a distribution over verb lexemes, a distribution over noun lexemes, etc. How many verb lexemes (noun lexemes, etc.) are there in a language? We do not know, and we do not impose an upper bound. Instead, we reserve probability mass for infinitely many possible lexemes for each part-of-speech tag.

This is linguistically appropriate since speakers of a natural language are indeed capable of inventing new lexemes and their paradigms. This may happen on request, as in Albright and Hayes (2003), where speakers were asked to inflect nonsense words like `bize`, `shurn` or `bredge`, or driven by the need to name new things, events or actions using neologisms, such as `reuploaded`, `shockvertising` or `un-follow`, to list some relatively recent English neologisms,[2] which may be morphologically inflected.

In short, there is no upper bound on possible lexemes in a language; any new word may be part of some new lexeme; therefore, any possible lexeme has some non-zero probability under our model. As noted above, each of the infinitely many lexemes is indexed by one of the numbers in $[0, 1]$ and has an inflectional paradigm associated (see Section 4.3.4 on page 93).

But what are these distributions G_T and how do we define them?

A distribution G_T in our model is a discrete probability distribution over (countably) infinitely many lexemes, i.e. infinitely many lexemes are possible, but the same lexemes may be drawn repeatedly. In fact, we want a distribution in which most probability mass concentrates on relatively few of the infinitely many lexemes. Lexemes corresponding to words that one would find in a dictionary of the language should be frequently drawn, while

[2] Taken from the lists here, `http://rdues.uce.ac.uk/neologisms.shtml`, accessed on October 11, 2010

others should hardly ever be drawn, like, for example, the lexeme whose paradigm contains the forms `bredge`, `bredging`, `bredges`, etc.

We obtain such probability distributions using the Dirichlet process (Ferguson, 1973). This is a popular tool from the statistics literature and has recently been used for many tasks in natural language processing as well, e.g., HMMs (Beal, Ghahramani, and Rasmussen, 2002a), word segmentation (Goldwater, 2006; Snyder and Barzilay, 2008), parsing (Beal, Ghahramani, and Rasmussen, 2002b; Liang, Petrov, Jordan, and Klein, 2007), machine translation (Post and Gildea, 2009; Yamangil and Shieber, 2010), information retrieval (Haffari and Teh, 2009), the decipherment of lost languages (Snyder, Barzilay, and Knight, 2010), and others.

The Dirichlet process is a prior over discrete distributions with infinite support. As such, the Dirichlet process is a distribution over distributions; sampling from a Dirichlet process means obtaining a discrete probability distribution with infinite support. The samples (i.e., distributions) drawn from a Dirichlet process $DP(G_0, \alpha_T)$ have mean G_0 and *concentration parameter* α_T.

The mean G_0 is the *base distribution*; all distributions drawn from the Dirichlet process will be centered around and have the same support as the base distribution. Although the drawn distributions are discrete, the base distribution is *continuous*, i.e. it has uncountably infinite support. In our model, G_0 is just uniform over all lexemes $\mathcal{L} = [0, 1]$. The relation between the continuous base distribution and the discrete output distributions can be explained by the stick-breaking construction of the Dirichlet process (Sethuraman, 1994). Since the base distribution is continuous, we can use the same base distribution for all $DP(G_0, \boldsymbol{\alpha})$ in our model; the drawn sets of lexemes will be disjoint with probability 1.

The concentration parameter α_T controls how often the same lexemes in the drawn distribution G_T repeat themselves. It can be understood as an inverse variance: A high α_T value means a low variance around the base distribution G_0. This means that the drawn distribution G_T will be close to the base distribution G_0, i.e. the probability mass in G_T will be spread among a high number of lexemes. If α_T is low, on the other hand, the probability mass in the drawn distribution G_T will be spread among a few lexemes only; these will be drawn frequently. This would be desirable for the few lexemes in *closed-class* or nearly-closed class tags. In general, it can be shown that, when we draw n lexemes from a G_T distribution drawn from $DP(G_0, \alpha_T)$, we will observe $O(\alpha_T \log n)$ different lexemes on expectation (Teh, 2010).

In summary, we have the following distributions:

$$G_{t_i} \sim DP(G_0, \alpha_{t_i}) \tag{4.3}$$

$$\ell_i \mid t_i, \alpha_{t_i} \sim G_{t_i} \tag{4.4}$$

We cannot reliably estimate the unknown distributions G_t from finite data, or even represent samples of them during inference, since each G_t is an infinite object. However, by integrating out all the G_t, we obtain a conditional distribution over the random sequence

$\boldsymbol{L}$ given $\boldsymbol{T}, \boldsymbol{\alpha}$. This conditional distribution is much easier to work with. It has a density that can be written in closed form, using a product of interleaved **Chinese restaurant processes** (**CRP**s) (Blei et al., 2004),

$$p(\boldsymbol{\ell} \mid \boldsymbol{t}, \boldsymbol{\alpha}) = \prod_{i=1}^{N} p(\ell_i \mid t_i, \alpha_{t_i},\ \ell_1, \ldots \ell_{i-1},\ t_1, \ldots t_{i-1}) \tag{4.5}$$

where the factor that generates ℓ_i is proportional to $|\{j < i : \ell_j = \ell_i \text{ and } t_j = t_i\}|$ if that integer is positive, and otherwise proportional to $\alpha_{t_i} G(\ell_i)$.

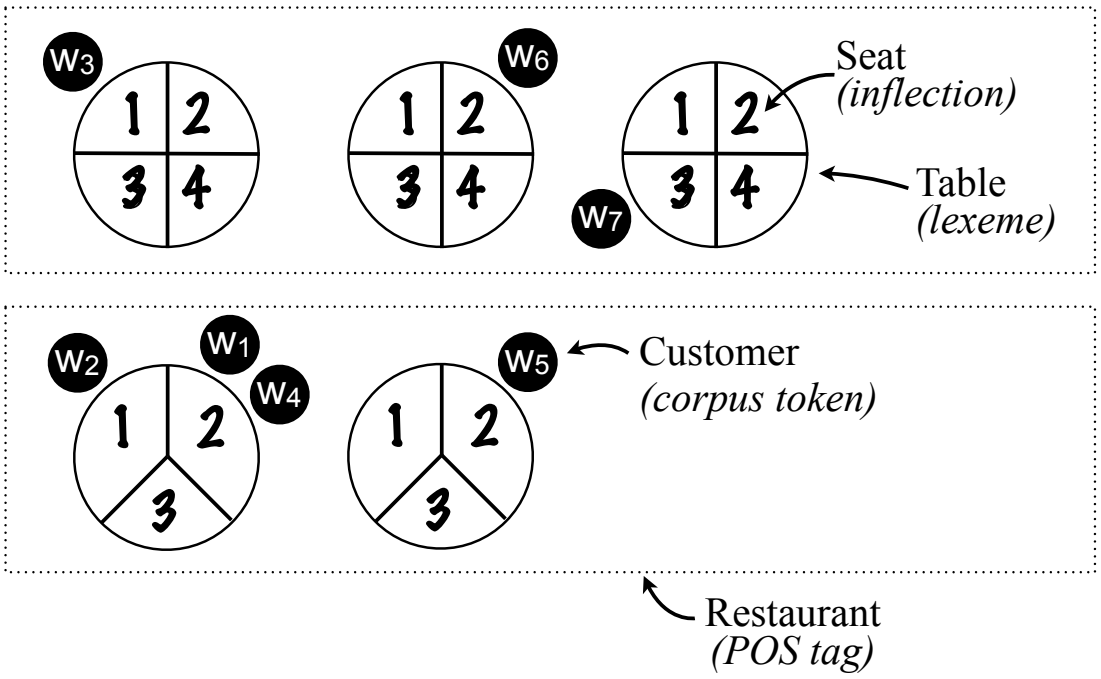

Figure 4.3: Illustration of the Chinese Restaurant Process for inflectional morphology: Each customer sits down in a particular restaurant at a particular table, in a particular seat. This corresponds to each *corpus token* (spelling) picking a *part-of-speech tag*, a *lexeme* and an *inflection*. Each restaurant has potentially infinitely many tables. When new customers enter a restaurant they are more likely to sit down at a more popular table and seat. Note that the inflections in a lexeme constitute an inflectional paradigm, so it is more likely that morphologically related spellings pick the same lexeme (given reasonable $\boldsymbol{\theta}$). Tokens w_1 and w_4, which have the same spelling, have a particular morphological relation with w_2.

Note that the distribution over the lexeme sequence, $p(\boldsymbol{\ell} \mid \boldsymbol{t}, \boldsymbol{\alpha})$, is making a unigram assumption in this model.

The metaphor underlying Equation 4.5 is a generative process in which each tag $t \in \mathcal{T}$ corresponds to a Chinese restaurant. Each restaurant has an infinite number of tables corresponding to the lexemes $\mathcal{L}$. When customer i enters restaurant T_i, he or she chooses a table

L_i to sit at—choosing an already-populated table with probability proportional to the number of customers already at that table, or a random table (sampled from G) with probability proportional to α_{t_i}. Note that this random table will (with probability 1) be empty, since the finite set of already-populated tables has measure 0 under G. See Figure 4.3 on the preceding page for an illustration of the process, which also includes the inflection sequence, whose distribution we will now describe.

4.3.3 Inflection Sequence

In our basic model, we make another unigram assumption about the inflections: S_i depends only on L_i and T_i. In other words, for each tagged lexeme (t, ℓ), the language specifies some distribution $H_{t,\ell}$ over inflections of that lexeme.

What is this distribution $H_{t,\ell}$? First, for each tag $t \in \mathcal{T}$, let H_t be some base distribution over $\mathcal{S}_t$. As $\mathcal{S}_t$ could be large, we will take H_t to be a log-linear distribution with hyperparameters $\boldsymbol{\phi}$. Now we model each $H_{t,\ell}$ as an independent draw from a finite-dimensional Dirichlet distribution with mean H_t and concentration hyperparameter α'_t.

(Mimno and McCallum, 2008) similarly used a log-linear distribution as the mean of a Dirichlet distribution.

We will learn explicit values for $\boldsymbol{\phi}$ and $\boldsymbol{\alpha}'$. However, much as in the previous section, we integrate over the infinitely many lexeme-specific distributions $H_{t,\ell}$. This again gives us a simpler distribution to work with:

$$p(\boldsymbol{s} \mid \boldsymbol{\ell}, \boldsymbol{t}, \boldsymbol{\phi}, \boldsymbol{\alpha}') = \prod_{i=1}^{N} p(s_i \mid \ell_i, t_i, H_{t_i}, \alpha'_{t_i}, s_1, \ldots s_{i-1},\ \ell_1, \ldots \ell_{i-1},\ t_1, \ldots t_{i-1}) \quad (4.6)$$

where the factor that generates s_i is proportional to $|\{j < i : s_j = s_i \text{ and } (t_j, \ell_j) = (t_i, \ell_i)\}| + \alpha'_{t_i} H_{t_i}(s_i)$.

Instead of explaining this via Pólya urn processes (the finite analogue of the CRP), we enrich the CRP from the previous section. Each table ℓ in Chinese restaurant t has a fixed, finite set of seats corresponding to the inflections $s \in \mathcal{S}_t$. When customer i chooses to sit at table L_i, as described in the previous section, he or she also chooses a seat S_i at that table—choosing an already-occupied seat (on someone's lap) with probability proportional to the number of customers already in that seat, or a random seat (sampled from H_t and not necessarily empty) with probability proportional to α'_t.

4.3.4 Paradigms

Each tagged lexeme (restaurant table) is associated not only with a probability of its own and a distribution over inflections (seats), but also with a paradigm of spelled-out words as in Table 1.1 on page 2. We assume that each of the infinitely many lexemes (tables) of

tag t independently draws its paradigm $\Pi_{t,\ell}$ from some distribution D_t over paradigms, parameterized by $\boldsymbol{\theta}$.

Naturally, this probability distribution over paradigms, D_t, is modeled by a *finite-state Markov Random Field (MRF)*, the novel multiple-string approach that we presented in Chapter 3. Therefore, for a paradigm π, the probability $D_t(\pi \mid \boldsymbol{\theta})$ is proportional to a product of non-negative factors of the form $\text{WFSA}_s(\pi(s))$ and $\text{WFST}_{s,s'}(\pi(s), \pi(s'))$ (for various inflections $s, s' \in S_t$), see Equation (3.1) on page 60. Here WFSA_s is a weighted finite-state acceptor that evaluates the inflected word $\pi(s) \in \Sigma^*$, typically using an n-gram character language model. It gives a high score to spellings that are appropriate (in this language) for s-inflected words. Similarly, $\text{WFST}_{s,s'}$ is a weighted finite-state transducer that evaluates how well the two spellings $\pi(s)$ and $\pi(s')$ go together (in this language), summing over all of the possible ways of aligning them. For example, $\text{WFST}_{\text{present,past}}$ might model the fact that the spellings of a lexeme's present and past inflections are typically related by a specific change to the word ending or stem vowel. We use a general architecture for the WFSAs and WFSTs, with language-specific weights derived from the learned hyperparameters $\boldsymbol{\theta}$ (Dreyer and Eisner, 2009).

4.3.5 Spell-out

The final factor in Equation 4.1 represents a deterministic *spell-out* step. Given the tag, lexeme, and inflection at position i, we generate the word W_i simply by looking up its spelling in the appropriate paradigm. That is, $p(W_i = w \mid \Pi_{T_i,L_i}(S_i) = w') = 1$ if $w = w'$, and $= 0$ otherwise.

To account for typographical errors in the corpus, the spell-out process could easily be made nondeterministic, with the observed word W_i derived from the correct spelling $\Pi_{T_i,L_i}(S_i)$ by a noisy channel model (e.g., Toutanova and Moore, 2002) represented as a WFST. This would make it possible to analyze `brkoen` as a misspelling of a common or contextually likely word, rather than treating it as an unpronounceable, irregularly inflected neologism, which is presumably less likely.

4.3.6 Discussion: Clustering and DPMMs

The lexeme sequence $\boldsymbol{L}$ induces a *partition* of $1, 2, \ldots N$ into groups of tokens that share a lexeme. Our model assigns higher probability to partitions $\hat{L}$ that group similar words into one paradigm. Why?

Suppose that in the process of generating the corpus, we have already generated `discombobulated` twice, as the past tense of lexeme ℓ. We are now generating a new token W_i. We do not have a strong bias against having many lexemes; in fact, if $\alpha_{\text{VERB}} > 2$, then we would *rather* pick a new lexeme than reuse the so-far-rare lexeme ℓ. However, *given that* the next word is observed to be `discombobulating`, we are far more likely to have obtained it as the present participle of ℓ, which is already expected to be

`discombobulating` or something similar, than by choosing a new lexeme and picking a new paradigm for it that *just happens* to contain the spelling `discombobulating`. The former explanation is more likely *a posteriori* because it avoids the cost of generating that specific unlikely string from scratch. Instead, it predicts that string from a previously generated string, a cost that can be kept small by setting the WFSA/WFST weights $\boldsymbol{\theta}$.

This clustering of words is much like clustering of points in $\mathbb{R}^d$, a well-studied problem. Such points are commonly clustered using a mixture model. A non-parametric version that allows unboundedly many clusters is the **Dirichlet process mixture model** or **DPMM** (Neal, 2000), from which we will borrow inference methods.

The DPMM generative story is similar to ours: The tables in an infinite Chinese restaurant are identified with distributions ℓ, such as Gaussians. Each customer $i = 1, 2, \ldots$ enters the restaurant and chooses an old or new table L_i to sit at, with probabilities exactly as in Section 4.3.2. (If a new table is chosen, it is drawn from some prior G over Gaussians.) The customer then assumes a spatial position $X_i \in \mathbb{R}^d$, sampled from the chosen Gaussian L_i. Each ℓ tends to generate points x_i in some region of $\mathbb{R}^d$. Conversely, observed points in the same small region of $\mathbb{R}^d$ are most easily explained as being generated from a single ℓ, giving a clustering behavior.

Our basic model of inflectional morphology is similar. In our setting, each table generates words W_i in a "region" of Σ^*, rather than points X_i in a region of $\mathbb{R}^d$. There are three differences:

First, we have a separate DPMM for each tag t.

Second, each table ℓ in tag t's DPMM is associated not with a Gaussian distribution over all of $\mathbb{R}^d$, but rather with an "inflectional distribution" over Σ^* that assigns positive probability to only (at most) $|\mathcal{S}_t|$ strings in Σ^*. This "inflectional distribution" is parameterized by a paradigm of strings, $\Pi_{t,\ell}$, together with a multinomial $H_{t,\ell}$ over the finitely many inflections in the paradigm.

Third, in a small departure from standard DPMMs, we opted in Section 4.3.2 to identify tables with distinct elements of $\mathcal{L} = [0, 1]$—"lexemes"—rather than with inflectional distributions over words. Each table's inflectional distribution is constructed later (sections 4.3.3–4.3.4). Thus, we disentangled the notion of an abstract lexeme (cluster) from the information associated with it. Our construction makes lexemes $\ell \in \mathcal{L}$ and inflections $s \in \mathcal{S}$ into first-class variables of the model, which can be (1) observed (Section 4.5.3), (2) inferred (Section 4.4), (3) modulated by additional factors (Section 4.8.2), or (4) associated with other linguistic information. The use of lexemes also permits polysemy, where two lexemes remain distinct despite having the same paradigm or inflectional distribution.

4.4 Inference

4.4.1 Collapsed Gibbs Sampling

Just as in a DPMM, our basic inference technique is **collapsed Gibbs sampling**. Given the corpus $\boldsymbol{w}$ and the hyperparameters, we will obtain samples from the posterior distribution

$$p(\boldsymbol{S}, \boldsymbol{L}, \boldsymbol{T} \mid \boldsymbol{W} = \boldsymbol{w},\ \boldsymbol{\tau}, \boldsymbol{\alpha}, \boldsymbol{\phi}, \boldsymbol{\alpha}', \boldsymbol{\theta}) \tag{4.7}$$

defined from the model (4.1). The sampler is initialized with some legal value of $(\boldsymbol{S}, \boldsymbol{L}, \boldsymbol{T})$, and modifies it stochastically one step at a time (Section 4.4.3). As in any Monte Carlo Markov Chain (MCMC) method, the tuple reached after m steps is a random quantity, whose distribution converges to the desired posterior distribution as $m \to \infty$.

The sampler is said to be *collapsed* because it only represents the values of $\boldsymbol{S}, \boldsymbol{L}, \boldsymbol{T}$. It does not represent the distributions G_t and $H_{t,\ell}$ or the paradigms $\Pi_{t,\ell}$. These infinitely large or infinitely numerous objects do not appear in (4.7) because they have been integrated out, i.e., we sum over all of their possibilities.

For convenience, our sampler is collapsed in another way as well. It does not actually represent $\boldsymbol{L}$, because we do not practically need to recover each lexeme L_i as an arbitrary value in $\mathcal{L} = [0, 1]$. The state of our sampler will thus represent only the *partition* $\hat{L}$ induced by $\boldsymbol{L}$: that is, a partition of $1, 2, \ldots N$ into groups of tokens that share a lexeme. We represent this partition computationally as a finite collection of non-empty "table" objects, each one labeled with some tag t and pointing to (and pointed to by) some disjoint group of customers i such that $T_i = t$. This state may be regarded as a *collapsed particle*—a uniform distribution over all values of $\boldsymbol{L}$ consistent with $\hat{L}$.

4.4.2 Reconstructing the Paradigms

Although the paradigms are collapsed out of the sampler state, it is not too hard to *reconstruct* the posterior distribution over any table's paradigm Π, given the collapsed sample and the hyperparameters. This is needed to evaluate our model and also, as we will see, to determine the probability of a stochastic move.

Recall from Section 4.3.4 that the prior distribution D_t over a table's paradigm Π is a finite-state graphical model. Crucially, the posterior distribution over Π depends only on the set of 0 or more words w_i sitting at the table, along with their seats s_i.

These words are observations that were generated using Π. For each one, we know that $\Pi(s_i) = w_i$ (because the spell-out factor of Section 4.3.5 is deterministic). Thus—conditioned on the state of the sampler—we *know* the spelling $\Pi(s)$ for each non-empty seat s at the table. This gives us a partially filled-in paradigm.

How about the other seats? We showed in Section 3.4 on page 64 how to run belief propagation on the string-based graphical model D_t, which calls various finite-state algorithms as subroutines.

Applying this method, we obtain the marginal distribution over each unknown spelling $\Pi(s')$ given the known spellings $\Pi(s)$. In general, this marginal distribution is represented as a probabilistic finite-state machine.

We find it convenient for each table in our sampler to maintain an approximate posterior distribution over each spelling in its paradigm. There are 3 cases:[3]

1. For a known spelling $\Pi(s)$ at a non-empty table, a single value has probability 1.

2. For an unknown spelling $\Pi(s')$ at a non-empty table, store a truncated distribution that enumerates the 1000 most likely strings, according to belief propagation.

3. All empty tables (in a given restaurant t) are identical, so we maintain a single generic empty table of tag t, where we again perform belief propagation. For each spelling $\Pi(s')$ at the empty table, a 1000-best list would not cover enough possibilities, so store a probabilistic finite-state acceptor that can score any string.

A hash table based on cases 1 and 2 can be used to map any word w to a list of seats at non-empty tables that might plausibly have generated w. As for the seats at the generic empty tables, we can score w using the finite-state acceptors from case 3 to see whether any of them might have generated w.

4.4.3 How to do Gibbs Sampling

Given the above, we can build a Gibbs sampler similar to that for a Gaussian DPMM (Rasmussen, 2000). For each token $i = 1, 2, \ldots n$ of the corpus, the sampler state specifies the observed word w_i, a tag t_i, a table $\hat{\ell}_i$ representing a particular but unknown lexeme, and an inflection s_i. A move of the Gibbs sampler chooses a token i uniformly at random, and resamples the triple $(t_i, \hat{\ell}_i, s_i)$ from its posterior distribution given the remaining state of the sampler. That is, it resamples the **location** of customer i (possibly starting a new table).

This move really serves to morphologically reanalyze token w_i in its current context. In our CRP metaphor, it corresponds to making customer i invisible and then guessing where she is probably sitting—which restaurant, table, and seat?—given knowledge of w_i and the locations of all other customers.

A well-known and easily verified property of our Chinese restaurant processes is their *exchangeability*. Once we know what set of customers will enter restaurant t, the probability that they will choose particular tables and seats does not depend on the order in which they enter.[4] Hence, when guessing whether i chose a particular seat given all the other customers, we may as well pretend that she was the *last* customer to enter. This will not change the answer and makes it easier to see what the answer must be.

[3] The first two cases must in general be recomputed whenever a seat at the table switches between being empty and non-empty. All cases must be recomputed if the hyperparameters $\boldsymbol{\theta}$ change.

[4] Given a tagging $\boldsymbol{T}$, the subsequence $\langle (L_i, S_i) : T_i = t \rangle$ is defined for each $t \in \mathcal{T}$ and is exchangeable.

Concretely, the sampler chooses location $(t_i, \hat{\ell}_i, s_i)$ with probability *proportional* to the product of

- $p(T_i = t_i \mid T_{i-2}, T_{i-1}, T_{i+1}, T_{i+2}, \boldsymbol{\tau})$
- the probability (from Section 4.3.2) that a new customer in restaurant t_i would choose table $\hat{\ell}_i$, given all the *other* customers in that restaurant
- the probability (from Section 4.3.3) that a new customer at table $\hat{\ell}_i$ would choose seat s_i, given all the *other* customers at that table
- $p(w_i \mid t_i, \hat{\ell}_i, s_i,$ seats of other customers at table $\hat{\ell}_i)$

All of these factors depend on the sampler's current state. The last factor is determined from the reconstructed paradigm $\Pi_{t_i,\hat{\ell}_i}$ as it stands *without* customer i (Section 4.4.2). It is just the probability of w_i according to the posterior distribution over the spelling $\Pi_{t_i,\hat{\ell}_i}(s_i)$. For most locations $(t_i, \hat{\ell}_i, s_i)$, this last factor is 0 or extremely small, and so the Gibbs sampler does not need to consider the location at all. The end of Section 4.4.2 explained how to identify all plausible locations for a word w_i, possibly including some seats at generic new tables.

4.4.4 Sampling Strategies for Faster Mixing

Gibbs sampling for a DPMM can be slow to mix—often, resampling customer i will not actually change its location, because it is immediately attracted back to her friends at the seat it was just removed from.

We implemented two alternatives that are instances of type-based MCMC (Liang, Jordan, and Klein, 2010):

- **Lexeme-based sampler:** A move in the Gibbs sampler consists of first picking one of the currently active lexemes uniformly at random, removing all its customers at once. Then, picking new tables and seats for these customers in random order. We can then run intermediate Gibbs sampling steps for just these customers, before we move on to another lexeme. One iteration of sampling is finished when N customers have been resampled in this way.
- **Spelling-based sampler:** This is similar to the lexeme-based sampler, but here we pick a spelling with probability proportional to its token frequency in the corpus,[5] and remove all similarly spelled tokens from their current restaurant locations. Now we can proceed as in the lexeme-based sampler.

Another technique would be to introduce Metropolis-Hastings moves that split and merge entire tables (Jain and Neal, 2004).

[5] In other words, we pick a certain corpus position and pick its spelling.

4.5 Training the Hyperparameters

4.5.1 Unsupervised Training

The goal of training is to adjust our hyperparameters to maximize the log-likelihood of the observed corpus $\boldsymbol{w}$,

$$\log p(\boldsymbol{W} = \boldsymbol{w} \mid \boldsymbol{\tau}, \boldsymbol{\alpha}, \boldsymbol{\phi}, \boldsymbol{\alpha}', \boldsymbol{\theta}) \tag{4.8}$$

A basic strategy is **Monte Carlo EM**. In the **E step**, we materialize an approximation to the posterior distribution (4.7) by drawing a number of samples from it, using our sampler from Section 4.4. Each sample is a possible annotation $\boldsymbol{s}, \hat{\ell}, \boldsymbol{t}$ of the corpus $\boldsymbol{w}$. In the **M step**, we adjust our hyperparameters to maximize or at least increase the average log-probability of these samples, using the supervised methods of the next section. The **E** and **M** steps are alternated.

This learning method is similar to Wallach (2006) and Mimno and McCallum (2008).

4.5.2 Supervised Training

The M step is a supervised training problem, since it must increase the average log-probability under (4.1) of fully observed samples $(\boldsymbol{w}, \boldsymbol{s}, \hat{\ell}, \boldsymbol{t})$.[6] This objective separates into several maximization problems, one per factor of (4.1).

It is straightforward to train the tag sequence model τ from samples of $\boldsymbol{T}$ (Section 4.3.1).

Next, we train the prior model of lexeme sequences (Section 4.3.2). For each tag t, we must estimate α_t. It is not too hard to see from the Chinese restaurant process that for a lexeme partition $\hat{\ell}$ and a tagging $\boldsymbol{t}$, we have

$$\log p(\hat{\ell} \mid \boldsymbol{t}, \alpha_t) = r_t \log \alpha_t - \sum_{i=0}^{n_t - 1} \log(i + \alpha_t) + \text{const}^{7} \tag{4.9}$$

where r_t is the number of tables in restaurant t and n_t is the number of customers in restaurant t. Quantity (4.9) (and, if desired, its derivative with respect to α_t) may be easily found from $\boldsymbol{t}$ and $\hat{\ell}$ for each of our samples. Maximizing its average over our samples is a simple one-dimensional optimization problem.

Note that Equation 4.9 will be replaced by Equation (4.15) on page 103 later when we consider a speedup, in which we explicitly assign tables only to some customers, while summing over possible tables for others (Section 4.5.4).

[6] We also add the log prior probability of the hyperparameters.

[7] The constant accounts for probability mass that does not depend on α_t.

For the inflection sequence model (Section 4.3.3), we must similarly estimate α'_t for each t, and also $\boldsymbol{\phi}$:

$$\log p(\boldsymbol{s} \mid \hat{\ell}, \boldsymbol{t}, \boldsymbol{\phi}, \alpha'_t) = \sum_{\ell} \left(\sum_{s \in \mathcal{S}_t} \sum_{i=0}^{n_{t,\ell,s}-1} \log(i + \alpha'_t H_t(s)) - \sum_{i=0}^{n_{t,\ell}-1} \log(i + \alpha'_t) \right) + \text{const} \quad (4.10)$$

where ℓ ranges over the r_t tables in restaurant t, and $n_{t,\ell}$ is the number of customers at table ℓ, of which $n_{t,\ell,s}$ are in seat s. The summation over s may be restricted to s such that $n_{t,\ell,s} > 0$. Recall that H_t is the base distribution over seats: $H_t(s) \propto \exp(\boldsymbol{\phi} \cdot \boldsymbol{f}(s))$. For a given value of $\boldsymbol{\phi}$ and hence H_t, we can easily compute quantity (4.10) and its gradient with respect to α_t. Its gradient with respect to $\boldsymbol{\phi}$ is $\alpha'_t \sum_{s \in \mathcal{S}_t} (c_s - c H_t(s)) \boldsymbol{f}(s)$, for $c = \sum_{s' \in \mathcal{S}_t} c_{s'}$ and

$$c_s = H_t(s) \sum_{\ell} \sum_{i=0}^{n_{t,\ell,s}-1} \frac{1}{i + \alpha'_t H_t(s)} \quad (4.11)$$

Finally, we estimate the parameters $\boldsymbol{\theta}$ of our prior distribution D_t over paradigms of the language (Section 4.3.4), to maximize the total log-probability of the partially observed paradigms at the tables in restaurant t (averaged over samples). This can be done with belief propagation, as explained by Dreyer and Eisner (2009). Crucially, each table represents a *single* partially observed sample of D_t, regardless of how many customers chose to sit there.[8] The training formulas consider our posterior distributions over the spellings at the empty seats (Section 4.4.2). In general, however, a table with many empty seats will have less influence on θ, and a completely empty table contributes 0 to our total-log-probability objective. This is because the probability of a partially observed paradigm marginalizes over its unseen spellings.

4.5.3 Semi-supervised Inference and Training

In addition to the corpus $\boldsymbol{w}$, we observe a small set of paradigms of the language (for each tag t). This should help us find a reasonable initial value for θ. For evaluation purposes, we also observe a set of *partial* known paradigms. In our experiments, these partial paradigms are derived from simple lists of uninflected words, so each one specifies *only* the spelling of the lemma inflection. The training and inference process will complete these

[8] In other words, $\boldsymbol{\theta}$ generates types, not tokens. Each of the uncountably many lexemes prefers to generate a paradigm that is likely under θ (Section 4.3.4), so the observation of *any* lexeme's paradigm provides information about θ. The fact that some tables have higher probability is irrelevant, since (at least in our model) a lexeme's probability is uncorrelated with its paradigm.

paradigms —by placing tokens from the corpus in their inflectional slots, or by computing marginals— and these completed paradigms can then be evaluated.

We condition our inference and training on our knowledge that the model generated these complete and partial paradigms along with the corpus $\boldsymbol{w}$. However, since the paradigms are *type* data rather than token data, we must posit a new generative process for them:

Suppose we observe k_t semi-supervised paradigms $\boldsymbol{\pi}$ for tag t. We assume that their lexemes $\boldsymbol{\ell}$ were sampled independently (with replacement) from the language's distribution G_t. This implies that these lexemes tend to have reasonably high probability.

It is easy to modify the generative process of (4.1) on page 89 to account for these observations. Our posterior estimate of the distribution G_t is implicit in the state of the CRP once the CRP has generated all N tokens. To sample k_t additional lexemes from G_t, we simply see which table (lexeme) the *next* k_t customers pick. Each of these customers brings its whole (complete or incomplete) paradigm to the table. We call such a customer **host** because it is at a table without actually taking any particular inflectional seat. It just stands by the table, reserving it and welcoming any future customer that is consistent with the observed paradigm at this table. In other words, just as an ordinary customer in a seat constrains a single spelling in the table's paradigm, a host standing at the table constrains the table's paradigm to be consistent with the complete or partial paradigm that was observed in semi-supervised data.

The exchangeability of the CRP means that the hosts can be treated as the first customers rather than the last. To modify our Gibbs sampler, we ensure that the state of a restaurant t includes a **reserved table** for each of the distinct lexemes (such as *break*) in the semi-supervised data. Each reserved table has (at least) one host who stands there permanently, thus permanently constraining its paradigm. Notice that without the host, ordinary customers would have only an infinitesimal chance of choosing this specific table (lexeme) from all of $\mathcal{L} = [0, 1]$, so we would be unlikely to complete this semi-supervised paradigm with corpus words.

The training of the hyperparameters from a sample is exactly as before. Notice that $\boldsymbol{\theta}$ will have to account for the partial paradigms at the reserved tables even if only hosts are there (see footnote 8). The hosts are counted in n_t in Equation 4.9 when estimating α_t. However, as they are not associated with any inflectional seat, they have no effect on estimating α'_t or $\boldsymbol{\phi}$; in particular, within Equation 4.10, interpret $n_{t,\ell}$ as $\sum_s n_{t,\ell,s}$, which excludes hosts.

4.5.4 Speeding up Inference by Summarizing Parts of the Corpus

When using large corpora $\boldsymbol{w}$, we may want to speed up the inference process by considering only certain parts of interest, while regarding the remaining parts only in summarized

form. We utilize information from the reserved tables (Section 4.5.3) to find out what parts of the corpus are of particular interest.

As described above, each table has a host that does not sit in any particular inflectional seat; this host has (supervised) information about (some or all of) the inflectional seats at that table. For those particular seats, the host will only welcome customers from the corpus that match the preferred spelling; all other customers are rejected. This creates preferences for the other seats as well: Given the distribution D_t over paradigms at that table and the spelling information for supervised seats, we can run inference (Section 3.4 on page 64) and obtain marginal distributions over the preferred spellings for the unsupervised seats that the host has no information about.

As a result, every seat at a reserved table has a required spelling or a distribution over plausible spellings, before any customer sits down. We prune each seat's distribution to its k-best values. We say that the host knows about all seats at its table, via supervised information or inferred k-best list for that seat, and allows customers to sit down only if their spelling matches that information.

We speed up inference by considering only those customers as **visible** that are welcomed by at least one host at a reserved table in a restaurant; all other tokens with the same part-of-speech tag are only regarded as a big group of *other*, or **invisible**, customers; we acknowledge that they are there and may sit down at some tables — but by definition not at any of the reserved ones — but we ignore their spellings. We do not explicitly assign them to tables in the restaurant; we collapse this information in the sampler and sum over all possible table locations for them. This sum, again, excludes some of the tables, which is valuable information.

Since we ignore the spellings of invisible customers, we can omit from the objective function the terms that would generate their spellings, which also means that we can ignore their seating arrangements by summing those out.

Why do we not just filter them out completely from the corpus? It would create a bias: The visible customers are concentrated at a few tables (the reserved tables). If we observed only those it could cause us to underestimate the concentration parameter α, which regulates how many tables we create. As described above, we know something about the invisible customers: They do *not* sit at any reserved tables (after all, they were not welcome there). This is an observed fact about any corpus sample. Since in practice most customers are invisible, the large fraction of such customers implies that the reserved tables are much less popular than they would appear if we had only visible customers. Therefore we want a higher estimate of alpha than we would get with visible customers only; this, of course, means that even visible customers will be more willing to sit at unreserved tables.

A sample from the MC E step will contain, for each visible customer, its current restaurant, table and seat location; for the invisible customers we only know which restaurant

they are in.[9] Given such a sample, we would like to find the most likely value of α_t at the M step. We do this as follows:

- Let n_t be the total number of customers and hosts in a particular restaurant.
- Let $c_t <= n_t$ be the number of invisible customers in that restaurant.
- Let $r_t <= n_t$ be the number of tables with at least one visible customer or host in the restaurant.
- Let $u_t >= 0$ be the number of visible customers sitting at unreserved tables in the restaurant.

We want to compute the probability that *all* the customers and hosts would sit as observed in the sample. We appeal to the Chinese Restaurant Process. For simplicity, suppose without loss of generality that the visible customers and hosts entered the restaurant before the invisible customers.

For the first visible customer or host at a table, the probability of them choosing that table was $\alpha_t/(\alpha_t+i)$, where i is the number of customers and hosts already in the restaurant.

For each additional visible customer or host at that table, the probability of them choosing that table was $(\text{count})/(\alpha_t + i)$, where count is the number of previous customers or hosts at that table already; note that the numerator is independent of alpha.

For each invisible customer, the probability of them choosing an unreserved and possibly novel table (which is all we know about them from the sample) is $(\alpha_t+u_t+j)/(\alpha_t+i)$, where j is the number of previous invisible customers. This fraction represents the total probability of all tables where the invisible customer could have sat.

It follows that the probability of the sampled lexeme sequence as a function of α_t is

$$p(\hat{\ell} \mid \boldsymbol{t}, \alpha_t) = (\alpha_t^r * \text{const} * \prod_{j=0}^{c_t-1}(\alpha_t + u_t + j))/(\prod_{i=0}^{n_t-1}(\alpha_t + i)) \tag{4.12}$$

Taking the log and simplifying, we obtain

$$\log p(\hat{\ell} \mid \boldsymbol{t}, \alpha_t) = r_t \log \alpha_t + \sum_{j=0}^{c_t-1} \log(\alpha_t + u_t + j) - \sum_{i=0}^{n_t-1} \log(\alpha_t + i) + \text{const} \tag{4.13}$$

$$= r_t \log \alpha_t - (\sum_{i=0}^{u_t-1} \log(\alpha_t + i) + \sum_{i=u_t+c}^{n_t-1} \log(\alpha_t + i)) + \text{const} \tag{4.14}$$

$$\approx r \log \alpha_t + f(\alpha_t, 0) - f(\alpha_t, u_t) + f(\alpha_t, u_t + c_t) - f(\alpha_t, n_t) + \text{const} \tag{4.15}$$

[9] In the experiments in this chapter, the restaurant is always determined by an *observed*, if noisy, part-of-speech tag.

where we define $f(\alpha_t, x) = \int \log(\alpha_t + x)dx = (\alpha_t + x)\log(\alpha_t + x) - x$.
The step from Equation 4.14 to Equation 4.15 can be made by observing that

$$\sum_{i=j}^{k-1} \log(\alpha_t + i) \approx \int_{x=j}^{k} \log(\alpha_t + x)dx \tag{4.16}$$

$$= f(\alpha_t, k) - f(\alpha_t, j) \tag{4.17}$$

Equation 4.15 can be minimized numerically using Brent's method (Brent, 2002) or any gradient-based method using its derivative with respect to alpha, $(r/\alpha_t + \log(\alpha_t) - \log(\alpha_t + u_t) + \log(\alpha_t + u_t + c_t) - \log(\alpha_t + n_t))$.

As a side note, our method of analyzing only *some* words—the ones that seem interesting based on prior knowledge—in a corpus, while treating others as "invisible", may be of interest from a cognitive point of view. Consider Chan's criticism (Chan, 2006) of Gibbs sampling (referring to the Gibbs sampler in (Goldwater, Griffiths, and Johnson, 2006)):

> Such approaches are not suitable as cognitively [sic] models of human acquisition, because they are based upon iterative refinement of a grammar covering the entire data set, and this would predict that child learners have grammars that reproduce every construction they hear.

This criticism would not affect our sampler, since we do not cover the entire data set and do not explicitly model the exact lexemes, inflections or spellings of every construction.

4.5.5 Obtaining Results

In order to obtain results that we can then evaluate against the truth, we can record the current state of the variables at any time, typically after each EM iteration.

We may be interested in recording the current state of certain paradigms or of the corpus tagging.

To record the state of a particular inflectional seat s in a particular paradigm $\pi_{t,\ell}$ at time m, we obtain its marginal probability $p^m(\pi_{t,\ell}(s))$ using belief propagation.[10] From this probability, we extract the k-best strings and store each of them together with its probability. We repeat this whenever we record the results and report the string with maximum expected probability, $\operatorname{argmax}_w \sum_m p^m(w)$ for that inflection. This can be done for all inflectional seats in paradigms of the lexemes we are interested in; these are lexemes that were added as semi-supervised data using just the lemma and a host.

Recording the token variables $(\boldsymbol{T}, \boldsymbol{L}, \boldsymbol{S})$ of the corpus is simpler since all of them are clamped to a particular value at all times. To report the result at time m and corpus position i we simply output the (t, l, s) tags that have been observed most frequently at this position during the m time steps.

[10] Obviously, at a seat that has one or more customers w this probability is just 1 for the spelling of those customers and 0 for all other spellings.

4.6 Experiments

4.6.1 Experimental Design

In this section, we describe experiments and results on real-world data. The task is to learn the inflectional morphology of a language.

4.6.1.0.1 DATA

Seed paradigms. We use the same inflectional paradigms as described in Section 3.7.3.1 on page 72. To summarize here briefly, we use sets of 50 and 100 complete seed paradigms. The smaller sets are subsets of the larger ones. Having sets of different sizes enables us to obtain learning curves. For each size, we have 10 different sets, so that we can get more stable, averaged results.

In each split, we have a test set of 5,415 lexemes in which the lemma is given and all other forms are missing. These are the forms we will evaluate on.

For details, see Section 3.7.3.1.

Unannotated text corpus. The novelty in this chapter compared to Chapter 3 is that we take advantage of and learn an unbounded number of paradigms discovered from additional unannotated text data. In our experiments, we use subsets of the WaCky corpus (Baroni et al., 2009), taking the first 1 million or 10 million words.

4.6.1.0.2 SIMPLIFICATIONS

In order to efficiently utilize large text corpora, we make two simplifications to our probability model in Equation (4.1) on page 89: (1) Although the model is capable of tagging the word sequence $\boldsymbol{W}$ of the corpus with part-of-speech tags $\boldsymbol{T}$ as part of the inference, we condition on an observed, though noisy, tag sequence instead. This tag sequence is delivered as part of the WaCky corpus and was automatically assigned using a part-of-speech tagger. (2) We focus on finding verbal paradigms, so during inference we assign (ℓ, s) tags only to words marked in $\boldsymbol{T}$ as verb.

As described in Section 4.3.4 on page 93, we use the finite-state Markov Random Fields that we developed in Chapter 3 to model inflectional paradigms. In particular, we use the locally normalized version described in Section 3.3.2 on page 62, which allows for very fast belief propagation.

The topology is star-formed, where the lemma form in the center is connected to all other forms. Therefore, the probability of a paradigm can simply be expressed as

$$p_{\boldsymbol{\theta}}(\pi) = p_{\boldsymbol{\theta}}(x) \prod_i p_{\boldsymbol{\theta}}(y_i \mid x) \tag{4.18}$$

where x is the lemma form and y_i are other forms in the paradigm π. Belief propagation is fast in the associated factor graph because whenever a form y_i is currently not observed—

which is often the case during inference since most inflections of most verbs will never occur in the text—the random variable does not have to send any message.

This star-formed directed graphical model over strings is different from what we referred to as *separately predicting output forms from the lemma form* in Chapter 3, since in this chapter, the *lemma* forms are often *unobserved*. This is the case whenever we have opened a new lexeme table and placed forms other than the lemma in it. In this case, there is indeed information flow from all forms placed in the paradigm so far to all other so far unobserved forms.

Note that some linguists argue for such a star-formed topology, where derived forms are built by transforming a *base* into the desired form (Albright, 2002; Chan, 2006), as opposed to derived forms influencing each other directly. We do not commit to any particular view point and chose the base-centered approach here for greater efficiency.

4.6.1.0.3 INITIALIZATION OF THE SAMPLER

We initialize the variables and parameters of our model as follows:

- Initialize the $\boldsymbol{\theta}$ parameters that define the distribution over paradigms, D_t by training from the small number of observed seed paradigms (see Section 3.5).
- Initialize the $\boldsymbol{\phi}$ parameters that define the lexeme-independent distribution over inflections uniformly at random from the interval $[-0.01, 0.01]$.
- Initialize α and α' to 1.
- Initialize $\boldsymbol{L}$ and $\boldsymbol{S}$. The straightforward way to do this would be to draw $\boldsymbol{L}$ and $\boldsymbol{S}$ from Equation 4.7 (T is here clamped to the observed $\boldsymbol{T}$), where the samples are drawn in random order. However, we make two changes to obtain a better initialization:

  1. $p(\Pi_{t,\ell})$ is used in sampling to provide marginal probabilities of a spelling w at a particular seat s of a particular lexeme ℓ. We found in early experiments that multiplying such marginal probabilities with a discriminatively trained probability distribution $p(\boldsymbol{S} \mid \boldsymbol{W})$ gives a better initialization. This distribution is trained from all (w, s) pairs contained in the seed paradigms. It is a simple log-linear model, where the features $\boldsymbol{f}(w, s)$ fire on unigrams, bigram and trigrams of w.[11] The feature weights for this simple maximum-entropy model were trained using MegaM (Daumé III, 2004). Note that this distribution is used only to obtain an initial tagging of the corpus, but not during inference.
  2. We sample the initial ℓ and s for all corpus positions in the order of confidence: First, determine that order by walking through the corpus positions in random

[11] w is annotated with a start and an end symbol.

order and sampling ℓ and s values, but without actually tagging the corpus or incrementing any counts in the Chinese Restaurant Process. Then, sort all positions by the probability of the drawn sample and obtain the actual initial tagging of the corpus by sampling $\boldsymbol{L}$ and $\boldsymbol{S}$ in that order (still using $p(\boldsymbol{S} \mid \boldsymbol{W})$ as described above).

In this way, corpus positions that we are sure about get tagged first, and this will help determining the ℓ and s tags for positions that we are less sure about. As an example from English morphology, suppose we are initially not sure if *cuss* is a third-person singular or a lemma form of some lexeme. If we tag words we are confident about first and these include the word *cusses*, for which the third-person singular tag is very likely, it will be easy to decide later that *cuss* is the lemma form in the same paradigm that *cusses* was placed in.

Once all variables are initialized in the described way, we start training and decoding. We do this on all 10 data sets.

We update $\boldsymbol{\theta}$ every 5 iterations using L-BFGS (Liu and Nocedal, 1989a). The parameters α, α' and $\boldsymbol{\phi}$ are updated after every iteration. Maximizing α and α' are simple one-parameter optimization problems, which makes Brent's algorithm (Brent, 2002) the obvious choice. For updating $\boldsymbol{\phi}$ we chose to run a few iterations of the RProp optimization algorithm (Riedmiller and Braun, 1992).[12] We avoided gradient descent for the optimization problems here because of its need for a learning rate schedule.[13]

For sampling, we use the lexeme-based method with no additional intermediate sampling steps, for speed.

These particular initialization and runtime settings have been determined to work well in early experiments, where we trained on just the first two data splits and evaluated on the corresponding development data.

4.6.2 Training π

Each of the probability distributions $p_{\boldsymbol{\theta}}(x)$ and $p_{\boldsymbol{\theta}}(y_i \mid x)$ that constitute D_t (see Equation (4.6.1.0.2) on page 105) is modeled by a finite-state machine (Chapter 2). For training $\boldsymbol{\theta}$, we consider the current state of the sampler, compute the marginal $p(x)$ in each of the currently active paradigms, which requires running belief propagation (Section 3.4 on page 64) in all paradigms where X is not currently observed by a customer sitting at the lemma inflectional seat. We then train all $p_{\boldsymbol{\theta}}(y_i \mid x)$ separately, using the $(p(x), y)$ pairs as training corpus and maximizing Equation (2.4) on page 34.

[12] L-BFGS would have been a reasonable choice too, but it typically takes more iterations to converge than RProp; for optimizing $\boldsymbol{\theta}$, L-BFGS was the only possibility because of the potentially divergent objective function (see Section 2.8.2 on page 37).

[13] However, there exist variants that do not require such a schedule, e.g., (Crammer, Dekel, Keshet, Shalev-Shwartz, and Singer, 2006).

Note that from this training procedure we obtain an *unnormalized* finite-state machine $u_{\boldsymbol{\theta}}(x, y)$[14] (see Section 2.2.2 on page 20), which can be normalized given a specific (distribution over) x, but is not a generally normalized distribution $p_{\boldsymbol{\theta}}(x)$ for all x. In order to obtain such a normalized distribution we divide the finite-state transducer $u_{\boldsymbol{\theta}}(x, y)$ by the normalizing machine $u_{\boldsymbol{\theta}}(x) = \sum_y u_{\boldsymbol{\theta}}(x, y)$, which contains the normalization constants for all x. That sum, however, obtained by projection, results in a big, highly non-deterministic machine. Therefore we approximate it by constructing $u'_{\boldsymbol{\theta}}(x)$, which has bigram topology, and minimizing the Kullback-Leibler divergence $KL(u_{\boldsymbol{\theta}}||u'_{\boldsymbol{\theta}})$. A $p_{\boldsymbol{\theta}}(y \mid x)$ obtained by normalizing $u_{\boldsymbol{\theta}}(x, y)$ by $u'_{\boldsymbol{\theta}}(x)$ is only approximately normalized.

4.6.3 Results

Table 4.1 on the next page and Table 4.2 on page 110 show the results on running with no corpus, versus with a corpus of size 10^6 and 10^7 words. The results are measured in whole-word accuracy (Table 4.1) and average edit distance (Table 4.2).

As can be seen from the accuracy result (row "all"), just using 50 seed paradigms, but no corpus, gives an accuracy of 89.9%.[15] By adding a corpus of 10 million words we reduce the error rate by 9.9%, corresponding to a one-point increase in absolute accuracy to 90.9%.

A similar trend can be seen when we use more seed paradigms. Simply training on 100 seed paradigms, but not using a corpus, results in an accuracy of 91.5%. Adding a corpus of 10 million words to these 100 paradigms reduces the error rate by 8.3%; this corresponds to about half-point increase to 92.2%.

Compared to the large corpus, the smaller corpus of 1 million words goes more than half the way; it results in error reductions of 6.9% (50 seed paradigms) and 5.8% (100 seed paradigms). Interestingly, when we count how many of the spellings from all the 5,616 CELEX paradigms we use can be found in the smaller and in the larger corpus, we find that the smaller corpus contains more than half the spellings that the larger corpus contains (7,376 versus 13,572 of all 63,778 spellings).

As mentioned in Section 3.7, a baseline that simply inflects each morphological form according to the basic regular German inflection pattern, reaches an accuracy of only 84.5% and an average edit distance to the correct forms of 0.27. All improvements are statistically significant; we used a pairwise permutation test ($p < 0.05$).

The improvements we have reported in this section so far are roughly similar to the orthogonal improvements we reported in Section 3.7.[16] Here we use a simple star-formed factor graph topology, where messages flow from observed inflected forms through the lemma to unobserved forms, but we use a large unannotated corpus in addition; there we use

[14] After composition with π_x^{-1} and $\pi(y)$.

[15] This is the same accuracy number as shown in Table 3.1 on page 74 column *sep*, since we simply train a star-formed factor graph on the seed paradigms.

[16] But see large differences in the token-based analysis below.

no corpus but more complex factor graph topologies. These techniques could potentially be combined to get further improvements. However, a more complex factor graph would substantially slow down the inference process (Section 4.4 on page 96), which already takes several days on a corpus of 10 million words.

	50 paradigms			100 paradigms		
Form	0	10^6	10^7	0	10^6	10^7
13PIA	74.81	78.31	**81.49**	81.39	82.04	**84.68**
13PIE	**99.98**	99.87	99.82	**99.98**	99.86	99.77
13PKA	74.67	77.75	**81.95**	81.15	81.55	**83.56**
13PKE	**99.92**	99.86	99.68	**99.93**	99.84	99.74
13SIA	84.24	**84.79**	84.61	**85.82**	**85.71**	83.53
13SKA	83.48	87.65	**87.96**	86.19	87.46	**88.23**
13SKE	**99.75**	98.11	98.73	**99.68**	**99.71**	99.41
1SIE	**99.64**	99.28	98.15	**99.49**	**99.51**	98.63
2PIA	**83.98**	81.84	81.97	**85.87**	84.92	85.26
2PIE	98.12	**99.21**	99.16	98.14	**99.32**	99.28
2PKA	**83.03**	79.61	77.16	85.15	**85.39**	84.36
2PKE	**99.88**	**99.88**	99.81	**99.90**	**99.89**	99.87
2SIA	**83.75**	83.32	82.53	85.78	**85.99**	85.98
2SIE	91.12	91.59	**91.91**	94.20	94.52	**94.63**
2SKA	82.23	82.35	**82.66**	84.98	**85.30**	85.13
2SKE	**99.90**	**99.90**	99.87	**99.91**	**99.91**	99.90
3SIE	93.88	95.76	**95.91**	94.41	95.80	**95.94**
pA	59.76	67.81	**70.80**	63.39	69.11	**70.80**
pE	**99.38**	**99.38**	99.37	**99.39**	**99.39**	99.38
rP	97.81	**98.58**	98.33	98.14	**99.14**	99.07
rS	**98.67**	98.40	97.88	98.71	**98.98**	98.82
all	89.89	90.61	**90.92**	91.49	92.04	**92.17**

Table 4.1: Whole-word accuracy on various morphological forms. The numbers 0, 10^6 and 10^7 denote the size of the corpus used. 50 (100) is the number of seed paradigms. All results are statistically significant under a paired permutation test, $p < 0.05$.

	50 paradigms			100 paradigms		
Form	0	10^6	10^7	0	10^6	10^7
13PIA	0.42	0.36	**0.32**	0.34	0.32	**0.29**
13PIE	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
13PKA	0.43	0.37	**0.32**	0.35	0.34	**0.31**
13PKE	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
13SIA	0.43	0.41	**0.39**	0.40	**0.39**	0.40
13SKA	0.34	0.28	**0.27**	0.30	**0.27**	**0.27**
13SKE	**0.00**	0.01	0.01	**0.00**	**0.00**	**0.00**
1SIE	**0.01**	**0.01**	0.02	0.01	**0.00**	0.01
2PIA	**0.39**	0.42	0.44	**0.36**	0.38	0.37
2PIE	0.02	**0.00**	**0.00**	0.02	**0.00**	**0.00**
2PKA	**0.33**	0.38	0.43	0.31	**0.30**	0.34
2PKE	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
2SIA	**0.39**	**0.39**	0.42	**0.36**	**0.36**	0.37
2SIE	0.10	**0.09**	**0.09**	0.07	**0.06**	**0.06**
2SKA	**0.34**	**0.34**	**0.34**	0.31	**0.30**	0.31
2SKE	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
3SIE	0.07	**0.05**	**0.05**	0.07	**0.05**	**0.05**
pA	0.91	0.74	**0.68**	0.84	0.71	**0.69**
pE	0.01	**0.00**	**0.00**	0.01	**0.00**	**0.00**
rP	0.02	**0.01**	**0.01**	0.02	**0.00**	**0.00**
rS	**0.02**	**0.02**	0.03	0.02	**0.01**	**0.01**
all	0.20	0.19	**0.18**	0.18	**0.17**	**0.17**

Table 4.2: Average edit distance of the predicted morphological forms to the truth. The numbers 0, 10^6 and 10^7 denote the size of the corpus used. 50 (100) is the number of seed paradigms. The numbers in bold denote the best model that uses 50 (100) seed paradigms as well as statistically undistinguishable ones.[18]

Token-based Analysis. We now conduct a token-based analysis of these results, similar to the analysis in Section 3.7.3.2, see Page 76. We use the same frequency counts for inflected forms, based on the 10-million-words corpus, see Page 76, and we use the same five frequency bins (see Table 3.4 on page 77).

The analysis here reveals large performance gains for frequent verb inflections, which tend to be more irregular (Jurafsky et al., 2000, p. 49); such inflected forms can be predicted much better after learning from an unannotated corpus. On inflections that occur more

	50 paradigms			100 paradigms		
Bin	0	10^6	10^7	0	10^6	10^7
1	90.50	90.95	**91.29**	92.09	92.41	**92.58**
2	78.07	**84.51**	84.42	80.17	**85.51**	85.06
3	71.64	**79.28**	78.08	73.29	**80.18**	79.09
4	57.35	61.36	**61.77**	57.35	**62.04**	59.86
5	20.73	**25.00**	**25.00**	20.73	**25.00**	**25.00**
all	52.58	57.47	**57.79**	53.41	**58.48**	57.76

Table 4.3: Token-based analysis: Effect of a using an unannotated corpus (10^6 or10^7 words) on whole-word accuracy of predicted inflections, separated into frequency bins. (Bin 1: least frequent, most regular.)

than 10 times in the large corpus (see Page 77), the 1-million-words corpus increases the absolute prediction accuracy by 6.4%, 7.6%, 4.0% and 4.3% (see Table 4.3, Frequency Bins 2, 3, 4 and 5) when 50 seed paradigms are used. When 100 seed paradigms are used, the absolute gains are similar, though slighly smaller (5.3%, 6.9%, 4.7%, 4.3%).

Surprisingly, the gains that we observe from the 10-million-words corpus are often smaller than the gains from the 1-million-words corpus. On more frequent forms, the large corpus does increase the performance of the no-corpus baseline, but often by not as much as the smaller corpus. This may indicate that the model cannot utilize the corpus perfectly; it learns regularities at the cost of some irregularities, which hurts it in this token-based analysis. We expect that further improvements to the model (see Section 4.8 on page 118) would result in a more steady learning curve across all frequency bins.

In general, however, adding a corpus, be it 1 or 10 million words, increases the prediction accuracy across all frequency bins, often dramatically. The performance on the *less* frequent inflections in Bin 1 increases steadily as we increase the size of the corpus from 0 to 1 to 10 million words. The performance on the *more* frequent, more irregular inflections increases greatly, especially when the 1-million-words corpus is used.

Note that, by using a corpus, we even improve our prediction accuracy for forms and spellings that are *not* found in the corpus, i.e., *novel* words.[19] In the token-based analysis above we have already seen that prediction accuracy does increase for forms that are never or extremely rarely found in the corpus (Bin 1). We add two more analyses that more explicitly show our performance on novel words.

- We find all paradigms that consist of novel spellings only, i.e. none of the correct spellings can be found in the corpus.[20] The whole-word prediction accuracies for

[19]The reason for this is that we update the general FST parameters ϕ based on the spellings of corpus samples.

[20]This is measured on the largest corpus used in inference, the 10-million-words corpus, so that we can evaluate all models on the same set of paradigms.

the models that use corpus size 0, 1 million, and 10 million words are, respectively, 94.0%, 94.4%, 94.4% using 50 seed paradigms, and 95.1%, 95.4%, 95.3% using 100 seed paradigms.

- Another, simpler measure is the prediction accuracy on all forms whose correct spelling cannot be found in the 10-million-word corpus. Here we measure accuracies of 91.6%, 92.1% and 91.9%, respectively, using 50 seed paradigms. With 100 seed paradigms, we have 93.0%, 93.5% and 93.3%. The accuracies for the models that use a corpus are higher, but do not always steadily increase as we increase the corpus size.[21]

By definition, the novel words that we just tested on are rare; their spellings do not occur in the large corpus. One might wonder why the accuracy on such, presumbably relatively regular forms is below 96%. We look at the errors that the models make.

Table 4.4 on the next page shows some typical errors that the models without a corpus make and that are fixed in the models that use a corpus; the listed errors are on novel words. Most errors are due to an incorrect application of an irregular rule that was learned from the seed paradigms. The corpus model learns not to apply these rules in many cases. The seed paradigms are not very representative since they are drawn uniformly at random from all types in CELEX.[22] But from a corpus the model can learn that some phenomena are more frequent than others.

Even if the prediction accuracy is overall higher when a corpus is used in addition to the seed paradigms—there are also errors that are introduced from the use of a corpus. Such errors are shown in Table 4.5 on the following page. Here, often a form that is found in the corpus is used instead of the correct one.

For example, the past participle form of *bitzeln* was predicted to be *besselt*. The correct form would be *gebitzelt*, but that does not occur in the corpus, while *besselt* does occur. The pair (*bitzeln*, *besselt*) is also morphologically somewhat plausible considering the correct pair (*sitzen*, *gesessen*) in German.[23] Similarly, *salierten* was predicted as a past-tense form of *silieren*. The correct form *silierten* does not occur in the corpus, while *salierten* does. *Salierten* is somewhat plausible due to the common `i`→ *a* change, as in (*bitten*, *baten*), so the morphological grammar did not give a very strong signal to prevent *salierten* from being (mis-)placed in the *silieren* paradigm.

Overall, the errors in Table 4.5 help explain why the edit distance results in Table 4.1 improve by only small fractions while the corresponding whole-word accuracy improvements are greater (Table 4.2): The corpus models make fewer errors, but the erors they do make can be more severe. In some cases, the corpus component may force a corpus token into a paradigm slot where the finite-state parameters otherwise would have generated a spelling that is closer to the truth. On the other hand, we have seen that the corpus is

[21] The differences are statistically significant, according to the paired permutation test, $p < 0.05$.

[22] In the future, we might want to experiment with more representative seed paradigms.

[23] In both pairs, we have the changes `i`→ `e` and `tz`→ `ss`.

Form	Error (no corpus)	Correct	Explanation
aalen, 2PIA	aieltest	aaltest	*ie* as in *(halten, hieltest)*
flügeln, pA	flügelt	geflügelt	no ge- as in *(erinnern, erinnert)*
welken, pA	gewolken	gewelkt	wrong analogy to *(melken, gemolken)*
prüfen, 2SIA	prüfst	prüftest	no *-te-* as in *(rufen, riefst)*

Table 4.4: Novel words and typical errors that a no-corpus model makes. These errors are corrected in the model that has learned from a corpus. Most errors come from an incorrect application of some irregular rule picked up from the seed paradigms (see the *Explanation* column).

Form	Error (corpus)	Correct	Explanation
bitzeln, pA	besselt	gebitzelt	used corpus token
ringeln, 13SIE	riegle	ring(e)le	*unclear*; incorrect rule
silieren, 13PIA	salierten	silierten	used corpus token
bearbeiten, 2SIA	bearbeitest	bearbeitetest	*bearbeitest* is frequent

Table 4.5: Novel words and typical errors that a corpus model makes. These errors are not made by the no-corpus baseline model. Often, a spelling that can be found in the corpus was preferred instead of the correct spelling.

often helpful in providing evidence on how high certain irregular constructions should be weighed in the morphological grammar.

This concludes our analysis. The token-based analysis we have conducted in this section shows the strength of the corpus-based approach presented in this chapter; we have seen large gains especially on more frequently occurring forms (Table 4.3 on page 111). The corpus as part of the overall graphical model (see Figure 4.2 on page 88) often plays a corrective role; it fixes the predictions that the integrated string-based Markov Random Field (MRF) from Chapter 3 makes. The MRF is good at learning the morphology directly from the seed paradigms; but the few seed paradigms cannot give much evidence of the complex rules and exceptions that would be necessary to correctly predict regular and irregular verbs; often irregular rules are applied too broadly or indistinctly. Integrating a corpus addresses that insufficiency; the corpus implicitly contains much more evidence than the sample paradigms. Building a joint model that combines the string-based MRFs and corpus-based methods into a single joint model combines the strengths of both models, and even if the corpus currently introduces some new errors, the overall prediction accuracy improves.

4.6.4 Some Statistics about the Inference Process

Here we briefly list some statistics that show how corpus size affects the inference process in our experiments: When we use the 10-million-words corpus, each lexeme has 88 customers on average; while each lexeme has only 11 customers with the 1-million-words corpus. With 10 million words, 1,516 of the 5,415 tables remain empty;[24] with 1 million words that number increases to 2,978. Still, most inflectional seats are empty: 96,758 of 113,715 inflectional seats are empty when using 10 million words, while 107,040 are empty using 1 million words.

4.7 Related Work

We have described a generative, non-parametric approach to the semi-supervised discovery of morphological paradigms from plain text, which is based on a probability model over the text, its hidden lexemes, paradigms, and inflections. We emphasize that our model seamlessly handles nonconcatenative and concatenative morphology alike. We now compare our approach to other methods of unsupervised or semi-supervised discovery of morphological knowledge from text.

4.7.1 Concatenative Approaches

Most previous work on unsupervised morphological discovery has modeled *concatenative* morphology only, making the simplifying assumption that the orthographic form of a word can be split neatly into stem and affixes. This, of course, is inappropriate for many languages (Kay, 1987), even for English, which is morphologically relatively impoverished, but has stem changes, consonant doublings and other irregular phenomena. However, concatenative morphology has proved to be a useful modeling assumption that made much progress in morphological knowledge discovery possible. Such work is often concerned with segmenting words of a corpus into their morphemes, or smallest meaningful units, thus unsupervisedly learning the morpheme inventory of a language. Some approaches cluster such morphemes into groups. A group of affixes can be regarded as a simple form of inflectional paradigm, as described below.

(Harris, 1955) was the first to automatically find morpheme boundaries and segment words accordingly. He proposed that, at any given point in a word, a morpheme boundary is likely if there is a large set of possible next letters, given an unannotated text corpus. As an example, in the word *deny* it is immediately clear that there is a morpheme boundary between *den* and *y* because other letters can occur after *den-*, as in *denies* and *denial*. Almost twenty years later, this approach was reinterpreted in information theoretic terms

[24] Remember that every paradigm that we evaluate on has a reserved, initially empty table during inference (see Section 4.5.3 on page 101).

by (Hafer and Weiss, 1974), who measured the confusability of possible next letters with conditional entropy and other measures.

Later (Déjean, 1998) integrated these distributional ideas into a bootstrapping procedure similar to Yarowsky (1995): He first used Harris' method to find common prefixes and suffixes in a corpus. For example, *-ing* was extracted as a common suffix because the letter before *-ing* is very unpredictable—it could be almost any letter in the alphabet. He then expanded this initial list by extracting other prefixes and suffixes in the corpus that appear at similar places as the affixes already found in the first step. As a final step, he used this expanded affix list to segment all words in the corpus.

Another frequently used approach to morphological segmentation is based on the Minimum Description Length (MDL) principle (Rissanen, 1989), which has its roots in information theory and learning theory. According to this principle, we can think of segmentation as a way of compressing the corpus. Each segment is assigned a codeword. The best segmentation of a corpus is the one that allows for the shortest codewords as well as the shortest corpus when it is expressed using the codewords. There is a trade-off involved: To find the shortest codewords one would, in the extreme case, assign each letter its own code; to find the shortest corpus one would, in the extreme case, assign the whole corpus a single codeword. The objective is to find the ideal balance.

(Brent, Murthy, and Lundberg, 1995) apply MDL to find the most common suffixes in a corpus. They assume that each word consists of a stem and a (possibly empty) suffix. They consider a list of candidate suffixes ranked by the ratio of the relative frequency of the sequence (e.g., *-ing*) and the frequency of its individual letters (*i*, *n*, *g*). Then they test suffixes in that order and pick them greedily if they decrease the description length.

(Goldsmith, 2001b) uses MDL as well in his *Linguistica* analysis system. He finds what he calls *signatures*—sets of affixes that are used with a given set of stems, for example *(NULL, er, ing, s)*. The number of found signatures is very large. (Monson, Lavie, Carbonell, and Levin, 2004) reduces such large numbers of signatures by connecting signatures in a network, where signatures that share affixes are connected, and applying heuristic pruning strategies based on occurrence counts.

An interesting variant of Goldsmith's work is (Goldwater et al., 2006), which is a probabilistic variant of Goldsmith's model. It is related to our model in that it is a Bayesian approach that uses Gibbs sampling for inference. It performs morphological segmentation of an unsupervised corpus, allowing only a stem and one suffix for each word. A word is generated by first picking a conjugation class, then picking a stem given the class, then picking a (possibly empty) suffix given the class. There are infinitely many classes, stems, and suffixes; they are modeled by Dirichlet distributions. For a given word, any position (presumably starting after the first letter) is considered as its segmentation point. (Naradowsky and Goldwater, 2009) later extended this model to be able to handle simple nonconcatenative phenomena as well through the use of spelling rules (see discussion below).

(Snyder, Naseem, Eisenstein, and Barzilay, 2008) and (Johnson, Goldwater, and Griffiths, 2008) use Bayesian approaches to segmentation as well. (Snover and Brent, 2001) unsupervisedly discover sets of affixes.

(Creutz and Lagus, 2002) propose two models for segmentation, one based on MDL, one based on maximum likelihood.

(Poon, Cherry, and Toutanova, 2009) use a global log-linear model for morphological segmentation, in which they add global priors to the objective function that are reminiscent of the MDL principle: A *lexicon prior* favors a small lexicon length, defined as the total number of characters in the morpheme lexicon. This simultaneously reduces the number of morpheme types as well as their respective lengths. And a *corpus prior* favors a corpus segmentation in which words are split into as few morphemes as possible. Note that these two priors pull the segmenter into different directions: The lexicon prior favors shorter morphemes to keep the lexicon simple, while the corpus prior favors longer morphemes to avoid over-segmentation of the words in the corpus. The model is trained on unsupervised data to maximize the objective function, finding a balance between these priors.

In the annual Morpho Challenge (Kurimo, Virpioja, Turunen, and Lagus, 2010), participants compete in a morphological segmentation task; ParaMor (Monson, Carbonell, Lavie, and Levin, 2007) and Morfessor (Creutz and Lagus, 2005) are influential contenders.

(Chan, 2008) presents a morphology model called *base-and-transform* that consists of lexical categories, morphological base forms (lemmas) and simple suffix rules (transforms) that convert a base form to a derived (inflected) form, e.g., *-e* $\rightarrow$ *-ing*. The vocabulary is clustered into base words, derived words and unmodeled words. In earlier work, Chan (2006) learns sets of morphologically related words using Latent Dirichlet Allocation (Blei, Ng, and Jordan, 2003). He calls these sets *paradigms* but notes they are not substructured entities, like the paradigms we model in this thesis. The paradigm discovery in Chan's work is restricted to concatenative and regular morphology.

4.7.2 Nonconcatenative Approaches

Morphology discovery approaches that handle *nonconcatenative* and irregular morphological phenomena are more closely related to our work; they are rarer.

Yarowsky and Wicentowski (2000a) present an original approach to identify inflection-root pairs in large corpora without supervision. This approach is able to align even highly irregular pairs like `take`, `took` through the use of distributional clues. The motivating example for the main idea is the verb `sing` and two possible past-tense candidates, the irregular `sang` and the regular `singed`. How can we decide without supervision which one is the correct past tense of `sing`? Both occur in corpora (the latter as past tense of the verb `singe`), so both may be plausible. An important insight of their approach is that the frequency ratios of a root to its past tense are relatively stable across different lexemes. Therefore, the frequency ratio of regular word pairs like `walk` and `walked`, which are easy to identify, should be roughly similar to the frequency ratio of `sing` to its correct past

tense. Measures based on such frequency ratios are used as one of several features that guide the identification of root-inflection pairs. Other features are based on (distributional) context similarity, string edit distance, rewrite rule probabilities including stem changes and suffixes, and the pigeonhole principle. This principle says that (1) a root should not have more than one inflection of a certain part-of-speech tag, and (2) different inflections of the same part-of-speech tag should not have the same root. In our work, we use (1), which is, in our terminology, a paradigm should not have more than one spelling in a given cell.[25] This principle can simplify a model and is almost always appropriate, although there are some exceptions like the alternative forms `dreamed` and `dreamt`, as noted in (Yarowsky and Wicentowski, 2000a).

(Schone and Jurafsky, 2001) take a large corpus and extract whole conflation sets, like "abuse, abused, abuses, abusive, abusively, ...". They use surface word similarity as well as distributional similarity as clues, which enables them to extract irregular forms as well. (Baroni, Matiasek, and Trost, 2002) have a similar objective, but they use mutual information as a measure of semantic similarity, proposing that words that occur near each other are semantically related.

(Naradowsky and Goldwater, 2009) is an extension of the Bayesian model in (Goldwater et al., 2006), described above, to some simple nonconcatenative cases. They add spelling rules that apply to the morpheme boundary in a word; such a rule can be an insertion or a deletion of a character (but not a substitution). That way, they can analyze `shutting`, which contains consonant doubling, as having the stem `shut` and suffix `-ing`, with an insertion rule applied to the boundary. Simpler, purely concatenative models like the ones described in the previous subsection would have to analyze as either `shutt` and `-ing` or `shut` and `-ting`, both of which are wrong. The model in (Naradowsky and Goldwater, 2009) is similar to ours in that it models some nonconcatenative processes, although in a much more restricted manner, and is a Bayesian non-parametric approach. As in our work, Gibbs-sampling inference and empirical Bayes hyperparameter estimation are used.

In summary, we have first contrasted our work against previous work in morphological knowledge discovery that reduces morphology to its purely concatenative properties. We have then described that some authors advanced the state of the art by using token frequency, context similarity and simple spelling rules to model irregular and nonconcatenative morphology. Compared to these approaches, ours is partially less and partially more advanced. In terms of spelling and transformation rules, our approach is by far the most flexible and adaptable of all, capable of learning any finite-state transformation between words in a flexible probability model that allows for linguistic features. On the other hand, we have not added frequency and contextual information to our model, but we regard this as an interesting opportunity for future work (Section 4.8.2 on page 119).

[25] In (Yarowsky and Wicentowski, 2000a), the part-of-speech tags are the commonly used Wall Street Journal tags which mark some inflectional forms, like 3rd person, or past participle. In our work, the part-of-speech tags are coarse (Section 4.3.1 on page 90), but are then subsequently split by the inflectional forms, s.

Of all the approaches we listed, none has attempted to model highly structured inflectional paradigms that exactly specify the spellings of all inflectional forms of the lexemes of a language. Describing such paradigms in a well-defined probability model that allows for arbitrary, nonconcatenative morphological processes, equipped with an inference algorithm that works on large corpora is the main contribution of this work.

4.8 Remaining Challenges and Future Work

As described above, we believe our model (4.1) is a solid account of handling inflectional morphology in a principled generative framework. Furthermore, we think this dissertation could be a starting point to continue and explore this work further into different directions.

4.8.1 Topic Models

Integrating topic information might be a useful extension of the presented work. If each word in the corpus were tagged with a topic, we could exploit this as a signal for better clustering behavior. As an example, consider a paradigm currently filled with the spelling `break`. It might, due to a flat or wrong distribution D_t over paradigms, have a low probability for the irregular `broke` as its past tense. However, one would hope that the topic distributions of `break` and `broke` were similar, providing a signal that `broke` should rather be in the `break` paradigm than in other candidate paradigms.

A specific way to integrate topics would be to run Latent Dirichlet Allocation (LDA; Blei et al., 2003) or a similar unsupervised topic model on the text corpus first. This will give each word token an observed topic, which it brings with it to the table and seat that it picks during the sampling inference algorithm. This information can be used to split the tables, which have already been split into inflectional seats, further, along the topics that its customers bring. The more customers at a certain table have a certain topic the more likely it is that a new customer with the same topic will sit down there. This model would be another instance of a hierarchical Dirichlet process, with a fixed number of classes, like in the inflection case (Section 4.3.3 on page 93).

It would also be possible to use the distributions $p(\text{word} \mid \text{topic})$ obtained from LDA as base distribution for a Dirichlet process over topics. In this case, we use the topic tagging that we get from LDA as initialization only. Our Gibbs sampler would now be able to resample the topic of a spelling: A customer that sits down in some seat at the table also orders a certain individual dish (topic), in proportion to the base distribution and the number of customers at this table already eating the same dish. From this process, we could hope to get a better topic tagging of the corpus than LDA does because tokens with different (but morphologically related) spellings can now influence each other, even over long distances, since they sit together at different tables.

4.8.2 Adding Context to the Model

Context is important for morphological disambiguation (Smith, Smith, and Tromble, 2005). It is particularly important for unsupervised learning of morphology, since there may be several types of external ("syntagmatic") as well as internal ("paradigmatic") clues to the correct analysis of a word, and these can effectively bootstrap one another during learning (Yarowsky and Wicentowski, 2000b).

In particular, inflections can be predicted to some extent from surrounding inflections, lexemes, and tags. Verbs tend to agree with their preceding nouns. We see that `broken` is a past participle because like other past participles, it is often preceded by the lexeme *have* (or simply by the particular word `has`, a pattern that might be easier to detect in earlier stages of learning). Immediate context is also helpful in predicting lexemes; for example, certain verb lexemes are associated with particular prepositions.

Lexemes are also influenced by wide context. `singed` is not a plausible past tense for `sing`, because it is associated with the same topics as `singe`, not `sing` (Yarowsky and Wicentowski, 2000b).

How can we model context? It is easy enough to modulate the probability of the sampler state using a *finite* number of new contextual features whose weights can be learned. These features might consider inflections, tags, and common words. For example, we might learn to lower the probability of sampler states where a verb does not agree in number with the immediately preceding noun. This is simply a matter of multiplying some additional factors into (4.1) and renormalizing. This yields a Markov random field (MRF), some of whose factors happen to be distributions over lexemes. The sampler is essentially unchanged, although training in a globally normalized model is more difficult; we expect to use contrastive divergence for training (Hinton, 2002).

It is more difficult to incorporate lexeme-specific feature templates. These would lead to infinitely many feature weights in our non-parametric model, leading to overfitting problems that cannot be solved by regularization. Such feature weights must be integrated out. Three basic techniques seem to be available. We can use richer non-parametric processes that still allow collapsed sampling—e.g., we can use a Hierarchical Dirichlet Process (Teh, Jordan, Beal, and Blei, 2006) to make lexeme probabilities depend on a latent topic, or a Distance-Dependent CRP (Blei and Frazier, 2009) to make lexeme probabilities depend on an arbitrary notion of context. We can multiply together several simple non-parametric processes, thus generating the lexemes by a product of experts. As a last resort, we can always do uncollapsed sampling, which integrates over arbitrary lexeme-specific parameters by including their values explicitly in the state of our MCMC sampler. The sampler state only needs to represent the parameters for its finitely many non-empty tables, but reversible-jump MCMC techniques (Green, 1995) must be used to correctly evaluate the probability of moves that create or destroy tables.

4.8.3 Derivational and Agglutinative Morphology

The work we have presented deals with *inflectional* morphology; our model can change the spelling of words to reflect different morphological properties like number, person, tense, etc. However, these changes do not affect the part of speech of a word. There can be one paradigm for the verb *to act*, and another for the noun *action*, and yet another one for the adjective *actionable*; they are unrelated. In each each of these paradigms then, the typical verb, noun or adjective transformations apply. *Derivational* morphology, on the other hand, is concerned with changing the meaning of words by applying transformations that may change their parts of speech, e.g., relating the verb *to act* to the noun *action*, or the adjective *happy* to the noun *happiness*. In future work, we could include such transformation in our model. We could use the separate inflectional paradigms that we have described in this and the previous chapter and add derivational factors that describe the typical derivational relationships, connecting noun, verb and adjective paradigms. Figure 4.4 on the next page shows an example, where simple derivational factors turn the adjective *wide* into the nouns *width* and *wideness* and the verb *widen*. Each of these paradigms for *width*, *wideness* or *widen* is then a standard inflectional paradigm.

Note that further derivational factors can create longer and more exceptional variants, such as a factor that changes *wideness* into *widenessless*, and *widenessless* into *widenesslessness*, and so forth. Each of these behaves like a standard noun; there is no particular cell for a *-less* noun, for example.

At this point, it becomes clear that this is also the way we propose to model *agglutinative* morphology in languages like Finnish or Turkish. These languages often use words like *unsystematicalness* (*epäjärjestelmällisyys* in Finnish), which can be modeled in our framework with several inflectional paradigms connected by derivational factors that turn the noun *system*, which has its own inflections, into the adjective *systematic*, which also has its own inflections, and into another adjective *systematical*, and so on. To summarize, there is no one cell that describes all morphological properties of *epäjärjestelmällisyys*; rather it is modeled as a normal noun, which is derived from a negated adjective, which in turn is derived from an adjective, etc. That way, we avoid the combinatorial explosion of morphological properties in agglutinative languages.

4.8.4 Identifying Free-floating Verb Prefixes

In German, some verbs may be prepended with a prefix like `ab-`, `an-`, `auf-`, `weg-`, as in `ablegen`, `anlegen`, `auflegen`, `weglegen`. These prefixes may be detached in a main clause, for example:

- Wir legen ab. ('We put off, take off, ...')
- Wir legen an. ('We put on, invest, connect, ...')
- Wir legen auf. ('We hang up.')

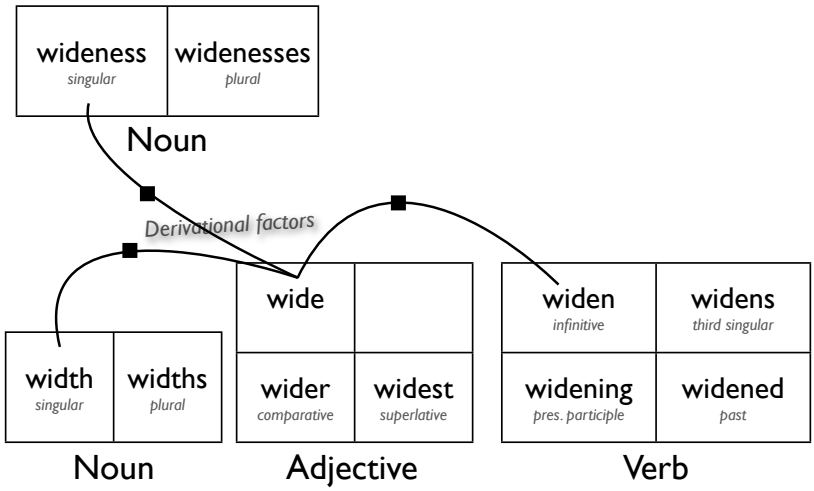

Figure 4.4: Derivational morphology in our framework: Derivational factors connect different inflectional paradigms with one another, encoding typical transformations, such as appending the suffix *-ness* to a verb to create a noun. Each inflectional paradigm is then modeled in the standard way (see, e.g., Figure 3.3 on page 62).

- Wir legen weg. ('We put away.')

These detachable prefixes may be far away from the verb in the sentence. Our model, as presented in this chapter, will recognize the main verb (e.g., `legen`) and separately treat `ab-`, `an-`, etc. as unrelated particles or conjunctions. We did not include a mechanism to identify `legen ... ab` as belonging in the same lexeme as (`ablegen`, `ablegte`, ...).

The simplest way to do this would be to preprocess the data and reattach the prefix to its verb, changing `legen ... ab` to `ablegen` before inference begins. The reattachment model could be a simple binary classifier or a dependency parser; the latter will provide the desired dependency information by design.

It would also be possible to integrate a dependency parser directly into our joint probability model. This would change the generative story such that, to generate a sentence, we first pick a dependency structure, then proceed as before (see page 89): pick part-of-speech tags for the leaves, then pick lexemes and inflections and look up the corresponding word forms in the generated inflectional paradigms.

4.9 Summary

This chapter has attempted to formulate a clean framework for inducing inflectional paradigms by statistical inference. Our model is distinguished by the fact that we consider a prior distribution over entire structured paradigms and do not require inflection to be concatenative. Our work also incorporates non-parametric modeling techniques to capture

the notion of a potentially infinite vocabulary of inflectable lexemes, generalizing Goldwater et al. (2006)'s potentially infinite vocabulary of individual words. Practically, we have shown that we are able to run our sampler over a large corpus (10 million word tokens), which reduces the prediction error for morphological inflections by 10%. Especially for more frequent, irregular inflections, our corpus-based method fixes many errors, increasing absolute accuracy by up to 8%.

Chapter 5

Conclusion

This concludes our presentation of novel probabilistic models for computational morphology. In the three preceding chapters we have, step by step, presented a series of increasingly complex and powerful models suited to learn morphological properties of a natural language.

5.1 Contributions of Chapter 2

We started in Chapter 2 with the goal of developing a simple and robust framework for modeling strings or string pairs. This framework is general enough to model many string-to-string problems in natural language processing and neighboring disciplines (e.g., computational biology). Given the scope of this thesis we mainly focused on modeling tasks in morphology, like inflecting or lemmatizing verbs, but also showed that we can use the same model to reach high performance on a task like transliteration. In our framework, many types of language properties can be learned from annotated or unannotated data. We have shown:

- that string-to-string alignments are learned as a natural part of that model,
- that linguistic properties and soft constraints can be expressed and learned, which may refer to parallel character sequences in the strings to be modeled, to vowels/ consonant sequences, general language model properties of the form to be generated, sequences of identities versus substitutions, etc. Examples of such soft constraints include:
  - *prefer* `CVC`,
  - *prefer identity between input and output*,
  - *transform* `-ing` *to* `-e` or *transform* `-ing` *to* ϵ (useful in lemmatization),
  - *transform* `-eak-` *to* `-ok-` *in one conjugation class, but leave* `-eak-` *unmodified in another*,

  - *make certain vowel changes in certain regions*,
  - and tens of thousands more.

- that conjugation classes can be learned from data without annotation,
- that string (pair) regions can be learned without annotaton, determining where certain vowel change, suffix or other regions start and end.

We have carefully investigated the learned models and found that many learned properties have the form of short rewrite rules that one might find in a grammar book (see Section 2.9.2.3 on page 49 and Appendix B on Page 135.

All such information is encoded in a conditional log-linear model, in which the predicted string forms may have arbitrary length. This distinguishes our model from simple linear-chain Conditional Random Fields (CRFs), where the output variables are fixed given the input and are always enumerable. Since our approach is finite-state-based, it can generate output strings of different lengths (without having to impose artificial upper bounds) using a finite number of states.

Our model is a probability model over strings. As such, it can be used to score string hypotheses, generate k-best string output, or it can be plugged into other probability models as a component. In this thesis, we made heavy use of the latter property, as we reused it in all proability models of the following chapters.

The presented string model is surprsingly simple and performs very well. It is also completely novel (at publication time, see Dreyer et al., 2008).

5.2 Contributions of Chapter 3

In Chapter 3, we extended the scope of our morphology models. While our model from Chapter 2 was developed for string-to-string problems we were here concerned with modeling whole morphological paradigms, which consist of 20 or more strings, all of which are related to one another in some particular ways. We were interested in predicting whole morphological paradigms for a given lexeme, or filling in empty paradigm cells, given the neighboring cells—in short, building joint probability models over whole paradigms. No suitable models for these tasks existed, so we developed a new one (first published in Dreyer and Eisner, 2009).

The key idea was to bring together two known concepts: The finite-state-based models from Chapter 2 and graphical models from the machine-learning literature. Graphical models are factored probability models in which the joint probability of interrelated random variables is factored into manageable parts, which are then coordinated during inference, in order to compute marginals and make predictions. The results of this combination is a novel, finite-state-based variant of graphical models—graphical models over (multiple) strings. In these models, the random variables range over strings of arbitrary length, and the factors (potential functions) are finite-state machines.

We also showed how to adapt a known inference algorithm for graphical models—belief propagation—to work with string-based distributions and messages, locally negotiating their various predictions.

Introducing string-based variable domains and finite-state-based factors into a graphical model has far-reaching consequences for the power and expressivity of this formalism and is a conceptually new idea: Factors in this model are not simple lookup tables of weights for particular variable configurations; instead, they are functions whose values are computed via dynamic programming. This new idea is potentially useful for other problems in natural language processing: We know how to deal with many structures using dynamic programming; the new variant of graphical models presented in this chapter can be a way to make different dynamic programs interact with each other, where their predictions are coordinated using belief propagation.

Taking a step back, a simple view of the novel work in this chapter is that, in order to model multiple strings and their interactions, we built many (weighted) finite-state transducers (developed in Chapter 2), had each of them look at a different string pair, plugged them together into a big network, and coordinated them to predict all strings jointly.

In this chapter, we also presented a novel way of structure induction for string-based models, based on finding constrained graphs that minimize edit-distance between pairs of string variables.

We then showed different ways of training the induced structures from supervised data and showed experimentally that predicting cell content in morphological paradigms is usually more accurate in joint models than when predicted separately from a lemma form, without modeling interactions between neighboring forms.

In short, the novel contributions in this chapter were the development of a string-based variant of graphical models, a belief propagation algorithm that works with finite-state acceptors, or messages over strings of unbounded lengths, investigations into the power of this new formalism and a structure induction method for string-based graphical models.

5.3 Contributions of Chapter 4

In Chapter 4, we presented a novel, principled approach to learning the inflectional morphology of a language. We developed a Bayesian non-parametric model that is a joint probability distribution over ordinary linguistic notions, such as lexemes, paradigms and inflections, and the spellings of a text corpus. We have shown that we can discover unbounded numbers of new paradigms from text by clustering its words into paradigms and further into the different inflectional cells in the paradigms. Such clustering of the corpus words results in partially observed paradigms; we maintain probability distributions over inflectional forms that we do not find in the corpus.

Thanks to the methods we developed in earlier chapters, this becomes a manageable, configurable system that makes use of trainable finite-state transducers (Chapter 2) and finite-state-based graphical models over strings (Chapter 3) to model paradigms; it then

uses other known methods in this new context, like the Dirichlet process and the Chinese Restaurant process, to build and decode in a larger probability model over the unpredictably large numbers of different lexemes and paradigms of a language, as observed from a large text corpus. We ran inference on a corpus of 10 million words and achieved large reductions of error rate for predicting inflectional paradigms.

The novel contributions in this chapter are a new, principled and configurable model to learn structured inflectional morphology from text, the definition of a probability distribution over an unbounded number of lexemes that are implicit and can be discovered in text, a suitable generative story that explains the morphology of a text corpus, the use of our novel string-based message passing inference methods as a way to collapse parts of the Gibbs sampler, the combination of the Dirichlet process with finite-state graphical models over strings, a novel method to explicitly use only the relevant parts of a large corpus while integrating out others, and in summary, a novel morphological knowledge discovery method that models structured inflectional paradigms and seemlessly handles nonconcatenative morphology.

5.4 Summary

In summary, this dissertation developed an integrated, generative, component-based model for inflectional morphology that is capable of discovering inflectional paradigms from text, with only minimal supervision.

We showed that the different components of the model itself provide novel contributions to the field of natural language processing. The general string-to-string model component is a new, principled way of performing inference over strings and predicting sequences in morphology, transliteration and other natural-language tasks. The finite-state graphical models over multiple strings are a novel way of building joint probabilities over strings and predicting multiple output string jointly, and may, due to its highly general formulation that generalizes concepts from the machine learning literature, find other uses in related fields like historical linguistics and computational biology.

We showed that the overall model for inflectional morphology that we arrived at in Chapter 4 is a novel and principled approach to semi-supervised morphological learning. Not only can it be used to cluster the words of a large corpus into groups of morphologically related words, but also to further structure each of these morphological clusters, or paradigms, such that each word is assigned its particular morphological inflection, e.g., *past participle* or *third singular past*, etc. Moreover, since most of these paradigms will only be partially filled with words found in the text, the presented model maintains probability distributions over all other possible inflections in the paradigms. To illustrate, the model might find the words *shaken* and *shakes* in the text, cluster them together as related forms, or members of the same lexeme, assign the inflections *past participle* and *third singular present* to these words, and infer the exact spellings of all other inflections of *shake*, such as the spellings for *present participle*, *first singular present* and others. In addition,

the model can generate new complete paradigms for any previously unseen spelling. Our model for these tasks is probabilistic and well-formed, and we present an effective statistical inference algorithm that can be used to learn from large text copora.

Much work has been done in minimally supervised or semi-supervised discovery of morphological properties in language, but we do not know of any other work that attempts to learn and discover the hidden structured inflectional paradigms from text. We think this model is an interesting and useful contribution for several reasons:

First, we argue that it advances the state of the art in computational morphology, by introducing a plausible generative model for the inflectional morphology of a language that is based on common linguistic notions like lexemes, paradigms and inflections. We have formalized and modeled inflectional morphology in a way that explains the data as well as allowing for efficient computation, while being able to model concatenative as well as nonconcatenative morphology and irregularities.

Second, we think that the probability distributions reflecting morphological knowledge learned from unannotated text corpora can be a useful resource and may be used in even larger models in the future. In machine translation, for example, one wishes to generate text in the output language that contains the morphologically correct word forms. The same is true for any natural language generation task. Our model provides useful generalization for such tasks: Most translation systems learn direct correspondences from inflected spelling in one language to inflected spelling in the other, without generalizing to the underlying lexemes. However, the detailed morphological knowledge that our model provides can be used to reduce forms to their lexemes when learning the lexical correspondences between languages and generate the correctly inflected spellings when producing target text in a morphologically rich language.

Appendix A

Generative regularization: Ensuring proper distributions

In this appendix, we present a solution to a general problem that we encountered when training finite-state machines by maximizing their conditional log-likelihood.

The problem is that, in the case of training a globally normalized conditional likelihood model in general, just maximizing the objective function is not guaranteed to give good, even finite, output on previously unseen test data.

Suppose, during training we never observe the input symbols leading to a certain state in the finite-state model topology (e.g., in Figure A.1). If a cycle starts there, the training objective may truly have no reason to reduce the weight of that cycle C. If C cannot be reached while reading any x in training data, then the objective function does not depend at all on the weight of that cycle. The regularization term will then prefer the arcs on C to have weight 1 (for typical L_2 or L_1 regularization of $\boldsymbol{\theta}$). If the arcs on C share parameters with other arcs, then as we learn to downweight ϵ-cycles that *are* reached in training data, we may get lucky and learn parameters that downweight C sufficiently as well. However, there is no guarantee of this. For example, the ϵ-cycles in training data may be downweighted with the help of additional context features that do not happen to appear on C, and C may not be sufficiently downweighted. In any case, one cannot count on such dynamics since an adversary could design an FST in which C did not share parameters with other cycles, or in which downweighting other cycles would *upweight* C.

In general, log-linear probability models have attractive mathematical properties (convexity of (2.4)) and computational properties (dynamic programming when $\boldsymbol{f}(x, y)$ decomposes according to a decomposition of y).[1] Let us write $\acute{p}(x, y)$ for the "hybrid distribution" $\tilde{p}(x)p(y \mid x)$, where $\tilde{p}(x)$ is the empirical distribution. Choosing the log-linear model is sometimes justified theoretically by the observation (Berger, Della Pietra, and Della Pietra,

[1] More generally, we may introduce a latent variable w, such as a path in an FST, and define $p_{\boldsymbol{\theta}}(y \mid x) = \frac{1}{Z_{\boldsymbol{\theta},x}} \sum_w \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(x, y, w))$. This sacrifices convexity but preserves dynamic programming. We avoid this case for simplicity of exposition; it is not necessary if the FST is unambiguous, i.e., any (x, y) pair is accepted along only one path.

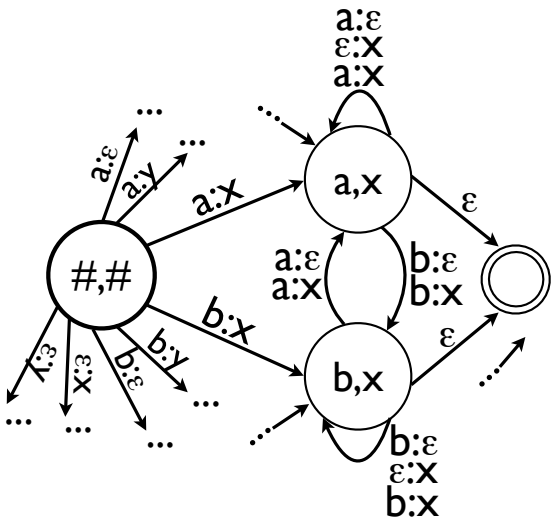

Figure A.1: This finite-state transducer represents a language model over input and output characters. Each state separately keeps track of the most recent input and output character. (We show only few states and arcs here for clarity.) When trained to maximize the conditional likelihood of training data, it is not guaranteed to give finite-length output to all test input. The state $(\#, \#)$ is the start state.

1996) that among *all* conditional distributions $p(y \mid x)$ such that $\mathbb{E}_{\acute{p}}[\boldsymbol{f}] = \mathbb{E}_{\tilde{p}}[\boldsymbol{f}]$, i.e.,

$$\sum_{x,y} \acute{p}(x,y)\boldsymbol{f}(x,y) = \sum_{x,y} \tilde{p}(x,y)\boldsymbol{f}(x,y), \tag{A.1}$$

the one whose hybrid distribution $\acute{p}$ has *maximum entropy* is the log-linear distribution p_θ with θ. However, we wish to point out that this observation is misleading:[2] many other conditional distributions have the same entropy. Indeed, the maximum entropy principle only tells us to use the above conditional distribution $p(y \mid x)$ for x that have been observed in training. For the vast number of other x (where $\tilde{p}(x) = 0$), the choice of $p(y \mid x)$ affects neither the entropy of $\acute{p}$ nor the constraints (A.1), so the constrained entropy maximization objective leaves us free to choose it as we like.[3] The solution described here is to propose a stronger principle for choosing $p(y \mid x)$.

[2]Our objection only applies to *conditional* maximum entropy training, not to the simpler case of training an unconditioned model $p(y)$ or a joint model $p(x, y)$.

[3]In the constrained entropy maximization problem, setting the derivatives of the Lagrangian to 0 determines the choice of $p(y \mid x)$ for $\tilde{p}(x) \neq 0$, but becomes a degenerate condition of the form $0 = 0$ when $\tilde{p}(x) \neq 0$.

A.0.0.1 Diagnosis

In essence, this is an overfitting problem that results from not having seen enough values of x in training data. With enough values of x, we would manage to explore all parts of the FST and learn to downweight all of the problematic cycles, ensuring that $\boldsymbol{\theta}$ was feasible.

Thus, the problem arises from training a large log-linear model with many sparse features on data that does not actually exercise all of the features. Such training setups are relatively common nowadays; one relies on L_1 or L_2 regularization for feature selection. However, the FST example shows that this common approach can lead to pathological results in the case of conditional likelihood training.

For joint likelihood training, the issue would not arise, since there would be a single shared $Z_{\boldsymbol{\theta}}$, independent of x, that is computed during training using all features and all parts of the FST. The joint likelihood formulation is

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \left(\sum_{x,y} \tilde{p}(x,y) \log p_{\boldsymbol{\theta}}(x,y) \right) \tag{A.2}$$

$$p_{\boldsymbol{\theta}}(x,y) = \frac{1}{Z_{\boldsymbol{\theta}}} \exp \boldsymbol{\theta} \cdot \boldsymbol{f}(x,y) \tag{A.3}$$

$$Z_{\boldsymbol{\theta}} = \sum_{x',y'} \exp \boldsymbol{\theta} \cdot \boldsymbol{f}(x',y') \tag{A.4}$$

The optimization here will certainly avoid $Z_{\boldsymbol{\theta}} = \infty$, since that would drive $\log p_\theta(x,y) = \log 0 = -\infty$ for any specific x, y, and hence drive the objective function (which we are trying to maximize) to $-\infty$ as well.

The reason to do conditional likelihood training is to avoid the need to design and train a real model of $p(x)$. Instead, we approximate such a model by the observed distribution $\tilde{p}(x)$ (cf. the bootstrap).

The pathological case above illustrates just how badly this strategy can go awry on sparse data. However, training of $p(y \mid x)$ may be arbitrarily bad even when $Z_{\boldsymbol{\theta},x}$ does not become infinite.

- For example, suppose that cycle C in our example above has weight that is only *close to* 1 (but still < 1). Then $p_{\boldsymbol{\theta}}(y \mid x)$ will assign a high probability to many very long strings y. The expected length $\mathbb{E}[|y|]$ under this distribution will therefore be very high, perhaps much higher than anything that is reasonable from training data. The pathological case is simply the limit where $\mathbb{E}[|y|]$ becomes infinite.

- Similar problems can arise even with finite $\mathcal{Y}$, even though $Z_{\boldsymbol{\theta},x}$ must then remain finite. Consider the simple case where $\mathcal{Y} = \{+1, -1\}$, so that Equation 2.4 is simply conditional logistic regression. Greenland, Schwartzbaum, and Finkle (2000) write: "It does not appear to be widely recognized that conditional logistic regression can suffer from bias away from the null value when it is used with too many covariates [features] or too few matched sets [training examples]."

It has been noted empirically that generative training tends to beat conditional training on *small* datasets. Perhaps this discussion sheds some light on why. However, conditional training also has advantages. In conditional training, we are not using the parameters $\boldsymbol{\theta}$ to try to match the distribution $\tilde{p}(x)$, which means that it is okay to have a simple model that does not try to model x (which may be rather complex!) and focuses only on modeling y and the relation between x and y. Training such a simple model jointly would distort the conditional distribution in order to do a better job of predicting x.

A.0.0.2 Solution as Optimization Problem

Our hybrid approach is to maximize Equation 2.4 as before, but now subject to the constraint

$$\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] \approx \mathbb{E}_{\tilde{p}}[\boldsymbol{g}], \tag{A.5}$$

that is,

$$\sum_{x,y} p_{\boldsymbol{\theta}}(x,y)\boldsymbol{g}(x,y) \approx \sum_{x,y} \tilde{p}(x,y)\boldsymbol{g}(x,y) \tag{A.6}$$

While Equation 2.4 still cares only about the conditional $p_{\boldsymbol{\theta}}(y \mid x)$ (and that only for observed values of x), the constraint pays attention to the full distribution $p_{\boldsymbol{\theta}}(x,y)$ (which is defined by Equation A.3; the resulting conditional distribution $p_{\boldsymbol{\theta}}(y \mid x)$ is given by Equation 2.4 as before).

The constraint is very similar to Equation A.1, except

- we are willing to allow an approximate match (as is done in the "soft" or "regularized" version of Equation A.1),
- we are *stipulating* that the model be log-linear with features $\boldsymbol{f}$, rather than *deriving* that from constraints on features $\boldsymbol{f}$ and a maximum-entropy criterion,
- the constraints are on features $\boldsymbol{g}$, not $\boldsymbol{f}$,
- the constraints use expectations under the full distribution $p_{\boldsymbol{\theta}}(x,y)$, not under a hybrid distribution.

Here $\boldsymbol{g}$ is a small set of features (perhaps a subset of $\boldsymbol{f}$) that is used to ensure that our conditional distribution is sane. In particular, recall that Equation 2.4 without any constraints may result in a distribution over output strings $\mathbb{E}_{p_{\boldsymbol{\theta}}(y|x)}[|y|]$ whose expected length is unreasonably large or infinite. Suppose we define $\boldsymbol{g}(x,y) \stackrel{\text{def}}{=} (|x|,|y|)$, and that $\mathbb{E}_{\tilde{p}}[\boldsymbol{g}] = (12,15)$. Then the resulting constraints ensures that the output y will have length 15 on average (like the training outputs: $\mathbb{E}_{p_{\boldsymbol{\theta}}}[|y|] \approx \mathbb{E}_{\tilde{p}}[|y|] = 15$)—when the input x is sampled from $p_{\boldsymbol{\theta}}(x)$, which has itself been constrained to have length 12 on average (like the training inputs: $\mathbb{E}_{p_{\boldsymbol{\theta}}}[|y|] \approx \mathbb{E}_{\tilde{p}}[|y|] = 12$). $p_{\boldsymbol{\theta}}(x)$ may be regarded as a smoothed version of $\tilde{p}(x)$, which has the same expected length but is positive for all $x \in \mathcal{X}$ (assuming that the parameters $\boldsymbol{\theta}$ are regularized to remain finite in the usual way).

Note that if $\boldsymbol{g}$ has dimension 0, then our method reduces to ordinary conditional likelihood training (2.4).

A.0.0.3 Objective Function

To be precise, we would like to maximize

$$\Lambda(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \sum_{x,y} \tilde{p}(x, y) \log p_{\boldsymbol{\theta}}(y \mid x) + \boldsymbol{\lambda} \cdot (\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] - \mathbb{E}_{\tilde{p}}[\boldsymbol{g}]) \tag{A.7}$$

(plus a regularizer on $\boldsymbol{\theta}$, which we omit for simplicity). Here, the interpretation of $\boldsymbol{\lambda}$ is that $\lambda_j > 0$ iff $\mathbb{E}_{p_{\boldsymbol{\theta}}}[g_j]$ is too small (compared to the empirical expectation $\mathbb{E}_{\tilde{p}}[g_j]$) and needs to be increased.

If we want to enforce $\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] = \mathbb{E}_{\tilde{p}}[\boldsymbol{g}]$ exactly, then $\boldsymbol{\lambda}$ represents a vector of Lagrange multipliers (of the same dimensionality as $\boldsymbol{g}$); these are to be optimized along with $\boldsymbol{\theta}$.[4] Note that the constraints are indeed exactly satisfiable in the case that $\boldsymbol{g}$ is a subset of $\boldsymbol{f}$, since in that case the joint maxent model would satisfy them exactly using a log-linear model with features $\boldsymbol{g}$ being those having nonzero weight.

If instead we want to impose an L_2 penalty on the difference $\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] - \mathbb{E}_{\tilde{p}}[\boldsymbol{g}]$, then we define $\boldsymbol{\lambda} = -\frac{1}{2\sigma^2}(\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] - \mathbb{E}_{\tilde{p}}[\boldsymbol{g}])$, so that $\boldsymbol{\lambda}$ is merely a function of $\boldsymbol{\theta}$. Similarly, if instead we want an L_1 penalty, then we define $\boldsymbol{\lambda} = -\mu \cdot \text{sign}(\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] - \mathbb{E}_{\tilde{p}}[\boldsymbol{g}])$.

A.0.0.4 Gradient-based Optimization

A.0.0.4.1 Expression of the gradient

It is instructive to write out an explicit representation of the gradient with respect to $\boldsymbol{\theta}$ (for fixed $\boldsymbol{\lambda}$). After some expansion of definitions, rearrangement of summations, etc., we find that

$$\nabla_{\boldsymbol{\theta}} \Lambda(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \sum_{x,y} \boldsymbol{f}(x, y) \cdot [\, \tilde{p}(x)(\tilde{p}(y \mid x) - p_{\boldsymbol{\theta}}(y \mid x)) + p_{\boldsymbol{\theta}}(x, y)(\boldsymbol{\lambda} \cdot (\boldsymbol{g}(x, y) - \mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}])) \,] \tag{A.8}$$

Ascending this gradient means making $\boldsymbol{\theta}$ more like the feature vector $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$ to the extent that

- x is in training data and y is the supervised answer rather than the currently likely answer

[4] We could also enforce the constraint approximately by using a slack variable and adding a penalty term on that slack variable.

- those $\boldsymbol{g}$ features whose expectations λ wants to increase (decrease) are respectively larger (smaller) in (x, y) than their current expectation

The first bullet corresponds to the first term within $[\ldots]$, and is standard for conditional likelihood training. The second bullet is our new addition, which pays attention to all x, not just x in training data.

A.0.0.4.2 FINITE-STATE COMPUTATION

The summations $\sum_{x,y}$ in equations (A.7) and (A.8) may be over an infinite set of string pairs. We therefore wish to compute them using dynamic programming over finite-state transducers.[5]

Assume that the feature vector function $\boldsymbol{f}$ can be computed by a weighted FST, in the sense that that each FST arc is labeled with a feature vector and $\boldsymbol{f}(x, y)$ is the sum of these feature vectors along the accepting path for (x, y).[6] Assume likewise for $\boldsymbol{g}$. (WLOG, the same FST can be used for both computations, with two distinct feature vectors labeling each arc.)

Then, for a fixed value of $\boldsymbol{\lambda}$, we can use methods from (Li and Eisner, 2009a) to compute the function $\Lambda(\boldsymbol{\theta}, \boldsymbol{\lambda})$ and its gradient $\nabla_{\boldsymbol{\theta}}\Lambda(\boldsymbol{\theta}, \boldsymbol{\lambda})$. Our computation here is derived from Equation A.7, not Equation A.8.

- First, it is necessary to compute $\log p_{\boldsymbol{\theta}}(y_k \mid x_k)$ for each training example (x_k, y_k). The numerator and denominator of $p_{\boldsymbol{\theta}}(y_k \mid x_k)$ are simply sums over the unnormalized probabilities of all FST paths that accept (x_k, y_k) or x_k, respectively. In the notation of (Li and Eisner, 2009a), these are instances of computing Z.

- The gradient of $\log p_{\boldsymbol{\theta}}(y_k \mid x_k)$ is

$$\boldsymbol{f}(x_k, y_k) - \mathbb{E}_{p_{\boldsymbol{\theta}}(y|x_k)}[\boldsymbol{f}(x_k, y)] \tag{A.9}$$

  In the notation of (Li and Eisner, 2009a), the expectation is an instance of computing $\boldsymbol{r}/Z$.[7]

- The expectation $\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}]$, meaning $\mathbb{E}_{p_{\boldsymbol{\theta}}(x,y)}[\boldsymbol{g}(x, y)]$, is likewise an instance of computing $\bar{r}/Z$. (By contrast, $\mathbb{E}_{\tilde{p}}[\boldsymbol{g}]$ is a constant that can be computed directly from training data.)

[5] The approach in this section generalizes straightforwardly to synchronous grammars of any sort, such as the hypergraphs in (Li and Eisner, 2009a).

[6] If there are multiple accepting paths, then $\boldsymbol{f}$ may be a function of the entire path; this takes us into the latent-variable situation $\boldsymbol{f}(x, y, w)$ mentioned before.

[7] If there are multiple paths accepting (x,y), as discussed in passing earlier, then $\boldsymbol{f}(x_k, y_k)$ must be replaced by an expectation over those paths, using another instance of $\bar{r}/Z$.

- Finally, Li and Eisner (2009a) also show how to compute the gradient of an expectation such as $\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}]$, using a "second-order expectation semiring." This remains quite efficient provided that either $\boldsymbol{f}$ or $\boldsymbol{g}$ has a small dimension, and in our setting we can expect $\boldsymbol{g}$ to be small.

A.0.0.5 Following the Gradient

If $\boldsymbol{\lambda}$ is defined to be a function of $\boldsymbol{\theta}$, then we must augment the gradient by $(\nabla_{\boldsymbol{\theta}}\boldsymbol{\lambda}) \cdot (\mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] - \mathbb{E}_{\tilde{p}}[\boldsymbol{g}])$. This is trivial to find using quantities already computed, when $\boldsymbol{\lambda}$ is defined as discussed above for L_1 or L_2 regularization. It is now straightforward to use any gradient-based optimizer, including a second-order optimizer.

If $\boldsymbol{\lambda}$ is taken to be a vector of Lagrange multipliers, then they must themselves be adjusted during optimization. In a first-order gradient method, they should be updated along the *negation* of the gradient, where the gradient is $\nabla_{\boldsymbol{\lambda}}\Lambda(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbb{E}_{p_{\boldsymbol{\theta}}}[\boldsymbol{g}] - \mathbb{E}_{\tilde{p}}[\boldsymbol{g}]$. In other words, to do our constrained maximization, we do gradient *ascent* on $\boldsymbol{\theta}$ and gradient *descent* on $\boldsymbol{\lambda}$ (Platt and Barr, 1988, sections 3.1–3.2).

A.0.0.6 Related Approaches

Related approaches include expectation regularization (Grandvalet and Bengio, 2004; Mann and McCallum, 2007), where the expected conditional log probability on some unlabeled data is maximized, and its generalized version, Generalized Expectation Criteria (Mann and McCallum, 2010). Key differences to this paper are that the conditional probability is used, whereas we need generative regularization, and that a certain portion of unlabeled data must be observed, whereas we generate *all* possible x in compact, smoothed form and try to match their lengths as well as the expected lengths of corresponding outputs.

Appendix B

Learned Lemmatization Rules

In this appendix, we show what our model has learned on the lemmatization task (Section 2.9.2 on page 41). We show the highest-ranked features learned in the training process for several languages.[1]

B.1 Basque Lemmatization

Rank	Feature	Feature (simplified)
1	(a,ϵ) Ex.: elgarretaratzera → elkarretaratu; elkarretaratzean → elkarretaratu	a →
2	(z,t) (e,ϵ) (ϵ,u) Ex.: puntuatzeko → puntuatu; konturatzeko → konturatu	ze → tu
3	(z,t) (e,ϵ) (n,u) Ex.: desegituratzen → desegituratu; gerturatzen → gerturatu; torturatzen → torturatu	zen → tu
4	(z,d) (e,ϵ) (ϵ,u) Ex.: ihardukitzeak → iharduki; erreproduzitzen → erreproduzitu	ze → du
5	(a,ϵ) (ϵ,a) Ex.: irabazia → irabazi; irabaziak → irabazi; baloratzeko → baloratu	a → a
6	(o,o) Ex.: ondorioztatuko → ondorioztatu; ondoriozta → ondorioztatu	o → o
7	(n,n) Ex.: mantentzen → mantendu; engainatzen → engainatu	n → n
8	(ϵ,t) (ϵ,u) (#,#) Ex.: lur → lurtu; lotuko → lotu; lortzeagatik → lortu	# → tu#

(continued on the next page)

[1] The numbers in some alignment characters denote the conjugation class or region; c2 means conjugation class 2, g5 means region 5.

Table B.1: (continued)

9	(i,i) Ex.: kontzientziatuta → kontzientziatu; aurreiritzirik → aurreiritzi	i → i
10	(ϵ,a) (a,ϵ) (ϵ,t) Ex.: plante → planteatu; sailkatze → sailkatu; sailkatuko → sailkatu	a → at
11	(n,n) (t,ϵ) (z,d) Ex.: mintzeko → mindu; mantentzea → mantendu; leuntzeko → leundu	ntz → nd
12	(z,d) (e,ϵ) (n,u) Ex.: mantentzen → mantendu; zaintzen → zaindu; apaltzen → apaldu	zen → du
13	(e,ϵ) Ex.: patentatzearekin → patentatu; bereganatzearekin → bereganatu	e →
14	(ϵ,e) (e,ϵ) Ex.: betetzearren → bete; betetzeke → bete; asebetetzeko → asebete	e → e
15	(t,ϵ) (ϵ,t) Ex.: ustelduko → usteldu; usteltzen → usteldu; babestutakoen → babestu	t → t

Table B.1: Lemmatization features for Basque

B.2 Dutch Lemmatization

Rank	Feature	Feature (simplified)
1	(ϵ,n) (#,#) Ex.: herkende → herkennen; kende → kennen; zweette → zweten	# → n#
2	(t,e) (e,ϵ) (n,n) Ex.: bedachten → bedenken; ontsnapten → ontsnappen	ten → en
3	(g,ϵ) (e,ϵ) Ex.: toegelaten → toelaten; aangetast → aantasten	ge →
4	(ϵ,l) (l,l) (d,ϵ) Ex.: gekweld → kwellen; gestild → stillen; gevuld → vullen	ld → ll
5	(i,a) (e,ϵ) (ϵ,l) Ex.: vielen → vallen; viel → vallen; bevielen → bevallen	ie → al
6	(e,e) (e,ϵ) (r,r) Ex.: getempeerd → temperen; getaxeerd → taxeren	eer → er
7	(ϵ,i) (e,j)	e → ij

(continued on the next page)

Table B.2: (continued)

	Ex.: afgeschreven → afschrijven; prezen → prijzen; grepen → grijpen	
8	(ϵ,e) (ϵ,n) (#,#) Ex.: onderhoud → onderhouden; ruim → ruimen; zuiver → zuiveren	# → en#
9	(g,g) Ex.: voorgegeven → voorgeven; doorgegaan → doorgaan	g → g
10	(ϵ,k) (k,k) (t,e) Ex.: aangeplakt → aanplakken; zakten → zakken; vertrekt → vertrekken	kt → kke
11	(t,n) (#,#) Ex.: bent → zijn; geweest → zijn; verstaat → verstaan	t# → n#
12	(d,d) Ex.: ordenend → ordenen; ordende → ordenen; orden → ordenen	d → d
13	(ϵ,f) (f,f) (ϵ,e) Ex.: geblaft → blaffen; snuffelen → snuffelen; snuffelden → snuffelen	f → ffe
14	(d,ϵ) (e,e) (ϵ,n) Ex.: verdedig → verdedigen; verdedigt → verdedigen	de → en
15	(n,n) (d,ϵ) (#,#) Ex.: aannemend → aannemen; aanslaand → aanslaan	nd# → n#

Table B.2: Lemmatization features for Dutch

B.3 English Lemmatization

Rank	Feature	Feature (simplified)
1	(ϵ,e) (i,ϵ) (n,ϵ) Ex.: freeing → free; feeding → feed; treeing → tree	in → e
2	(e,ϵ) (ϵ,e) Ex.: made → make; been → be; prepares → prepare	e → e
3	(i,ϵ) (ϵ,i) Ex.: materializes → materialize; materialize → materialize; petitioned → petition	i → i
4	(ϵ,e) (e,ϵ) (e,ϵ) Ex.: agree → agree; been → be; pioneer → pioneer	ee → e
5	(c2) (c2) (c2)	

(continued on the next page)

Table B.3: (continued)

	Ex.: seeing → see; wedding → wed; tossing → toss; founding → found; mounding → mound	
6	(i,ϵ) (e,ϵ) (d,y) Ex.: supplied → supply; unified → unify; allied → ally	ied → y
7	(ϵ,e) (e,ϵ) Ex.: exceeded → exceed; succeeded → succeed; keened → keen	e → e
8	(i,ϵ) (e,ϵ) (s,y) Ex.: fries → fry; qualifies → qualify; tidies → tidy	ies → y
9	(i,ϵ) (n,ϵ) (g,ϵ) Ex.: impinging → impinge; engendering → engender; possessing → possess	ing →
10	(ϵ,e) (e,ϵ) (s,ϵ) Ex.: presides → preside; wrestles → wrestle; incenses → incense	es → e
11	(ϵ,e) (e,ϵ) (#,#) Ex.: tyrannize → tyrannize; torture → torture; tangle → tangle	e# → e#
12	(e,ϵ) (d,ϵ) (#,#) Ex.: manipulated → manipulate; marched → march; matched → match	ed# → #
13	(ϵ,o) (e,ϵ) (w,w) Ex.: grew → grow; knew → know; threw → throw	ew → ow
14	(e,ϵ) (e,ϵ) (ϵ,e) Ex.: agree → agree; been → be; pioneer → pioneer	ee → e
15	(g5) (g5) (g5) Ex.: engendered → engender; seeing → see; engendering → engender	
18	(w,b) (e,ϵ) (r,ϵ) Ex.: were → be	wer → b
19	(c1) (c1) (c1) Ex.: would → will; been → be; tying → tie; tried → try; lying → lie	

Table B.3: Lemmatization features for English

B.4 German Lemmatization

Rank	Feature	Feature (simplified)
1	(s,n) (t,ϵ) (#,#)	st# → n#

(continued on the next page)

Table B.4: (continued)

	Ex.: drosselst → drosseln; durchsuchst → durchsuchen	
2	(t,n) (#,#) Ex.: merktet → merken; meinet → meinen; neidet → neiden	t# → n#
3	(t,t) Ex.: rettetet → retten; wettetet → wetten; bettetet → betten	t → t
4	(ϵ,e) (l,l) (e,ϵ) Ex.: radlet → radeln; radlen → radeln; nahelegend → nahelegen	le → el
5	(e,ϵ) Ex.: kettetet → ketten; kettetest → ketten; ketteten → ketten	e →
6	(ϵ,n) (#,#) Ex.: verbanne → verbannen; irrte → irren; irritierte → irritieren	# → n#
7	(t,ϵ) Ex.: kettetet → ketten; bettetet → betten; bewegtet → bewegen	t →
8	(ϵ,e) (r,r) (e,ϵ) Ex.: bereichre → bereichern	re → er
9	(e,ϵ) (e,ϵ) (ϵ,e) Ex.: entleere → entleeren; entleeret → entleeren; geleert → leeren	ee → e
10	(t,t) (e,ϵ) (t,ϵ) Ex.: verrottetet → verrotten; rettetet → retten; kettetet → ketten	tet → t
11	(n,ϵ) (g,h) Ex.: ergangen → ergehen; hinausgangen → hinausgehen	ng → h
12	(ϵ,e) (i,i) (e,ϵ) Ex.: bliebe → bleiben; erschienet → erscheinen	ie → ei
13	(ϵ,e) (t,n) (#,#) Ex.: beruft → berufen; besiegt → besiegen	t# → en#
14	(d,d) Ex.: beeindrucket → beeindrucken	d → d
15	(n,n) (d,ϵ) (#,#) Ex.: entwertend → entwerten; erbend → erben; erfordernd → erfordern	nd# → n#

Table B.4: Lemmatization features for German

B.5 Tagalog Lemmatization

Rank	Feature	Feature (simplified)
1	(a,ϵ)	a →
	Ex.: makakapagpakilala → pakilala; nakakapagpakilala → pakilala	
2	(i,ϵ)	i →
	Ex.: makapakikinig → pakinig; nakapakikinig → pakinig	
3	(s,ϵ) (a,ϵ) (s,s)	sas → s
	Ex.: magsasauli' → sauli'; naisasagot → sagot; naisasali → sali	
4	(l,l) (a,ϵ) (l,a)	lal → la
	Ex.: lalaki → laki; lalapit → lapit; magpakilala → pakilala	
5	(t,ϵ) (a,ϵ) (t,t)	tat → t
	Ex.: makatatawid → tawid; mapagtatakpan → takip	
6	(l,l) (i,ϵ) (l,i)	lil → li
	Ex.: naipanglilinis → linis; nalilimutan → limot; nalilinis → linis	
7	(g,ϵ) (a,ϵ) (g,g)	gag → g
	Ex.: gagaling → galing; pinaggagamot → gamot; pinagaganti → ganti	
8	(r,d) (a,ϵ) (n,ϵ)	ran → d
	Ex.: bayaran → bayad; pabayaran → bayad; pinapabayaran → bayad	
9	(h,ϵ) (a,ϵ) (h,h)	hah → h
	Ex.: nakahahabol → habol; naipanghahagis → hagis; naihahatid → hatid	
10	(t,ϵ) (i,ϵ) (t,t)	tit → t
	Ex.: nakapagtitigil → tigil; nagtitinda → tinda; makatitingin → tingin	
11	(p,p)	p → p
	Ex.: pinapapagipon → ipon; magpapaipon → ipon; nagpapaipon → ipon	
12	(d,d) (a,ϵ) (d,a)	dad → da
	Ex.: nakapagdadala → dala; makadadalaw → dalaw; kadadala → dala	
13	(s,ϵ) (i,ϵ) (s,s)	sis → s
	Ex.: magsisikap → sikap; mapagsisikapan → sikap; nagsisikap → sikap	
14	(s,s)	s → s
	Ex.: ipinasasalubong → pasalubong; pinasasalubungan → pasalubong	
15	(y,y) (a,ϵ) (y,a)	yay → ya
	Ex.: ipayayakap → yakap; yayakap → yakap; payayakapin → yakap	

Table B.5: Lemmatization features for Tagalog

Appendix C

List of German CELEX Forms

Name	Explanation	Examples
13PIA	1st or 3rd plural indicative past	liebten, redeten, rieben, waren, zogen
13PIE	1st or 3rd plural ind. present	lieben, reden, reiben, sind, ziehen
13PKA	1st or 3rd plural subjunctive past	liebten, redeten, rieben, waeren, zögen
13PKE	1st or 3rd plural subj. pres.	lieben, reden, reiben, seien, ziehen
13SIA	1st or 3rd singular ind. past	liebte, redete, rieb, war, zog
13SKA	1st or 3rd singular subj. past	liebte, redete, riebe, wäre, zöge
13SKE	1st or 3rd singular subj. pres.	liebe, rede, reibe, sei, ziehe
1SIE	1st singular ind. pres.	liebe, rede, reibe, bin, ziehe
2PIA	2nd plural ind. past	liebtet, redetet, riebt, wart, zogt
2PIE	2nd plural ind. pres.	liebt, redet, reibt, seid, zieht
2PKA	2nd plural subj. past	liebtet, redetet, riebet, wärt, zögt
2PKE	2nd plural subj. pres.	liebet, redet, reibet, seiet, ziehet
2SIA	2nd singular ind. past	liebtest, redetest, riebst, warst, zogst
2SIE	2nd singular ind. pres.	liebst, redest, reibst, bist, ziehst
2SKA	2nd singular subj. past	liebtest, redetest, riebest, wärst, zögst
2SKE	2nd singular subj. pres.	liebest, redest, reibest, seist, ziehest
3SIE	3rd singular ind. pres.	liebt, redet, reibt, ist, zieht
pA	past participle	geliebt, geredet, gerieben, gewesen, gezogen
pE	pres. participle	liebend, redend, reibend, seiend, ziehend
rP	imperative plural	liebt, redet, reibt, seid, zieht
rS	imperative pres.	liebe, rede, reibe, sei, ziehe

Table C.1: List of German CELEX forms with some examples (*to love, to speak, to rub, to be, to pull*).

Appendix D

Obtaining Frequency Estimates using Indirect Supervision

We are given a large text corpus of observed words $\boldsymbol{x}$. Each word x_i has a correct tag y_i. The task is to obtain a frequency estimate for each tag in the overall tagset $\mathcal{Y}_{\text{all}}$ without relying on direct supervision; the tag sequence $\boldsymbol{y}$ is not directly observed. We do, however, have a form of supervision: For each word x_i, we observe a tag set $\mathcal{Y}_i$, a small set of possible tags.

In our specific task (see Section 3.7.3.2 on page 76), the tags are lexeme-inflection pairs, e.g. *third person singular indicative of* BRECHEN. From the CELEX inflectional paradigms we know for each spelling the small set of possible lexeme-inflection pairs. For many spellings in CELEX (37.9%), there is only one possible lexeme-inflection pair; on average we have 2.0 possible pairs per spelling. In 99.5% of the spellings, all possible lexeme-inflection pairs in the set have the same lexeme. Therefore, we simplify by using only the inflection part as tag, dropping the lexeme. The model, described below, is then used to disambiguate the possible inflections for each spelling.

We pick a simple model, since the task is almost entirely supervised and we are not interested in predicting the exact tag sequence, only an estimate of how often each tag occurs. Therefore we model the observed sequence of tag sets $\boldsymbol{\mathcal{Y}}$ using the following log-linear unigram model,

$$p_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{Y}}) = \prod_{i=1}^{n} \frac{\sum_{y \in \boldsymbol{\mathcal{Y}_i}} \exp \sum_k \theta_k f_k(y)}{\sum_{y' \in \mathcal{Y}_{\text{all}}} \exp \sum_k \theta_k f_k(y')} \tag{D.1}$$

where n is the length of the sequence. We train the weight vector $\boldsymbol{\theta}$ by maximizing $\log p_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{Y}})$ minus a Gaussian regularization term, using stochastic gradient descent (SGD) due to its ease of implementation. We arbitrarily fix the variance to 10 and run for 100 iterations.[1] This can be implemented in less than 50 lines of Perl code.

[1] We also tried a variance of 5 and found it did not significantly affect the outcome.

As features we use the obvious properties of the different inflections and combinations thereof, e.g., (*third person*); (*singular*); (*third person singular*); etc. The full inflection form (e.g., *third person singular indicative*) is always a feature as well.

After training, we compute for each spelling the probability of each inflection y in its set using the log-linear feature sum $\exp \sum_k \theta_k f_k(y)$ and normalizing. For the very few spellings whose possible morphological analyses allow more than one lexeme we assume the distribution over these lexemes to be uniform.

Bibliography

A. Albright and B. Hayes. Rules vs. analogy in english past tenses: A computational/experimental study. *Cognition*, 90(2):119–161, 2003.

A.C. Albright. *The identification of bases in morphological paradigms*. PhD thesis, University of California, Los Angeles, 2002.

C. Allauzen, M. Mohri, and B. Roark. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 40–47. Association for Computational Linguistics, 2003.

C. Allauzen, M. Mohri, and B. Roark. The design principles and algorithms of a weighted grammar library. *International Journal of Foundations of Computer Science*, 16(3): 403–421, 2005.

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 International Conference on Implementation and Application of Automata*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23, 2007.

C. E. Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, 2(6):1152–1174, 1974.

R. H. Baayen, R. Piepenbrock, and L. Gulikers. The celex lexical database (release 2)[cdrom]. *Philadelphia, PA: Linguistic Data Consortium, University of Pennsylvania [Distributor]*, 1995.

M. Baroni, J. Matiasek, and H. Trost. Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, page 57, 2002.

M. Baroni, S. Bernardini, A. Ferraresi, and E. Zanchetta. The wacky wide web: A collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226, 2009.

M. J. Beal, Z. Ghahramani, and C. E. Rasmussen. The infinite hidden markov model. *Advances in Neural Information Processing Systems*, 1:577–584, 2002a.

M. J. Beal, Z. Ghahramani, and C. E. Rasmussen. The infinite hidden markov model. *Advances in Neural Information Processing Systems*, 1:577–584, 2002b.

Kenneth Beesley and Lauri Karttunen. *Finite State Morphology*. Center for the Study of Language and Information, April 2003. ISBN 1575864347.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum-entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures, 2003.

M. Bisani and H. Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.

Maximilian Bisani and Hermann Ney. Investigations on joint-multigram models for grapheme-to-phoneme conversion, 2002.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

D. Blei and P. I. Frazier. Distance dependent Chinese restaurant processes. *stat*, 1050:6, 2009.

D. Blei, T. L. Griffiths, M. I. Jordan, and J. B. Tenenbaum. Hierarchical topic models and the nested Chinese restaurant process. *Advances in neural information processing systems*, 16:106, 2004.

D. M. Blei, A. Y. Ng, and M. I Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

Alexandre Bouchard-Côté, Thomas L. Griffiths, and Dan Klein. Improved reconstruction of protolanguage word forms. In *Proceedings of HLT-NAACL*, pages 65–73, Boulder, Colorado, June 2009. Association for Computational Linguistics.

M. R. Brent, S. K. Murthy, and A. Lundberg. Discovering morphemic suffixes: A case study in minimum description length induction. In *Proceedings of the fifth international workshop on artificial intelligence and statistics*, 1995.

R. P. Brent. *Algorithms for minimization without derivatives*. Dover Pubns, 2002.

E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on applied natural language processing*, pages 152–155. Association for Computational Linguistics, 1992.

Francisco Casacuberta. Inference of finite-state transducers by using regular grammars and morphisms. In A. L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2000. 5th International Colloquium Grammatical Inference -ICGI2000-. Lisboa. Portugal. Septiembre.

Francisco Casacuberta and Colin De La Higuera. Computational complexity of problems on probabilistic grammars and transducers. In *Proc. of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Computer Science*, pages 15–24, 2000.

E. 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*, pages 69–78, 2006.

E. Chan. *Structures and distributions in morphology learning*. PhD thesis, University of Pennsylvania, 2008.

Stanley F. Chen. Conditional and joint models for grapheme-to-phoneme conversion. In *Proceedings of Interspeech*, 2003.

C. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

Alexander Clark. Learning morphology with Pair Hidden Markov Models. In *Proc. of the Student Workshop at the 39th Annual Meeting of the Association for Computational Linguistics*, pages 55–60, Toulouse, France, July 2001.

M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8. Association for Computational Linguistics, 2002.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991, 2003. ISSN 1532-4435.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

M. Creutz and K. Lagus. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, pages 21–30, 2002.

M. Creutz and K. Lagus. Unsupervised morpheme segmentation and morphology induction from text corpora using morfessor 1.0. *Computer and information science, Report A*, 81, 2005.

Fabien Cromierès and Sadao Kurohashi. An alignment algorithm using belief propagation and a structure-based distortion model. In *Proceedings of the EACL*, pages 166–174, Athens, Greece, March 2009. Association for Computational Linguistics.

Hal Daumé III. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at http://pub.hal3.name#daume04cg-bfgs, implementation available at http://hal3.name/megam/, August 2004.

C. G. De Marcken. *Unsupervised language acquisition*. PhD thesis, Massachusetts Institute of Technology, 1996.

H. Déjean. Morphemes as necessary concept for structures discovery from untagged corpora. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, pages 295–298, 1998.

Sabine Deligne, Francois Yvon, and Frédéric Bimbot. Variable-length sequence matching for phonetic transcription using joint multigrams. In *Eurospeech*, pages 2243–2246, 1995.

Vera Demberg, Helmut Schmid, and Gregor Möhler. Phonological constraints and morphological preprocessing for grapheme-to-phoneme conversion. In *Proceedings of ACL*, pages 96–103, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

A. P. Dempster, N. M. Laird, and D. 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. ISSN 0035-9246.

C. Do, S. Gross, and S. Batzoglou. Contralign: Discriminative training for protein sequence alignment. *Tenth Annual International Conference on Computational Molecular Biology (RECOMB)*, 2006.

Markus Dreyer and Jason Eisner. Better informed training of latent syntactic features. In *Proceedings of EMNLP*, pages 317–326, Sydney, Australia, July 2006. Association for Computational Linguistics.

Markus Dreyer and Jason Eisner. Graphical models over multiple strings. In *Proceedings of EMNLP*, Singapore, August 2009.

Markus Dreyer, Jason Smith, and Jason Eisner. Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of EMNLP*, Honolulu, Hawaii, October 2008.

J. Eisner. Expectation semirings: Flexible em for learning finite-state transducers. In *Proceedings of the ESSLLI workshop on finite-state methods in NLP*, 2001.

Jason Eisner. Efficient generation in primitive Optimality Theory. In *Proceedings of ACL-EACL*, pages 313–320, Madrid, July 1997.

Jason Eisner. Comprehension and compilation in Optimality Theory. In *Proceedings of ACL*, pages 56–63, Philadelphia, July 2002a.

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of ACL*, pages 1–8, Philadelphia, July 2002b.

Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, 1990.

T. S. Ferguson. A Bayesian analysis of some nonparametric problems. *The annals of statistics*, 1(2):209–230, 1973.

J. R. Finkel, A. Kleeman, and C. D Manning. Efficient, feature-based, conditional random field parsing. *Proceedings of ACL-08: HLT*, pages 959–967, 2008a.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio, June 2008b. Association for Computational Linguistics.

Dayne Freitag and Shahram Khadivi. A sequence alignment model based on the averaged perceptron. In *Proceedings of EMNLP-CoNLL*, pages 238–247, 2007.

Lucian Galescu and James F. Allen. Bi-directional conversion between graphemes and phonemes using a joint N-gram model, 2001.

J. Goldsmith. *Unsupervised learning of the morphology of a natural language*. PhD thesis, Univ. of Chicago, 1997.

J. Goldsmith. An algorithm for the unsupervised learning of morphology. *Computational Linguistics*, 27(2):153–198, 2001a.

J. Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198, 2001b.

S. Goldwater, T. Griffiths, and M. Johnson. Interpolating between types and tokens by estimating power-law generators. *Advances in neural information processing systems*, 18:459, 2006.

Sharon Goldwater. *Nonparametric Bayesian models of lexical acquisition*. PhD thesis, Brown University, 2006.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences*. The Johns Hopkins University Press, 3rd edition, October 1996. ISBN 0801854148.

David Graff and Christopher Cieri. English Gigaword. *Linguistic Data Consortium, Philadelphia*, 2003.

Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. *Advances in neural information processing systems*, 17:529–536, 2004.

P. J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711, 1995.

Sander Greenland, Judith A. Schwartzbaum, and William D. Finkle. Problems due to small samples and sparse data in conditional logistic regression analysis. *American Journal of Epidemiology*, 151(5):531–539, 2000.

Asela Gunawardana, Milind Mahajan, Alex Acero, and John C. Platt. Hidden conditional random fields for phone classification. In *Proceedings of Interspeech*, 2005.

M. A. Hafer and S. F. Weiss. Word segmentation by letter successor varieties. *Information Storage and Retrieval*, 1974.

G. Haffari and Y. W Teh. Hierarchical dirichlet trees for information retrieval. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 173–181, 2009.

Z. S. Harris. From phoneme to morpheme. *Language*, 31(2):190–222, 1955.

Bruce Hayes and Colin Wilson. A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, 39(3):379–440, 2008.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

G. Hong, M. J. Kim, D. G. Lee, and H. C Rim. A hybrid approach to english-korean name transliteration. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 108–111, 2009.

S. Jain and R. M. Neal. A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13(1): 158–182, 2004.

Klara Janecki. *300 Polish Verbs*. Barron's Educational Series, 2000.

M. Jansche and R. Sproat. Named entity transcription with pair n-gram models. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 32–35, 2009.

F. Jelinek. *Statistical methods for speech recognition*. MIT Press, 1997. ISBN 0262100665.

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. Applying many-to-many alignments and hidden markov models to letter-to-phoneme conversion. In *Proceedings of NAACL-HLT*, pages 372–379, Rochester, New York, April 2007. Association for Computational Linguistics.

M. Johnson, S. Goldwater, and T. Griffiths. Unsupervised word segmentation for sesotho using adaptor grammars. In *SIGMORPHON 2008 Tenth Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, page 20, 2008.

Bart Jongejan and Hercules Dalianis. Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 145–153, Suntec, Singapore, August 2009. Association for Computational Linguistics.

M. I Jordan. *Learning in graphical models*. Kluwer Academic Publishers, 1998. ISBN 0792350170.

D. Jurafsky, J. H. Martin, A. Kehler, K. Vander Linden, and N. Ward. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, volume 163. MIT Press, 2000.

R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378, 1994.

L. Karttunen and K. R. Beesley. Twenty-five years of finite-state morphology. *Inquiries Into Words, a Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 71–83, 2005.

L. Karttunen, R. M. Kaplan, and A. Zaenen. Two-level morphology with composition. In *Proceedings of the 14th conference on Computational linguistics-Volume 1*, pages 141–148, 1992.

M. Kay. Nonconcatenative finite-state morphology. In *Proceedings of the third conference on European chapter of the Association for Computational Linguistics*, pages 2–10, 1987.

M. Kay and R. Kaplan. Phonological rules and finite-state transducers. In *Unpublished conference paper*, 1981.

André Kempe, Jean-Marc Champarnaud, and Jason Eisner. A note on join and auto-intersection of n-ary rational relations. In Loek Cleophas and Bruce Watson, editors, *Proceedings of the Eindhoven FASTAR Days (Computer Science Technical Report 04-40)*. Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Netherlands, 2004.

K. Knight and Y. Al-Onaizan. Translation with finite-state devices. *Machine Translation and the Information Soup*, pages 421–437, 1998.

Kevin Knight and Jonathan Graehl. Machine transliteration. In *Proceedings of ACL*, pages 128–135, 1997.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL, Companion Volume*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

K. Koskenniemi. A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, pages 178–181, 1984.

J. B. Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. ISSN 0003-4851.

M. Kurimo, S. Virpioja, V. Turunen, and K. Lagus. Morpho challenge competition 2005-2010: Evaluations and results. *ACL 2010*, page 87, 2010.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 282–289, 2001a.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001b.

Thomas Lavergne, Olivier Cappé, and François Yvon. Practical very large scale crfs. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 504–513. Association for Computational Linguistics, July 2010.

S. I. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of Markov networks using L1 regularization. In *Proceedings of NIPS*, 2006.

L. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

Haizhou Li, A. Kumaran, Min Zhang, and Vladimir Pervouchine. Whitepaper of news 2009 machine transliteration shared task. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pages 19–26. Association for Computational Linguistics, Aug 2009a.

Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 40–51, Singapore, August 2009a.

Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of EMNLP*, 2009b.

Zhifei Li, Jason Eisner, and Sanjeev Khudanpur. Variational decoding for statistical machine translation. In *Proceedings of ACL*, 2009b.

P. Liang, M. I. Jordan, and D. Klein. Type-based mcmc. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 573–581, 2010.

Percy Liang, Slav Petrov, Michael I Jordan, and Dan Klein. The infinite pcfg using hierarchical dirichlet processes. *IN EMNLP '07*, pages 688–697, 2007.

J. Lin. Divergence measures based on the Shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991. ISSN 0018-9448.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989a.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528, 1989b.

G. S. Mann and A. McCallum. Simple, robust, scalable semi-supervised learning via expectation regularization. In *Proceedings of the 24th international conference on Machine learning*, 2007.

Gideon Mann and Andrew McCallum. Generalized expectation criteria for Semi-Supervised learning with weakly labeled data. *Journal of Machine Learning Research*, 2010.

J. B. Mariño, R. Banchs, J. M. Crego, A. de Gispert, P. Lambert, JA Fonollosa, and M. Ruiz. Bilingual n-gram statistical machine translation. *Proc. of Machine Translation Summit X*, pages 275–82, 2005.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *Proc. of ACL*, pages 75–82, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

P. H. Matthews. *Inflectional morphology: A theoretical study based on aspects of Latin verb conjugation*. Cambridge University Press, 1972.

J. D. McAuliffe, D. M. Blei, and M. I Jordan. Nonparametric empirical bayes for the dirichlet process mixture model. *Statistics and Computing*, 16(1):5–14, 2006.

Andrew McCallum, Kedar Bellare, and Fernando Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In *Proceedings of UAI*, 2005.

J. McCarthy. A prosodic Theory of Nonconcatenative Morphology. *Linguistic Inquiry*, 12: 373–418, 1981.

J. J. McCarthy. Sympathy and phonological opacity. *Phonology*, 16(03):331–399, 2002.

D. Mimno and A. McCallum. Topic models conditioned on arbitrary features with dirichlet-multinomial regression. In *Proc. of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

M. Mohri. Weighted automata algorithms. *Handbook of weighted automata*, pages 213–254, 2009.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2), 1997.

C. Monson, A. Lavie, J. Carbonell, and L. Levin. Unsupervised induction of natural language morphology inflection classes. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, pages 52–61, 2004.

Christian Monson, Jaime Carbonell, Alon Lavie, and Lori Levin. ParaMor: minimally supervised induction of paradigm structure and morphological analysis. In *Proceedings of Ninth Meeting of the ACL Special Interest Group in Computational Morphology and Phonology*, pages 117–125, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

J. Naradowsky and S. Goldwater. Improving morphology induction by learning spelling rules. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*, pages 1531–1536, 2009.

R. M. Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

J. Peng, L. Bo, and J. Xu. Conditional neural fields. *Advances in Neural Information Processing Systems*, 22:1419–1427, 2009.

Fernando Pereira, Michael Riley, and Mehryar Mohri. Weighted finite-state transducers in speech recognition. *Computer speech and Language*, 16:69–88, 2002.

Fernando C. N. Pereira and Michael Riley. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, Cambridge, MA, 1997.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356, 2003.

Slav Petrov and Dan Klein. Discriminative log-linear grammars with latent variables. In J. C. Platt, D. Koller, Y. Singer, and S.Editors Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1153–1160. MIT Press, 2008.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics, July 2006.

Slav Petrov, Adam Pauls, and Dan Klein. Learning structured models for phone recognition. In *Proceedings of EMNLP-CoNLL*, pages 897–905, 2007.

S. Pinker and A. Prince. On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Connections and symbols*, pages 73–193, 1988.

John C. Platt and Alan H. Barr. Constrained differential optimization for neural networks. Technical Report Caltech-CS-TR-88-17, California Institute of Technology, 1988.

Hoifung Poon, Colin Cherry, and Kristina Toutanova. Unsupervised morphological segmentation with Log-Linear models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 209–217, Boulder, Colorado, June 2009. Association for Computational Linguistics.

Matt Post and Daniel Gildea. Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48. Association for Computational Linguistics, Aug 2009.

Alan Prince and Paul Smolensky. Optimality theory: Constraint interaction in generative grammar. Technical Report RuCCS-TR-2. ROA-537., Rutgers University, 1993.

A. Quattoni, S. Wang, L. P. Morency, M. Collins, and T. Darrell. Hidden conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29 (10):1848–1852, 2007.

L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSp Magazine*, 3(1):4–16, 1986.

C. E. Rasmussen. The infinite Gaussian mixture model. *Advances in neural information processing systems*, 12:554–560, 2000.

Martin Riedmiller and Heinrich Braun. RPROP – A Fast Adaptive Learning Algorithm. *PROC. OF ISCIS VII), UNIVERSITAT*, 1992.

J. Rissanen. *Stochastic complexity in statistical inquiry theory*. World Scientific Publishing Co., Inc. River Edge, NJ USA, 1989.

Eric Sven Ristad and Peter N. Yianilos. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998.

B. Roark, M. Saraclar, M. Collins, and M. Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 47, 2004a.

Brian Roark and Richard Sproat. *Computational approaches to morphology and syntax*. Oxford University Press, 2007.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 47–54, Barcelona, Spain, July 2004b. doi: 10.3115/1218955.1218962.

Frank Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Spartan Books, Washington, 1962.

D. E. Rumelhart and J. L. McClelland. On learning the past tense of english verbs. 2: 216–271, 1986.

P. Schone and D. Jurafsky. Knowledge-free induction of inflectional morphologies. In *Proceedings of the North American Chapter of the Association of Computational Linguistics*, volume 183, page 191, 2001.

Marcel-Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4: 185–196, 1961.

J. Sethuraman. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4(2):639–650, 1994.

Tarek Sherif and Grzegorz Kondrak. Substring-based transliteration. In *Proceedings of ACL*, pages 944–951, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

David Smith and Jason Eisner. Dependency parsing by belief propagation. In *Proceedings of EMNLP*, 2008.

Noah A Smith, David A Smith, and Roy W Tromble. Context-Based morphological disambiguation with random fields. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 475–482, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

M. G. Snover and M. R. Brent. A bayesian model for morpheme and paradigm identification. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, volume 39, pages 482–490, 2001.

B. Snyder, T. Naseem, J. Eisenstein, and R. Barzilay. Unsupervised multilingual learning for POS tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1041–1050, 2008.

B. Snyder, R. Barzilay, and K. Knight. A statistical model for lost language decipherment. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1048–1057, 2010.

Benjamin Snyder and Regina Barzilay. Unsupervised multilingual learning for morphological segmentation. In *Proceedings of ACL-08: HLT*, pages 737–745. Association for Computational Linguistics, June 2008.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of AISTATS*, 2011 (to appear). URL http://cs.jhu.edu/~jason/papers/#aistats11.

G. T. Stump. *Inflectional morphology: A theory of paradigm structure*. Cambridge University Press, 2001. ch.1 is a good overview over morph. theories.

Erik B. Sudderth, Alexander T. Ihler, Er T. Ihler, William T. Freeman, and Alan S. Willsky. Nonparametric belief propagation. In *Proc. of CVPR*, pages 605–612, 2002.

Charles Sutton and Andrew McCallum. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields*, 2004.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of ICML*, 2004.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Y. W. Teh. Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer, 2010.

Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

Bruce Tesar and Paul Smolensky. *Learnability in Optimality Theory*. MIT Press, May 2000. ISBN 0262201267.

K. Toutanova and R. C. Moore. Pronunciation modeling for improved spelling correction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, page 151. Association for Computational Linguistics, 2002.

Kristina Toutanova, Hisami Suzuki, and Achim Ruopp. Applying morphology generation models to machine translation. In *Proceedings of ACL-08: HLT*, pages 514–522, Columbus, Ohio, June 2008. Association for Computational Linguistics.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, page 477–485. Association for Computational Linguistics, Aug 2009.

H. M. Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984, 2006.

Richard Wicentowski. *Modeling and Learning Multilingual Inflectional Morphology in a Minimally Supervised Framework*. PhD thesis, Johns-Hopkins University, 2002.

Colin Wilson and Bruce Hayes. A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, 39:379–440, 2008.

Jun Wu and Sanjeev Khudanpur. Efficient training methods for maximum entropy language modeling. In *Proceedings of ICSLP*, volume 3, pages 114–117, Beijing, October 2000.

Elif Yamangil and Stuart M Shieber. Bayesian synchronous tree-substitution grammar induction and its application to sentence compression. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 937–947. Association for Computational Linguistics, July 2010.

D. Yarowsky and R. Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In K. Vijay-Shanker and Chang-Ning Huang, editors, *Proceedings of ACL*, pages 207–216, Hong Kong, October 2000a.

David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995.

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, Hong Kong, October 2000b. Association for Computational Linguistics. doi: 10.3115/1075218.1075245.

Bing Zhao, Nguyen Bach, Ian Lane, and Stephan Vogel. A log-linear block transliteration model based on bi-stream HMMs. In *Proceedings of NAACL-HLT*, pages 364–371, Rochester, New York, April 2007. Association for Computational Linguistics.

Kie Zuraw. The Role of Phonetic Knowledge in Phonological Patterning: Corpus and Survey Evidence from Tagalog Infixation. *Language*, 83(2):277–316, 2007.

Vita

Markus Dreyer received the M.A. degree in Computational Linguistics, German Linguistics and Literature, and Philosophy from Heidelberg University, Germany, in 2002. During his time in Germany, he has also worked in the I.B.M. European Speech Research Group and at SAP. He enrolled in the Computer Science Ph.D. program at Johns Hopkins University in 2003 and has been a member of the Center for Language and Speech Processing (CLSP) and the Human Language Technology Center of Excellence (HLT-COE). He was awarded a Wolman Fellowship and an NSF research award.

In 2010, he defended this dissertation and moved to Los Angeles, California. There he started as a Research Scientist at SDL Language Weaver, working on statistical machine translation.