General Methods for Fine-Grained Morphological and Syntactic Disambiguation

Thomas Müller

München 2015

General Methods for Fine-Grained Morphological and Syntactic Disambiguation

Thomas Müller

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Thomas Müller
aus Erfurt

München, 11. März 2015

Erstgutachter: Prof. Hinrich Schütze
Zweitgutachter: Prof. Jan Hajič
Tag der mündlichen Prüfung: 4. Mai 2015

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Müller, Thomas

Name, Vorname

München, 11. März 2015

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2

Danksagung

An meine Frau Elena, die sich in den letzten zwei Jahren mehr für die Familie aufopfern musste, als es im 21. Jahrhundert der Fall sein sollte und an meine Eltern Jutta und Alfred, die immer im richtigen Maße nachsehend und fordernd waren.

Contents

List of Figures

List of Tables

List of Algorithms

Prepublications

Some of the chapters of this thesis contain material that has been published at international conferences.

Chapters

- **A Morphological Language Model**
  This chapter covers work already published at international peer-reviewed conferences. The relevant publications are Müller and Schütze (2011) and Müller et al. (2012). The research described in this chapter was carried out in its entirety by myself. The other authors of the publications acted as advisors or were responsible for work that was reported in the publication, but is not included in this chapter.

- **HMM-LA Training for Dependency Parsing**
  This chapter covers work already published at an international peer-reviewed conference. The relevant publication is Müller et al. (2014). The research described in this chapter was carried out in its entirety by myself. The other authors of the publication acted as advisors or were responsible for work that was reported in the publication, but is not included in this chapter.

- **Morphological Tagging with Higher-Order CRFs**
  This chapter covers work already published at international peer-reviewed conferences. The most relevant publication is Müller et al. (2013). The chapter also covers a small part of Björkelund et al. (2013). The research described in this chapter was carried out in its entirety by myself. The other authors of the publications acted as advisors or were responsible for work that was reported in the publication, but is not included in this chapter.

- **Morphological Tagging with Word Representations**
  This chapter covers work already published at international peer-reviewed conferences. The most relevant publication is Müller and Schütze (2015)[1]. The research described in this chapter was carried out in its entirety by myself. The other authors of the publication acted as advisors or were responsible for work that was reported in the publication, but is not included in this chapter.

[1]This paper has been accepted at NAACL 2015 (http://naacl.org/naacl-hlt-2015/papers.html) but has not been published at the time of the submission of this thesis

Publications

- Müller and Schütze (2015). *Robust morphological tagging with word representations*. Thomas Müller and Hinrich Schütze. In Proceedings of NAACL [1]
- Müller et al. (2014). *Dependency Parsing with Latent Refinements of Part-of-Speech Tags*. Thomas Müller, Richard Farkas, Alex Judea, Helmut Schmied and Hinrich Schütze. In Proceedings of EMNLP (short paper)
- Müller et al. (2013). *Efficient Higher-Order CRFs for Morphological Tagging*. Thomas Müller, Helmut Schmied and Hinrich Schütze. In Proceedings of EMNLP
- Björkelund et al. (2013). *(Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task*. Anders Björkelund, Özlem Çetinoğlu, Richárd Farkas, Thomas Müller and Wolfgang Seeker. SPMRL
- Müller et al. (2012). *A Comparative Investigation of Morphological Language Modeling for the Languages of the EU*. Thomas Müller, Hinrich Schütze and Helmut Schmied. In Proceedings of NAACL
- Müller and Schütze (2011). *Improved Modeling of Out-Of-Vocabulary Words Using Morphological Classes*. Thomas Müller and Hinrich Schütze. In Proceedings of ACL (short paper)

Abstract

We present methods for improved handling of morphologically rich languages (MRLs) where we define MRLs as languages that are morphologically more complex than English. Standard algorithms for language modeling, tagging and parsing have problems with the productive nature of such languages. Consider for example the possible forms of a typical English verb like *work* that generally has four four different forms: *work*, *works*, *working* and *worked*. Its Spanish counterpart *trabajar* has 6 different forms in present tense: *trabajo*, *trabajas*, *trabaja*, *trabajamos*, *trabajáis* and *trabajan* and more than 50 different forms when including the different tenses, moods (indicative, subjunctive and imperative) and participles. Such a high number of forms leads to sparsity issues: In a recent Wikipedia dump of more than 400 million tokens we find that 20 of these forms occur only twice or less and that 10 forms do not occur at all. This means that even if we only need unlabeled data to estimate a model and even when looking at a relatively common and frequent verb, we do not have enough data to make reasonable estimates for some of its forms. However, if we decompose an unseen form such as *trabajaréis* 'you will work', we find that it is *trabajar* in future tense and second person plural. This allows us to make the predictions that are needed to decide on the grammaticality (language modeling) or syntax (tagging and parsing) of a sentence.

In the first part of this thesis, we develop a **morphological language model**. A language model estimates the grammaticality and coherence of a sentence. Most language models used today are word-based n-gram models, which means that they estimate the transitional probability of a word following a history, the sequence of the $(n-1)$ preceding words. The probabilities are estimated from the frequencies of the history and the history followed by the target word in a huge text corpus. If either of the sequences is unseen, the length of the history has to be reduced. This leads to a less accurate estimate as less context is taken into account.

Our morphological language model estimates an additional probability from the morphological classes of the words. These classes are built automatically by extracting morphological features from the word forms. To this end, we use unsupervised segmentation algorithms to find the suffixes of word forms. Such an algorithm might for example segment *trabajaréis* into *trabaja* and *réis* and we can then estimate the properties of *trabajaréis* from other word forms with the same or similar morphological properties. The data-driven nature of the segmentation algorithms allows them to not only find inflectional suffixes (such as *-réis*), but also more derivational phenomena such as the head nouns of compounds or even endings such as *-tec*, which identify technology oriented companies such as *Vortec*, *Memotec* and *Portec* and would not be regarded as a morphological suffix by traditional linguistics. Additionally, we extract shape features such

as if a form contains digits or capital characters. This is important because many rare or unseen forms are proper names or numbers and often do not have meaningful suffixes. Our class-based morphological model is then interpolated with a word-based model to combine the generalization capabilities of the first and the high accuracy in case of sufficient data of the second.

We evaluate our model across 21 European languages and find improvements between 3% and 11% in perplexity, a standard language modeling evaluation measure. Improvements are highest for languages with more productive and complex morphology such as Finnish and Estonian, but also visible for languages with a relatively simple morphology such as English and Dutch. We conclude that a morphological component yields consistent improvements for all the tested languages and argue that it should be part of every language model.

Dependency trees represent the syntactic structure of a sentence by attaching each word to its syntactic head, the word it is directly modifying. **Dependency parsing** is usually tackled using heavily lexicalized (word-based) models and a thorough morphological preprocessing is important for optimal performance, especially for MRLs. We investigate if the lack of morphological features can be compensated by features induced using **hidden Markov models with latent annotations** (HMM-LAs) (Huang et al., 2009) and find this to be the case for German. HMM-LAs were proposed as a method to increase part-of-speech tagging accuracy. The model splits the observed part-of-speech tags (such as verb and noun) into subtags. An expectation maximization algorithm is then used to fit the subtags to different roles. A verb tag for example might be split into an auxiliary verb and a full verb subtag. Such a split is usually beneficial because these two verb classes have different contexts. That is, a full verb might follow an auxiliary verb, but usually not another full verb.

For German and English, we find that our model leads to consistent improvements over a parser (Bohnet, 2010) not using subtag features. Looking at the labeled attachment score (LAS), the number of words correctly attached to their head, we observe an improvement from 90.34 to 90.75 for English and from 87.92 to 88.24 for German. For German, we additionally find that our model achieves almost the same performance (88.24) as a model using tags annotated by a supervised morphological tagger (LAS of 88.35). We also find that the German latent tags correlate with morphology. Articles for example are split by their grammatical case.

We also investigate the part-of-speech tagging accuracies of models using the traditional treebank tagset and models using induced tagsets of the same size and find that the latter outperform the former, but are in turn outperformed by a discriminative tagger.

Furthermore, we present a method for fast and accurate **morphological tagging**. While part-of-speech tagging annotates tokens in context with their respective word categories, morphological tagging produces a complete annotation containing all the relevant inflectional features such as case, gender and tense. A complete reading is represented as a single tag. As a reading might consist of several morphological features the resulting tagset usually contains hundreds or even thousands of tags. This is an issue for many decoding algorithms such as Viterbi which have runtimes depending quadratically on the number of tags. In the case of morphological tagging, the problem can be avoided by using a morphological analyzer. A morphological analyzer is a manually created finite-state transducer that produces the possible morphological readings of a word form. This analyzer can be used to prune the tagging lattice and to allow for the application of standard sequence labeling algorithms. The downside of this approach is that such an

analyzer is not available for every language or might not have the coverage required for the task. Additionally, the output tags of some analyzers are not compatible with the annotations of the treebanks, which might require some manual mapping of the different annotations or even to reduce the complexity of the annotation.

To avoid this problem we propose to use the posterior probabilities of a conditional random field (CRF) (Lafferty et al., 2001) lattice to prune the space of possible taggings. At the zero-order level the posterior probabilities of a token can be calculated independently from the other tokens of a sentence. The necessary computations can thus be performed in linear time. The features available to the model at this time are similar to the features used by a morphological analyzer (essentially the word form and features based on it), but also include the immediate lexical context. As the ambiguity of word types varies substantially, we just fix the average number of readings after pruning by dynamically estimating a probability threshold. Once we obtain the pruned lattice, we can add tag transitions and convert it into a first-order lattice. The quadratic forward-backward computations are now executed on the remaining plausible readings and thus efficient. We can now continue pruning and extending the lattice order at a relatively low additional runtime cost (depending on the pruning thresholds). The training of the model can be implemented efficiently by applying stochastic gradient descent (SGD). The CRF gradient can be calculated from a lattice of any order as long as the correct reading is still in the lattice. During training, we thus run the lattice pruning until we either reach the maximal order or until the correct reading is pruned. If the reading is pruned we perform the gradient update with the highest order lattice still containing the reading. This approach is similar to early updating in the structured perceptron literature and forces the model to learn how to keep the correct readings in the lower order lattices. In practice, we observe a high number of lower updates during the first training epoch and almost exclusively higher order updates during later epochs.

We evaluate our CRF tagger on six languages with different morphological properties. We find that for languages with a high word form ambiguity such as German, the pruning results in a moderate drop in tagging accuracy while for languages with less ambiguity such as Spanish and Hungarian the loss due to pruning is negligible. However, our pruning strategy allows us to train higher order models (order > 1), which give substantial improvements for all languages and also outperform unpruned first-order models. That is, the model might lose some of the correct readings during pruning, but is also able to solve more of the harder cases that require more context. We also find our model to substantially and significantly outperform a number of frequently used taggers such as Morfette (Chrupala et al., 2008) and SVMTool (Giménez and Màrquez, 2004).

Based on our morphological tagger we develop a simple method to increase the performance of a state-of-the-art constituency parser (Petrov et al., 2006). A constituency tree describes the syntactic properties of a sentence by assigning spans of text to a hierarchical bracket structure. Petrov et al. (2006) developed a language-independent approach for the automatic annotation of accurate and compact grammars. Their implementation – known as the Berkeley parser – gives state-of-the-art results for many languages such as English and German. For some MRLs such as Basque and Korean, however, the parser gives unsatisfactory results because of its simple unknown word model. This model maps unknown words to a small number of signatures (similar to our morphological classes). These signatures do not seem expressive enough for many of the

subtle distinctions made during parsing. We propose to replace rare words by the morphological reading generated by our tagger instead. The motivation is twofold. First, our tagger has access to a number of lexical and sublexical features not available during parsing. Second, we expect the morphological readings to contain most of the information required to make the correct parsing decision even though we know that things such as the correct attachment of prepositional phrases might require some notion of lexical semantics.

In experiments on the SPMRL 2013 (Seddah et al., 2013) dataset of nine MRLs we find our method to give improvements for all languages except French for which we observe a minor drop in the PARSEVAL score of 0.06. For Hebrew, Hungarian and Basque we find substantial absolute improvements of 5.65, 11.87 and 15.16, respectively.

We also performed an extensive evaluation on the utility of word representations for morphological tagging. Our goal was to reduce the drop in performance that is caused when a model trained on a specific domain is applied to some other domain. This problem is usually addressed by domain adaption (DA). DA adapts a model towards a specific domain using a small amount of labeled or a huge amount of unlabeled data from that domain. However, this procedure requires us to train a model for every target domain. Instead we are trying to build a robust system that is trained on domain-specific labeled and domain-independent or general unlabeled data. We believe word representations to be key in the development of such models because they allow us to leverage unlabeled data efficiently. We compare data-driven representations to manually created morphological analyzers. We understand data-driven representations as models that cluster word forms or map them to a vectorial representation. Examples heavily used in the literature include Brown clusters (Brown et al., 1992a), Singular Value Decompositions of count vectors (Schütze, 1995) and neural-network-based embeddings (Collobert et al., 2011). We create a test suite of six languages consisting of in-domain and out-of-domain test sets. To this end we converted annotations for Spanish and Czech and annotated the German part of the Smultron (Volk et al., 2010) treebank with a morphological layer. In our experiments on these data sets we find Brown clusters to outperform the other data-driven representations. Regarding the comparison with morphological analyzers, we find Brown clusters to give slightly better performance in part-of-speech tagging, but to be substantially outperformed in morphological tagging.

Zusammenfassung

Wir stellen Methoden zur verbesserten Verarbeitung von morphologisch reichen Sprachen (MRL) vor, wobei wir MRLs als Sprachen definieren, deren Morphologie komplexer als die Englische Morphologie ist. Betrachten wir z.B. die typischen Formen eines Englischen Verbs so wie *work* 'arbeiten', welches in vier verschiedenen Formen auftreten kann: *work*, *works*, *working* und *worked*. Sein spanisches Gegenstück *trabajar* 'arbeiten' hat sechs verschiedene einfache Präsensformen: *trabajo*, *trabajas*, *trabaja*, *trabajamos*, *trabajáis* und *trabajan* und mehr als 50 verschiedene Formen wenn wir auch die anderen Zeitformen, Modi (Indikativ, Subjunktiv und Imperativ) und Partizipien berücksichtigen. Solch eine hohe Anzahl an Formen führt dazu, dass viele Formen sehr selten sind. So kommen in einem aktuellen Wikipedia Auszug von über 400 Millionen Wörtern 20 Formen weniger als zweimal vor und 10 Formen überhaupt nicht. Das bedeutet, dass selbst wenn wir nur unannotierte Daten benötigen um unser Modell zu schätzen und selbst unter Betrachtung eines relativ gewöhnlichen und häufigen Verbs haben wir nicht genug Daten um die Eigenschaften einiger Formen dieses Verbs zuverlässig zu schätzen. Würden wir aber eine Form wie *trabajaréis* 'ihr werdet arbeiten' zerlegen dann würden wir sehen, dass es sich um das Verb *trabajar* in der zweiten Person Plural Futur handelt. Diese Information würde uns dann erlauben über die Grammatikalität (Sprachmodellierung) oder Syntax (Tagging und Parsing) eines Satzes zu urteilen.

Im ersten Teil dieser Dissertation, entwickeln wir ein morphologisches Sprachmodell. Ein Sprachmodell schätzt die Grammatikalität und Kohärenz eines Satzes. Die meisten heutzutage verwendeten Sprachmodelle sind wortbasierte N-Gramm Modelle. Das bedeutet, dass sie die Übergangswahrscheinlichkeit eines Wortes nach einer *History* 'Verlauf', d.h. den $(n-1)$ vorhergehenden Wörtern beurteilen. Die Wahrscheinlichkeiten werden aus der Häufigkeit der History und der Häufigkeit von History und Zielwort in einem riesigen Textkorpus ermittelt. Sollte eine der beiden Wortfolgen nicht vorkommen, dann muss die Länge der History verkürzt werden. Dies führt zu einer ungenaueren Schätzung, da weniger Kontext berücksichtigt wird.

Unser morphologisches Sprachmodell schätzt eine zusätzliche Wahrscheinlichkeit über die morphologische Klasse der Wörter. Diese Klassen werden automatisch erzeugt in dem die morphologischen Merkmale der Wortformen extrahiert werden. Dazu benutzen wir unüberwachte Segmentierungsalgorithmen um die Suffixe der Formen zu ermitteln. Solch ein Algorithmus könnte z.B. das Wort *trabajaréis* in *trabaja* und *réis* unterteilen und wir könnten die Eigenschaften von *trabajaréis* aus anderen Wörtern mit den gleichen morphologischen Eigenschaften ableiten. Die datengetriebene Natur der Segmentierungsalgorithmen bringt sie dazu nicht nur flektive Suffixe (so wie *-réir*) zu finden, sondern auch eher derivative Phänomene wie Hauptwörter

von Komposita oder sogar Endungen wie *-tec*, welche technologieorientierte Firmen wie *Vortec*, *Memotec* und *Portec* identifizieren aber nach traditionellen linguistischen Kriterien nicht als Suffix zählen würden. Zusätzlich extrahieren wir Merkmale der Gestalt oder Schreibweise wie z.B. ob eine Form eine Ziffer oder einen Großbuchstaben enthält. Diese Merkmale sind wichtig, da viele seltene oder ungesehene Formen Eigennamen oder Zahlen sind die oftmals keine bedeutungstragenden Suffixe haben. Unser klassenbasiertes morphologisches Modell wird dann mit einem wortbasierten Modell interpoliert um die Verallgemeinerungsfähigkeiten des ersten mit der hohen Genauigkeit für häufige Wortfolgen des zweiten kombinieren zu können.

Wir evaluieren unser Modell über 21 europäische Sprachen hinweg und zeigen Perplexitäts-Verbesserungen zwischen 3% und 11%, wobei die Perplexität ein Standardevaluierungsmaß der Sprachmodellierung ist. Die Verbesserungen sind am höchsten für Sprachen mit eher produktiver und komplexer Morphologie wie z.B. Finnisch und Estländisch, können aber auch für Sprachen mit relativ einfacher Morphologie so wie Englisch und Niederländisch beobachtet werden. Zusammenfassend können wir sagen, dass eine morphologische Komponente konsistente Verbesserungen für alle Sprachen bringt und wir folgern daraus, dass solch eine Komponente in jedes Sprachmodell integriert werden sollte.

Dependenzbäume repräsentieren die syntaktische Struktur eines Satze indem jedes Wort an seinen syntaktischen Kopf, d.h. das Wort das es direkt modifiziert, angehängt wird. Das Dependenzparsen (d.h. das automatische Erzeugen von Dependenzbäumen) wird üblicherweise mit stark lexikalisierten (wortbasierten) Modellen in Angriff genommen. Eine gründliche morphologische Vorverarbeitung ist dabei wichtig um eine optimale Genauigkeit zu erreichen, insbesondere im Fall von MRLs. Wir untersuchen ob das Fehlen von morphologischen Merkmalen durch automatisch erzeugte Merkmale kompensiert werden kann. Dazu verwenden wir Hidden Markov Modelle mit latenten Annotationen (HMM-LA), d.h. Hidden Markov Modelle, die die Anzahl ihrer Zustände automatisch erweitern können. Diese Modelle wurden ursprünglich entwickelt um die Genauigkeit von *Part-of-Speech* (POS) 'Wortart' Taggern zu erhöhen. Dabei werden die annotierten POS-Tags (z.B. Verb oder Substantiv) in Unterkategorien unterteilt. Ein Expectation Maximization (EM) Algorithmus wird dann benutzt um diese Unterkategorien automatisch an unterschiedliche Rollen anzupassen. So könnte ein Verb in eine Hilfsverb- und Vollverbrolle unterteilt werden. Solch eine Unterteilung ist oft Vorteilhaft für das Modell, da beide Unterklassen in sehr verschiedenen Kontexten auftauchen. So kommen Vollverben oft nach Hilfsverben nicht aber nach anderen Vollverben vor.

Für Deutsch und Englisch zeigen unsere Experimente konsistente Verbesserung über einen Parser (Bohnet, 2010), der keine Unterkategorien verwendet. Wenn wir den Labeled-Attachment-Score (LAS), also die relative Anzahl der Wörter die an ihren richtigen Kopf angehängt werden, betrachten, dann beobachten wir Verbesserungen von 90,34% zu 90,75% für Englisch und von 87,92% zu 88,24% für Deutsch. Für Deutsch zeigt sich des Weiteren, dass unser Modell fast die gleiche Genauigkeit (88,24%) wie ein Modell dass manuell erstellte morphologische Merkmale verwendet (88,35%) erreicht. Auch zeigt sich das die automatisch induzierten Unterkategorien mit den morphologischen Merkmalen korrelieren. So werden z.B. Artikel nach ihrem Kasus unterteilt.

Wir untersuchen auch die *Part-of-Speech*-Tagginggenauigkeit von HMMs die auf traditionelle Baumbank Tagsets und Modellen die auf induzierten Tagsets der selben Größe beruhen. Dabei

zeigen wir, dass letztere performanter als ersteren sind, also das automatisch erzeugte Tagsets besser geeignet sind als die manuell erzeugten. Beide HMMs sind allerdings weniger genau als diskriminative Tagger.

Des Weiteren, präsentieren wir eine Methode zum schnellen und genauen morphologischem Tagging, während Part-of-Speech Tagging Wörter im Kontext mit ihrer Wortart annotiert, produziert morphologisches Tagging eine vollständige Annotation aller relevanten flektionalen Merkmale. Solche Merkmale sind z.B. Kasus, Genus und bei Verben die Zeitform. Beim morphologischem Tagging wird dabei eine vollständige Lesart als ein einzelnes Tag repräsentiert. Da diese Lesarten sehr viele Merkmale enthalten können führt das oft zu Tagsets von hunderten oder sogar tausenden Tags. Das stellt ein Problem für viele Dekodierungsalgorithmen wie z.B. Viterbi dar, da diese of Laufzeiten haben die quadratisch von der Größe des Tagsets abhängen. Diese Problem kann mit Hilfe einer computationellen Morphologie behoben werden. Diese Morphologien sind manuell erstellte Finte-State Transducer, die alle möglichen Lesarten einer Form produzieren. Mit einer Morphologie können die Lesarten eines Wortes gefiltert werden, was dann die Anwendung von Standardmethoden der sequentiellen Vorhersage erlaubt. Ein Nachteil dieses Ansatzes ist jedoch, dass die Ausgabe der Morphologie kompatibel mit den Annotationen der Baumbank sein muss, was oft eine manuelle Konvertierung oder sogar das Entfernen bestimmter Merkmale erfordert.

Um dieses Problem zu vermeiden schlagen wir vor die A-posteriori Wahrscheinlichkeiten eines *Conditional Random Fields* (CRF) (Lafferty et al., 2001) zum filtern der Lesarten zu verwenden. Bei einem Modell nullter Ordnung können diese Wahrscheinlichkeiten unabhängig von den anderen Tags des Satzes berechnet werden. Daher können die nötigen Berechnungen in linearer Zeit erfolgen. Die Merkmale, die das Modell zu diesem Zeitpunkt zur Verfügung hat ähneln den Merkmalen, die eine *Finite-State* Morphologie benutzen würde, können aber auch den unmittelbaren lexikalischen Kontext berücksichtigen. Da die Ambiguität der Wörter stark variiert, setzen wir den Filterschwellwert, so, dass eine bestimmte durchschnittliche Anzahl von Lesarten nach dem Filtern erreicht wird. Nach dem Filtern können wir die Tags mit Übergangskanten verbinden und dadurch ein Modell erster Ordnung erzeugen. Die quadratischen Forward-Backward-Berechnungen, welche nötig sind um die A-posteriori Wahrscheinlichkeiten zu berechnen, sind nun wesentlich effizienter, da sie nur auf den noch verbleibenden Tags ausgeführt werden. Das Filtern und Erhöhen der Modellordnung kann nun, in Abhängigkeit der Filterschwellwerte, zu sehr geringen zusätzlichen Laufzeitkosten erfolgen. Das Modell kann effizient mit einem stochastischen Gradientenabstieg (SGD) trainiert werden. Die CRF-Gradienten können wie üblich von jedem Satz Kandidaten berechnet werden, so lange die korrekten Lesarten noch enthalten sind. Daher filtern wir während des Training nur solange, wie wir nicht die korrekten Lesarten verlieren und berechnen den Gradienten vom letzten noch vollständigen Kandidatensatz. Dieser Ansatz ist vergleichbar mit *Early Updating* in der Perzeptronliteratur und zwingt das Modell dazu die richtigen Lesarten zu filtern. In der Praxis, sehen wir Updates niedriger Ordnung fast nur im ersten Trainingsdurchlauf.

Wir evaluieren unseren CRF-Tagger auf sechs Sprachen mit verschiedenen morphologischen Eigenschaften. Dabei sehen wir, dass das Filtern bei Sprachen mit hoher Wortformambiguität so wie z.B. Deutsch zu einem leichten Rückgang der Tagginggenauigkeit führt, während bei Sprachen mit weniger Ambiguität so wie z.B. Spanisch und Ungarisch der Genauigkeitsverlust

vernachlässigbar ist. Unsere Filterstrategie erlaubt es uns aber Modelle höherer Ordnung zu trainieren, welche bei allen Sprachen zu erheblichen Verbesserung führen auch über ungefilterte Modelle erster Ordnung. Dies bedeutet, dass das Modell einige korrekte Lesarten durch das Filtern verliert, aber dafür auch eine große Anzahl von schwierigen Fällen, die mehr Kontext benötigen richtig lösen. Unser Modell erweist sich auch als akkurater, als die in der Literatur häufig verwendeten Morfette (Chrupala et al., 2008) und SVMTool (Giménez and Màrquez, 2004).

Wir stellen auch eine einfache Methode vor mit der die Performanz eines State-of-the-Art Konstituentenparsers (Petrov et al., 2006) wesentlich erhöht werden kann. Ein Konstituentenbaum bildet die syntaktischen Eigenschaften eines Satzes ab indem er zusammenhängende Wortfolgen auf eine hierarchische Klammerstruktur abbildet. Petrov et al. (2006) entwickelten dazu eine sprachunabhängigen Ansatz der das manuelle Annotieren der Baumbankgrammatik durch eine datengetriebene Annotation ersetzt. Ihre Implementierung, welche als Berkeley Parser bekannt ist, produziert State-of-the-Art Parsingergebnisse für viele Sprachen so wie z.B. Englisch und Deutsch. Für einige MRLs so wie Baskisch und Koreanisch erzielt ihr Parser nur mäßige Ergebnisse, da das Modell für die Behandlung von unbekannten Wörtern zu einfach ist. Dieses Modell bildet unbekannte Wörter auf eine sehr einfache Signatur ab. Diese Signaturen scheinen aber nicht in allen Fällen die Information zu enthalten die man für viele der oft subtilen Parsingentscheidungen braucht. Wir schlagen daher vor, seltene Wörter durch ihre morphologische Lesart zu ersetzen. Diese Lesart kann durch unseren morphologischen Tagger schnell und zuverlässig produziert werden. Unsere Motivation hierbei ist zum einen, dass unser Tagger Zugriff auf eine Menge von lexikalischen und sublexikalischen Merkmalen hat die, der Berkeley Parser nicht benutzt und zum anderen, dass wir vermuten, dass die morphologische Lesart oft ausreicht um eine korrekte Parsingentscheidung zu treffen, auch wenn einige Dinge wie z.B. das korrekte Anhängen von Präpositionalphrasen auch semantische Information benötigen.

In Experimenten auf den SPMRL 2013 (Seddah et al., 2013) Datensätzen von 9 MRLs, führt unsere Methode zu Verbesserungen bei allen Sprachen außer bei Französisch, wo wir einen geringen Rückgang im PARSEVAL-Score von 0.06 beobachten. Für Hebräisch, Ungarisch und Baskisch zeigen sich aber Verbesserungen von 5,65, 11,87 und 15,16 F-Score-Punkten.

Wir präsentieren auch eine umfassende Evaluierung der Nützlichkeit von Wortrepräsentationen für das morphologische Tagging. Unser Ziel dabei ist es den Genauigkeitsrückgang zu reduzieren, der verursacht wird wenn man ein Modell, das auf einer speziellen Domäne trainiert wurde auf eine andere Domäne anwendet. Im Detail untersuchen wir dabei *Brown-Cluster* (Brown et al., 1992a), Singulärwertzerlegungen von Häufigkeitsvektoren (Schütze, 1995) und Einbettungen die auf neuronalen Netzwerken basieren (Collobert et al., 2011). Wir erstellen eine Test-Suite aus sechs Sprachen die *In-Domain* und *Out-Of-Domain* Testdaten enthält. Dazu konvertierten wir die Annotationen von spanischen und tschechischen Datensätzen und erweiterten den deutschen Teil des Smultron Korpus (Volk et al., 2010) um eine manuell annotierte morphologische Schicht. In unseren Experimenten zeigt sich, dass *Brown-Cluster* die beste datengetriebene Repräsentation sind. Im Vergleich mit *Finite-State* Morphologien sind *Brown-Cluster* besser zum *Part-of-Speech*-Tagging geeignet, aber erheblich schlechter für das morphologische Tagging.

Zusammenfassend präsentieren wir ein neuartiges morphologisches Sprachmodell, für welches

wir konsistente Verbesserungen über 21 europäische Sprachen zeigen können. Des Weiteren zeigen wir, dass HMM-LAs interpretierbare Unterkategorien produzieren, welche benutzt werden können um die Genauigkeit eines Dependenzparsers zu erhöhen. Wir stellen auch einen schnellen und akkuraten Ansatz zum morphologischen Tagging vor und seine Anwendung im Konstituenten-Parsing. Schlussendlich diskutieren wir eine Evaluierung von Wortrepräsentationen in der wir zeigen, dass linguistisch motivierte Morphologien am besten für das morphologische Tagging geeignet sind.

Chapter 1

Introduction

1.1 Motivation

Morphologically rich languages (MRLs) can be loosely defined as languages with a morphology more complex than English. MRLs make up a big part of the languages of the world, but are still only inadequately handled by many natural language processing (NLP) models. In this thesis we present several approaches to adapt standard NLP methods to better support MRLs.

Most modern statistical approaches to NLP tasks rely heavily on statistics based on word forms. They require a word form to occur multiple times in a training corpus in order to make reasonable predictions. MRLs present a challenge to such approaches as their productive morphology generates a huge number of rare word forms and thus sparser datasets than languages with simpler morphology. Consider for example a typical regular English verb such as 'to work', which takes four different word forms: 'work', 'works', 'working' and 'worked'. Its Spanish counter part 'trabajar' can take more than 50 different word forms, depending on person, number, tense and mood. A different form of complexity can be seen in the following example from Greenlandic:

Greenlandic	
Paasi-nngil-luinnar-para	*ilaa-juma-sutit*
understand-not-completely-1SG.SBJ.3SG.OBJ.IND	come-want-2SG.PTCP
'I didn't understand at all that you wanted to come along.'	

Table 1.1: Greenlandic as an example of a MRL. (Source: Haspelmath and Sims (2013))

The example shows that a MRL might represent a relatively complex sentence such as 'I didn't understand at all that you wanted to come along' by just two word forms. The second Greenlandic word form *ilaajumasutit*, translates to *wanted to come*. If we assume, that Greenlandic also represents similar phrases such as *wants to dance* or *will want to leave* as single word forms then we are forced to conclude that the sheer number of possible word forms makes it impossible to gather enough statistics for all of them. We thus have to apply some form of decomposition to make use of the fact that the word *ilaajumasutit* is built out of *ilaa*, *juma* and *sutit*

by the application of somewhat general rules. In this thesis we discuss methods that implicitly or explicitly make use of the internal structure of such complex words and we show applications to three standard NLP tasks that suffer from morphological complexity: language modeling, morphological tagging and syntactic parsing:

- Language models assess whether a sequence of words is grammatical, fluent and semantically coherent. They play an important role in standard approaches to machine translation, automatic speech recognition and natural language generation. The most widely used models are n-gram models which model the probability of a word following a number of context words. If the word has not been seen after a given context, the model has to back off to a smaller and less informative context. As MRLs lead to especially sparse n-gram statistics their modeling requires some sort of morphological analysis in order to compensate for the missing lexical information. In this thesis we propose to group word forms of similar morphology and spelling together and thus use a class-based language model. We show that an interpolation of a word-based and morphological class-based model improves the prediction after rare contexts as well as the overall prediction quality. The morphological clusters are built by assigning every word form to a morphological signature. The morphological signature consists of a number of spelling features such as if the word form is in uppercase (a good indicator for proper nouns) or whether the word contains digits and the inflectional suffix of the form. The suffix is extracted by applying an unsupervised segmentation algorithm. To this end, we evaluate several segmentation algorithms and find that most of the improvement can be obtained by a simple frequency-based heuristic.

- A morphological tagger produces the morphological features of a token in context. The resulting morphological readings are useful features for syntactic parsing and machine translation. During tagging the entire morphological reading of a word form is represented as a single tag. As MRLs have many features per word form this leads to big tagsets and high training and tagging times as most sequence prediction decoders depend quadratically on the size of the tagset. We propose an approximate training and decoding algorithm that prunes implausible readings from the tag lattice and yields substantial decoding speed ups. This new algorithm also allows us to increase the order of the model, which is prohibitive for standard models as it would mean an exponential increase in the runtime of the model. We then show that pruned higher-order models give significant improvements over unpruned first-order models. We also created a test-suite for robust morphological tagging. This test-suite was built by converting existing data sources of different annotations and also by annotating new resources. We use this test-suite to perform a large evaluation of the influence of word representations on morphological tagging.

- A syntactic parser produces the syntactic analysis of a sentence. The syntactic analysis is typically in tree form and describes the syntactic dependencies between words or the hierarchical phrasal structure of the constituents. The resulting syntactic trees are crucial for many NLP problems related to natural language understanding such as coreference resolution and information extraction, but are also frequently used in other tasks such as machine translation and paraphrasing. A proper morphology analysis is important for

parsing as many syntactic roles are tightly correlated with certain morphological features. Dative case, for example usually indicates an indirect object. Our contribution here is twofold: We first show that hidden Markov models with latent annotations (HMM-LA) can be used to induce latent part-of-speech tags that for some languages (such as German) act as a form of unsupervised morphological tags and significantly increase parsing accuracy. Furthermore, we show that morphological signatures obtained from our supervised morphological tagger lead to substantial improvements, when incorporated into a state-of-the art constituency parser.

1.2 Main Contributions

The issues with applying standard NLP approaches, mentioned in the last section, lead us to the development of several supervised and unsupervised approaches to improve the handling of MRLs.

- **Morphological language model:** Addressing the limitations of n-gram models, we present a novel morphological language model. The model is a linear interpolation between a standard word-based model and a class-based morphological model. The class-based model is built by grouping infrequent word forms of similar spelling and morphology. In an evaluation across 21 languages of varying morphological complexity the interpolated model yields consistent improvements over a word-based model. The model gives high improvements for MRLs (e.g., Finnish) and lower improvements for morphologically simpler languages (e.g., English).

- **HMM-based feature induction:** We discuss hidden Markov models with latent annotations as a method to induce tagsets with interesting linguistic properties and some correlation with morphological features. The induced tagset can be used to improve statistical dependency parsing with significant improvements for English and German. For German, the improvements obtained from the tagset and the improvements obtained when using a supervised tagger are statistically indistinguishable. The approach can be interpreted as a form of semi-supervised morphological tagging, as the German tags also show some correlation with morphological features such as case.

- **Fast morphological tagging:** We present a novel approximating decoding algorithm for Conditional Random Fields. The algorithm creates a sequence of pruned tag lattices of increasing order and complexity and can be applied during training and decoding. It is especially appropriate when used with high model orders and tagset sizes. For morphological tagging, the algorithm turns out to be several times faster than unpruned first-order tagging and more accurate when used with higher orders. We also show how the outcome of the resulting tagger can be used to improve the modeling of rare words in a state-of-the art constituency parser.

- **Robust morphological tagging:** We present a dataset for the evaluation of multilingual morphological tagging and a survey on the utility of word representations. We show that

a simple clustering approach known as Brown clustering (Brown et al., 1992a) yields the highest improvements in part-of-speech tagging. For full morphological tagging we find that handcrafted computational morphologies outperform all the tested data-driven representations by a great margin.

1.3 Outline

The remainder of this thesis is structured as follows: Chapter 2 begins by introducing the needed linguistic and mathematical foundations. We discuss the necessary linguistic terminology and the basics of statistical language modeling using n-grams as well as class-based language modeling, the Kneser-Ney model and linear interpolation as a simple way of building integrated language models. We look at structured prediction, hidden Markov models with and without latent annotations and conditional random fields (CRFs). We also survey the most important word representations. In Chapter 3 we discuss the morphological class-based language model and an extensive evaluation across 21 languages. Chapter 4 introduces the application of HMMs with latent annotations to induce a latent part-of-speech tagset useful for statistical dependency parsing. We explain the utility of the tagset by pointing out its correlation with certain morphological features and semantic classes. In Chapter 5 we present the approximated inference algorithm for CRFs. Chapter 6 discusses robust morphological tagging. We investigate the out-of-domain performance of our CRF tagger and survey the utility of different word representations.

Chapter 2

Foundations

2.1 Morphology

In this section we discuss the linguistic terminology used throughout this thesis. We begin with the definition of **morphology**. Morphology is the study of the internal structure of words. More precisely it studies how **complex words** are built using smaller segments, called **morphemes** and how these morphemes determine the semantic properties of words. Consider for example English regular plural building: *cats* is a complex word that is a concatenation of *cat* and *s*. The morpheme *s* at the end of the word indicates that we are referring to more than one cat and the process is systematic, as can be seen in many other examples such as *dog* - *dog-s*, *cow* - *cow-s* and *horse* - *horse-s*. Throughout this section we denote morpheme boundaries by '-'. The different languages of the world have a different degree of morphological and syntactic complexity. Languages that tend to express semantic properties using syntax are called **analytic** while languages that express these properties mostly using morphology are called **synthetic**. Two examples from Vietnamese (analytic) and Greenlandic (synthetic) are shown in Figure 2.1.

Vietnamese

Hai	*dú.a*	*bo*	*nhau*	*là*	*tai*	*gia-dinh*	*thang*	*chông.*
two	individual	leave	each.other	be	because.of	family	guy	husband

'They divorced because of his family.'

Greenlandic

Paasi-nngil-luinnar-para	*ilaa-juma-sutit*
understand-not-completely-1SG.SBJ.3SG.OBJ.IND	come-want-2SG.PTCP

'I didn't understand at all that you wanted to come along.'

Figure 2.1: Examples of an analytic and a synthetic language. (Source: Haspelmath and Sims (2013))

Synthetic languages can be further subdivided into fusional and agglutinative languages. Ag-

glutinative languages represent most morphological features by a specific morpheme. Consider for example the Hungarian paradigm in Figure 2.2, where 'ok' marks plural and 'al' marks instrumental case. Fusional languages such as German (Figure 2.2) on the other side tend to merge different features into one morpheme.

	Singular	Plural
	Hungarian	
Nominative	*nap*	*nap-ok*
Dative	*nap-ot*	*nap-ok-at*
Accusative	*nap-nak*	*nap-ok-nak*
Instrumental	*napp-al*	*nap-okk-al*
	...	
	German	
Nominative	*Tag*	*Tag-e*
Genitive	*Tag-es*	*Tag-en*
Dative	*Tag*	*Tag-en*
Accusative	*Tag*	*Tag-e*

Figure 2.2: (Partial) Paradigms of an agglutinative (Hungarian) and a fusional (German) language

2.1.1 Terminology

We call an occurrence of a word in running text **word form** and the set of word forms that is represented by the same entry in a lexicon **paradigm** or **lexeme**. The canonical form that is used to represent the paradigm is called **lemma**. An example for the Danish word *dag* 'day' is shown in Figure 2.3.

	Singular		Plural	
	Indefinite	Definite	Indefinite	Definite
Nominative	*dag*	*dagen*	*dage*	*dagene*
Genitive	*dags*	*dagens*	*dages*	*dagenes*

Figure 2.3: Paradigm of the Danish noun *dag* 'day'

The example shows that the word forms *dag*, *dagen*, *dage*, *dagene*, *dags*, *dagens*, *dages* and *dagenes* make up a lexeme that can be summarized by the lemma *dag*, which corresponds to the singular nominative indefinite form. A set of related lexemes is called a **word family**. Consider for example the two families in Figure 2.4.

a) *build*, *build-ing*, *build-er*, *build-able*, *un-build-able*, ...
b) *mort*, *mort-al*, *mort-al-itat*, *in-mort-al*, *in-mort-al-izar*, *in-mort-al-izasión*, ...

Figure 2.4: An English (a) and a Catalan (b) word family

The relationship between different word forms of a lexeme is called **inflectional**, while the relationship between different lexemes of a word family is called **derivational** (Haspelmath and Sims, 2013).

An important part of morphology is **concatenative** and can be subdivided into **affixation** and **compounding**. Affixation is the attachment of morphemes with abstract meaning to morphemes with concrete meaning. The morphemes with concrete meaning are called **roots**, while the abstract morphemes are known as **affixes**. Affixes get further specified by how they are attached to the root. Affixes preceeding the root are called **prefixes**, affixes succeeding the root **suffixes**, affixes inserted into the root are called **infixes** and affixes surrounding the root **circumfixes**. In the example above *build* and *mort* are roots, *in* and *un* are prefixes and *er* and *al* are suffixes. In Seri the infix *tóo* is a plural marker: consider for example *ic* 'plant' versus *i-tóo-c* 'plants'. The German *ge-t* is a circumfix such as in *ge-sag-t* 'said'. The – possibly complex – unit an affix is attached to is known as **base** and also as **stem** in inflectional morphology. Morphemes that can be found attached to word forms, but also as free words are called **clitics**. An example are Spanish object pronouns such as *lo*. Consider for example, *lo compré* 'I bought it', but *compra-lo* 'buy it'. In compounding, words are built by concatenating existing lexemes. In many languages – such as English – noun-noun compounds make up the majority of the compounded words, examples include *home-work* and *fire-fly*, but also *green tea* and *snow storm*. In other languages different types of compounds might be more frequent. In modern Spanish, for example, verb-noun compounds are more frequent: *lava-platos* 'dish washer' (lit., washes plates) and *rompe-cabezas* 'riddle' (lit., breaks heads).

Morphological changes that cannot be explained by compounding or affixation are called **nonconcatenative**. An important class of these changes is **stem alternation**, where the stem is changed during the creation of a word form. Stem alternation often effects vowels. Consider for example the English *goose* – *geese* and sleep – *slep-t*, German *buch* 'book' – *büch-er* 'books' or Spanish *quier-o* 'I want' – *quer-emos* 'we want'. A class of nonconcatenative morphology that is important for semitic languages such as Hebrew and Arabic is **transfixation**. In transfixation the root can be seen as a consonant pattern where vowel features get inserted in order to form the word. Consider the example in Figure 2.5.

Active Perfect	Passive Perfect	Root Pattern
kataba 'wrote'	kutiba 'was written'	k-t-b
halaqa 'shaved'	huliqa 'was shaved'	h-l-q
farada 'decided'	furida 'was decided'	f-r-d

Figure 2.5: Examples of Arabic transfixation (Source: Haspelmath and Sims (2013))

The example demonstrates how for example the form *kataba* 'wrote' is created by mixing the inflectional active perfect pattern *a-a-a* with the root pattern *k-t-b*. There are many other classes of nonconcatenative morphology for which we refer the reader to Haspelmath and Sims (2013).

2.1.2 Part-Of-Speech

This thesis focuses in part on the accurate prediction of morphological features that often represent inflectional properties of the word forms. In the remainder of this section we give an overview of the most common parts of speech and their typical inflectional properties. **Nouns** and **pronouns** represent entities such as persons, animals and abstract concepts. English examples include *dog*, *house* and *democracy*. Pronouns such as *it*, *her* and *him* are substitutes for nouns. The most important categories are **personal pronouns** such as *he* and *him*, **possessive pronouns** such as *his* which denote possession of something, **reflexive pronouns** such *herself* which usually refer to a previous mention of the subject of a sentence and **demonstrative pronouns** such as *this* and *that* which are identified by some external reference (such as a gesture or the distance to the speaker). The four typical morphological features are **number**, **gender**, **case** and **definiteness**. We already discussed that number specifies the number of entities referred to. The most common values are **singular** (sg) and **plural** (pl), but some languages also have a **dual** form to refer to exactly two entities. It is often used to refer to things that naturally occur as a pair such as arms and legs. Gender might denote the grammatical gender or natural gender of an entity. Languages such as German and Spanish have grammatical gender, which is also assigned to lifeless objects and is not necessarily consistent between different languages: compare German masculine *der Mond* 'moon' (masculine) to Spanish *la luna* 'moon' (feminine). The typical gender values are **masculine** (masc), **feminine** (fem) and **neuter** (neut). Although gender is usually specified by the root, some languages also have certain derivational affixes that fix the gender of a form. In Spanish *-cion* as in *comunicación* indicates feminine gender, while the German diminutive suffix *-chen* such as in *Mädchen* 'girl' indicates neuter. Some languages such as for example Basque only make a distinction into **animate** and **inanimate**, although this is sometimes considered as a separate morphological feature. Case reflects the grammatical function of a word form, while languages can have many different case values. In Figure 2.6, we give examples for the classical Latin cases.

Case	Function	Latin	English
Nominative	subject	**nauta** ibi stat	'the sailor is standing there'
Genitive	possessing object	nomen **nautae** Claudius est	'the sailor's name is Claudius'
Dative	indirect object	**nautae** donum dedi	'I gave a present to the sailor'
Accusative	direct object	**nautam** vidi	'I saw the sailor'
Ablative	various uses	sum altior **nauta**	'I am taller than the sailor'
Vocative	addressing the object	gratias tibi ago, **nauta**	'I thank you, sailor'

Figure 2.6: Cases of nauta 'sailor' (Source: Wikipedia.org)

Definiteness marks whether (**definite**) or not (**indefinite**) a referred entity is identifiable in a

given context.

Determiners (such as ‘a’ and ‘the’) and adjectives (such as ‘red’ and ‘tired’) accompany nouns and agree in their morphological features. Adjectives can be used attributively (*the red car*) or predicatively (*the car is red*) and their agreement behavior may vary in different uses: consider German *ein rotes Auto* ‘A red car’, but *ein Auto is rot* (a car is red). Whether certain features are marked might depend on special circumstances. Spanish and German adjectives for example mark gender, but German only does so (unambiguously) in very special cases: If the adjective gets used as an attribute, with indefinite determiner and only in nominative and accusative case. Figure 2.7 illustrates this.

Spanish	Sg	Pl
Masc	rojo	rojos
Fem	roja	rojas

German	Sg	Pl
Masc	roter	rote
Fem	rote	rote
Neut	rotes	rote

Figure 2.7: Noun-Adjective gender and plural agreement in Spanish and German for ‘red’

Verbs represent the actions performed by entities. The common morphological features are **person**, **number**, **tense**, **voice** and **mood**. Figure 2.8 shows the paradigm of the Dutch verb *jagen* ‘hunt’.

	Present Tense	Past Tense
1st person singular	jaag	joeg
2nd person singular	jaagt	joeg
3rd person singular	jaagt	joeg
Plural	jagen	joegen
Subjunctive singular	jage	joege
Subjunctive plural	jagen	joegen
Imperative singular	jaag	
Imperative plural	jaagt	
Participles	jagend	gejaagd

Figure 2.8: Paradigm of the Dutch verb *jagen* ‘hunt’. (Source: en.wiktionary.org)

In many European languages person denotes whether some action is performed by the speaker, someone directly addressed by the speaker or someone else. It might also include a formal and informal way of addressing someone or other distinctions. Quechua, for example, has an inclusive and exclusive *we* form. Tense marks when an action takes place and how it temporarily relates to other actions. The classical Latin tenses are: present, imperfect, perfect, pluperfect

(before some event in the past), future and future perfect (event happening before point of time in the future: 'will have seen'). Voice specifies whether the action is active or passive and mood specifies the attitude of the speaker. Classical mood features are **indicative** (standard form), **subjunctive** (hypothetical), **conditional** (dependent on some other statement) and **imperative** (direct command).

The remaining most important parts of speech are **adverbs**, **prepositions** and **conjunctions**. Adverbs such as *often* and *hastily* modify verbs just as adjectives modify nouns. Prepositions such as *on* and *in* are small words that express spatial, temporal or abstract relationships with a noun. Synthetic languages such as Hungarian might often use case marking in places where analytic languages use prepositions. Conjunctions such as *and* and *that* link sentences. **Coordinate conjunctions** join phrase of the (usually) same category (*Mary and John*), while **subordinate conjunctions** attach a primary sentence to a secondary sentence (*Mary said that she likes John*).

2.2 Language Modeling

Language models estimate the probability of a sentence and are important for NLP applications such as machine translation and speech recognition, where they are needed to estimate the grammaticality, fluency and semantic coherence of a sentence or utterance. In practice we are given a set of sentences D called the training set and are trying to estimate the probability distribution P_D that generated the data set. We assume that the words w generated by P_D are members of a finite vocabulary V. We define V to contain all the word forms occurring in D plus the unknown word which represents all the word forms of a language not occurring in D. As the term word is ambiguous we refer to the members of V by (word) type and and call occurrences of types in the running text (word) tokens.

n-gram Models

We first use the chain rule to decompose the probability of a sequence $w_1^k = w_1 w_2 w_3 \dots w_k$ (with $w_i \in V$) into a product of conditional probabilities:

$$P(w_1 \dots w_k) = \prod_{i \leq k} P(w_i | w_1^{i-1}) \tag{2.1}$$

The conditional probabilities are also called transitional probabilities and their right side is known as the history of w_i. Note that the last probability distribution $P(w_k|w_1^{k-1})$ would need $|V|^k$ parameters if stored as a simple probability table. As it is infeasible to estimate such probability tables for large sentences we make a Markov assumption and bound the number of words in our history by $n-1$, where n is called the order of the model.

$$P(w_i|w_1^{i-1}) \approx P(w_i|w_{i-n+1}^{i-1}) = P(w_i|h_i) \tag{2.2}$$

Such models are called n-gram models as the largest word sequences they rely on are of order n. We assume that histories with indexes < 1 are padded with distinctive boundary symbols.

Maximum Likelihood Estimate

The simplest way of estimating the probability distribution that generated D is to assume that it is the empirical distribution maximizing the likelihood $\mathrm{LL}(D)$ of D, the probability that D is generated by the distribution P_θ:

$$\mathrm{LL}(D) = P_\theta(D) = \prod_{i=1}^{N} P_\theta(w_i|h_i) \tag{2.3}$$

where N is the number of tokens in D. If P is an n-gram model we can express the log-likelihood $\mathrm{ll} = \log \mathrm{LL}$ in terms of the parameters λ_{hw} of the probability distribution $P_\theta(w|h)$:

$$\mathrm{ll}(D, \theta) = \sum_{hw \in V^n} c(hw) \log P_\theta(w|h) = \sum_{hw \in V^n} c(hw) \log \lambda_{hw} \tag{2.4}$$

where θ is a tuple containing all the model parameters λ and $c(hw)$ is the frequency of the n-gram hw in D. In order to satisfy the axioms of probability theory, the parameters λ belonging to the same history have to sum up to 1. We build this constraint into the objective function by adding the Lagrange multiplier π. As the probabilities for different histories are independent of each other we can optimize for one history at a time:

$$\mathrm{ll'}(D, \theta, h) = \sum_{hw \in V^n} c(hw) \log \lambda_{hw} + \pi \cdot (\sum_{w \in V} \lambda_{hw} - 1) \tag{2.5}$$

As ll' is concave, we can derive optimal model parameters by finding the root of its gradient. Differentiating for the model parameters λ_{hw} and π yields:

$$\frac{\partial\, \mathrm{ll'}(D, \theta, h)}{\partial \lambda_{hw}} = \frac{c(hw)}{\lambda_{hw}} + \pi \tag{2.6}$$

$$\frac{\partial\, \mathrm{ll'}(D, \theta, h)}{\partial \pi} = \sum_{w \in V} \lambda_{hw} - 1 \tag{2.7}$$

Setting the derivative to zero we arrive at:

$$\lambda_{hw} = \frac{c(hw)}{\sum_{w' \in V} c(hw')} \tag{2.8}$$

We can conclude that setting the n-gram model parameters to their relative frequencies in the data set D maximizes the likelihood of D. A model so defined is called the maximum likelihood estimate and with $c(h\bullet) = \sum_{w' \in V} c(hw')$ for simplicity we set:

$$P_{\mathrm{ML}}(w|h) = \frac{c(hw)}{c(h\bullet)} \tag{2.9}$$

While this model is the n-gram model closest to the empirical data distribution it is bad in practice. The reason is that it assigns zeros to all n-grams not seen in D. In order to get

some estimate for unseen n-grams we need to redistribute some of the probability mass of seen events to unseen events. While doing that we are trying to change the seen counts as little as possible. As we do not want to redistribute the probability mass uniformly we also need a way of estimating the probability of a higher order n-gram by looking at lower order n-grams. In the next subsection we discuss linear interpolation as a way of estimating higher order n-grams from lower order n-grams and as a general way of combining language models. The redistributions of probability mass to unseen events is known as discounting or smoothing and is discussed afterwards.

Linear Interpolation

The basic form of linear interpolation (Jelinek, 1980; Brown et al., 1992b) between two models can be given as:

$$P_{\text{inter}}(w|h) = \gamma(h) \cdot P_1(w|h) + (1 - \gamma(h)) \cdot P_2(w|h), \text{ with } 0 \leq \gamma(h) \leq 1 \tag{2.10}$$

It is guaranteed that when the $\gamma(h)$ are chosen in an optimal way wrt to the data set D the log-likelihood of P_{inter} will not be smaller than the likelihood of both P_1 and P_2. Another nice property of linear interpolation is that the new model can be encoded as a simple n-gram model.

As the likelihood of the interpolated model is concave the optimal interpolation parameters can be found in a similar fashion as shown in the previous subsection. In this thesis we use the algorithm by Bahl et al. (1991), which finds the optimal values using binary search in the interval $[0, 1]$ of the gradient (Algorithm 2.1).

Algorithm 2.1 The Bahl Algorithm for finding optimal interpolation parameters

function BAHL-OPTIMIZE($\text{ll}_{\text{inter}}, D$)
  $\gamma_l \leftarrow 0$
  $\gamma_r \leftarrow 1$
  while true do
    $\gamma_m \leftarrow \frac{\gamma_l + \gamma_r}{2}$
    if $\frac{\partial \, \text{ll}_{\text{inter}}(D, \gamma_m)}{\partial \gamma} = 0$ **then**
      return γ_m
    else if $\frac{\partial \, \text{ll}_{\text{inter}}(D, \gamma_m)}{\partial \gamma} > 0$ **then**
      $\gamma_l \leftarrow \gamma_m$
    else
      $\gamma_r \leftarrow \gamma_m$
    end if
  end while
end function

The algorithm uses the log-likelihood of an interpolated model:

$$\mathrm{ll}_{\mathrm{inter}}(h, D, \gamma(h)) = \sum_{hw \in D} c(hw) \cdot \log \gamma(h) \cdot P_1(w|h) + (1 - \gamma(h)) \cdot P_2(w|h) \tag{2.11}$$

$$\frac{\partial \, \mathrm{ll}_{\mathrm{inter}}(h, D, \gamma(h))}{\partial \gamma(h)} = \sum_{hw \in D} \frac{c(hw)}{\gamma(h) + \frac{P_2(w|h)}{P_1(w|h) - P_2(w|h)}} \tag{2.12}$$

Linear interpolation can also be generalized to an arbirary number of models:

$$P_{\text{general-inter}}(w|h) = \sum_i \gamma_i(h) \cdot P_i(w|h), \text{ with } \sum_i \gamma_i(h) = 1 \tag{2.13}$$

The parameters γ_i can then be estimated using a generalized version of Bahl et al. (1991) or using the expectation-maximization algorithm (EM) (Dempster et al., 1977). The general idea behind EM is discussed in Section 2.3.2 for the implementation in the case of linear interpolation we refer to Clarkson and Rosenfeld (1997).

Relative Discounting

In order to avoid zeroes in the probability distributions we redistribute mass from seen to unseen events. The easiest way to accomplish this is to add a small positive δ to every count. This results in the following conditional probabilities:

$$P_{\mathrm{ADD}}(w|h) = \frac{c(hw) + \delta}{c(h\bullet) + |V| \cdot \delta} \tag{2.14}$$

This form of smoothing is known as additive smoothing and the special case of $\delta = 1$ as Laplace smoothing. We can also rewrite additive smoothing as a linear interpolation between a ML model and a uniform distribution:

$$\begin{aligned} P_{\mathrm{ADD}}(w|h) &= \frac{c(hw) + \delta}{c(h\bullet) + |V| \cdot \delta} \\ &= \frac{c(h\bullet)}{c(h\bullet) + |V| \cdot \delta} \cdot \frac{c(hw)}{c(h\bullet)} + \frac{\delta \cdot |V|}{c(h\bullet) + |V| \cdot \delta} \cdot \frac{1}{|V|} \\ &= \gamma(h) \cdot P_{\mathrm{ML}}(w|h) + (1 - \gamma(h)) \cdot \frac{1}{|V|} \end{aligned}$$

An important generalization of additive smoothing is known as relative discounting:

$$P_{\mathrm{REL}}(w|h) = \gamma(h) \cdot P_{\mathrm{ML}}(w|h) + (1 - \gamma(h)) \cdot P_{\mathrm{SMOOTH}}(w|h) \tag{2.15}$$

where $P_{\mathrm{SMOOTH}}(w|h)$ is an arbitrary smooth distribution. Gale and Church (1994) showed that plain additive smoothing does not work well for language modeling, as even for small n the number of seen events is much smaller than the number of unseen events and we thus remove too much probability mass from frequent events.

Witten-Bell Smoothing

Witten-Bell Smoothing (Witten and Bell, 1991) is a form of relative discounting. Similar to additive smoothing it is motivated by the idea that for a frequent history we should make less usage of the back-off distribution, but this should not be our only criterion to use: Compare for example the two histories *said Mr.* and *wish you a merry*. The first history is much more likely to be followed by a new word than the second, even though it should be more frequent in most corpora. Witten-Bell Smoothing thus incorporates the number of words a history is followed by into the interpolation parameters:

$$P_{\mathrm{WB}}(w|h) = (1 - \gamma_{\mathrm{WB}}(h)) \cdot P_{\mathrm{ML}}(w|h) + \gamma_{\mathrm{WB}}(h) \cdot P_{\mathrm{WB}}(w|h') \tag{2.16}$$

$$\gamma_{\mathrm{WB}}(h) = \frac{N_{1+}(h\bullet)}{N_{1+}(h\bullet) + c(h\bullet)} \tag{2.17}$$

where h' is a history derived from h by removing the left most word and where we define N_{1+} as the number of words following a certain history:

$$N_{1+}(h\bullet) = |\{w : c(hw) > 0\}| \tag{2.18}$$

Absolute Discounting

Absolute Discounting (Ney and Essen, 1991; Ney et al., 1994) is based on the intuition of removing relatively more mass from small counts than from higher counts by simply subtracting a small positive constant $D < 1$:

$$P_{\mathrm{ABS}}(w|h) = \frac{\max\{c(hw) - D, 0\}}{c(h\bullet)} + \gamma_{\mathrm{ABS}}(h) \cdot P'_{\mathrm{SMOOTH}}(w|h') \tag{2.19}$$

Again, $\gamma_{\mathrm{ABS}}(h)$ is set so that P_{ABS} is properly normalized. The method can be justified by experiments made by Church and Gale (1991). Given n-grams with a specific count r in one corpus they looked at the average of counts r' these n-grams had in a second corpus. They found that the difference between r and r' was almost constant for $r \geq 3$. With a derivation using deleted interpolation on the training set Ney et al. (1994) set:

$$D = \frac{N_1}{N_1 + 2 \cdot N_2} \tag{2.20}$$

where N_1 and N_2 are the total numbers of n-grams with count 1 and 2, respectively. Based on the observation made by Church and Gale (1991), Chen and Goodman (1999) later argued to use three different D_i for the n-grams with count 1, 2 and ≥ 3:

$$D_i = i - (i + 1) \cdot D \cdot \frac{N_{i+1}}{N_i} \tag{2.21}$$

Note that $D_1 = D$.

Kneser-Ney Smoothing

Kneser-Ney smoothing (Kneser and Ney, 1995) is arguably the best performing n-gram smoothing technique and a variant of Absolute Discounting. The back-off distribution $P'_{\text{KN}}(w|h)$ is defined as the ratio of n-grams of the highest order ending in hw and n-grams ending in the history h and some other word:

$$P_{\text{KN}}(w|h) = \frac{\max\{c(hw) - D, 0\}}{c(h\bullet)} + \gamma_{\text{KN}}(h) \cdot P'_{\text{KN}}(w|h') \tag{2.22}$$

$$P'_{\text{KN}}(w|h) = \frac{\max\{N_{1+}(\bullet hw) - D, 0\}}{N_{1+}(\bullet h\bullet)} + \gamma'_{\text{KN}}(h) \cdot P'_{\text{KN}}(w|h') \tag{2.23}$$

where N_{1+} is defined as the number of n-grams of the highest order n that have been seen in D and end in hw:

$$N_{1+}(\bullet hw) = |\{xhw|c(xhw) > 0 \wedge |xhw| = n\}| \tag{2.24}$$

where x is an n-gram and $|xhw|$ denotes the order of the n-gram xhw. Similarly $N_{1+}(\bullet h\bullet)$ is defined as:

$$N_{1+}(\bullet h\bullet) = \sum_{w} N_{1+}(\bullet hw) \tag{2.25}$$

The model thus only uses the discounted MLE counts for the highest order model, because lower-order counts might lead to wrong predictions. For the sake of argument, assume that we use a Witten-Bell model to decide which of the two phrases in Figure 2.9 is correct.

1. *He was filled with spite for his ex-wife.*
2. **He was filled with spite of his ex-wife.*

Figure 2.9: Example of a problematic case when simply using lower-order n-gram counts to estimate higher-order n-gram counts: The high frequency of *in spite of* will make the incorrect second sentence more likely. '*' denotes an ungrammatical sentence.

We assume that we have not seen any of the *with spite for/of* n-grams and thus have to back-off to bigram probabilities to settle the matter. As *spite* will occur almost exclusively as *in spite of*, we will decide on the incorrect *of*. A Kneser-Ney model would estimate how likely the bigrams are to occur in a novel context by looking at the number of different n-grams they occur in and might come up with a different prediction.

Class-based Language Models

Class-based language models (Brown et al., 1992a) define n-gram probabilities by mapping word forms to word classes and by estimating probabilities over class n-grams:

$$P_{\text{CLASS}}(w|h = w_1^k) = P(g(w)|g(w_1)\dots g(w_k)) \cdot P(w|g(w)) \tag{2.26}$$

$P(g(w)|g(w_i)\dots g(w_{i+k}))$ is called transition probability and $P(w|g(w))$ emission probability. The class assignment function $g(w)$ maps word types to classes. Class-based LMs can be seen as another smoothing method as – similar to back-off models – they group different word n-grams in order to obtain more reliable statistics. Class-based models are usually combined with word-based models in order to obtain optimal performance (Goodman and Gao, 2000; Uszkoreit and Brants, 2008). Methods to estimate class assignment functions from data are discussed in Section 2.4 and Chapter 3 .

Evaluation of statistical Language Modeling

Language models can be evaluated as part of machine translation or speech recognition systems, but in this thesis we prefer the intrinsic evaluation using perplexity. We already discussed log-likelihood as a way of evaluating the performance of a language model:

$$\text{ll}(D, \theta) = \sum_{hw \in V^n} c(hw) \log P_\theta(w|h) \tag{2.4}$$

However, as ll declines with the number of tokens in the test corpus and because it is always negative, the cross entropy between the empirical distribution of the test corpus and the model has been proposed as an alternative evaluation measure, more suited to compare values from different corpora:

$$\text{H}(D, \theta) = -\frac{\text{ll}(D, \theta)}{N} = -\sum_{hw \in V^n} \frac{c(hw)}{N} \log P_\theta(w|h) \tag{2.27}$$

From an information theoretic perspective the entropy is the "average number of bits per word that would be necessary to encode the test data using an optimal coder" (Goodman, 2001). However, entropy is not a very intuitive measure as the difference between two models is often extremely small. We thus use perplexity, which is defined as the cross entropy raised to the power of 2:

$$\text{PP}(D, \theta) = 2^{\text{H}(D,\theta)} = 2^{-\sum_{hw \in V^n} \frac{c(hw)}{N} \log P_\theta(w|h)} \tag{2.28}$$

The nature of perplexity can be revealed if it is applied to a simple uniform model over the vocabulary V:

$$\begin{aligned} \text{PP}(D, \theta = (\frac{1}{|V|})) &= 2^{-\sum_{hw \in V^n} \frac{c(hw)}{N} \log \frac{1}{|V|}} \\ &= |V| \end{aligned}$$

The perplexity of a uniform distribution is thus just the number of possible outcomes. Intuitively, perplexity measures the average number of guesses the model has to make in order

to generate the test data. Both cross entropy and perplexity are minimized by the MLE of the empirical distribution of the test set. As already mentioned, perplexity is the preferred measure in the literature, because it is more responsive to small changes in the model, e.g., reducing H by 1 bit is equivalent to reducing PP by 50%.

2.3 Sequence Prediction

In this section we discuss sequence prediction models which are the standard tools to address many important NLP problems such as part-of-speech and morphological tagging, named-entity recognition and information extraction. Sequence prediction – or more generally structured prediction – can be seen as a form of classification. Classification is formally defined in the following way: Given a set of objects $\mathcal{X}$ and a discrete set of classes $\mathcal{Y}$. A classifier is a function $f(\mathbf{x}) = \mathbf{y}$. The estimation problem is to find a *good* classifier given some training examples $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$. Throughout this section we only discuss probabilistic classifiers and thus define *good* as maximizing the conditional likelihood of the training data.

Noisy Channel

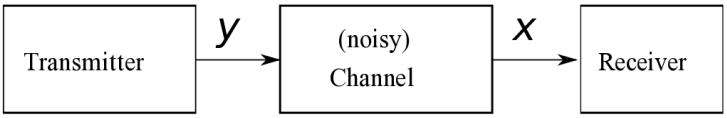

Figure 2.10: Scheme of the noisy channel model (Source: Wikimedia.org)

The problem behind sequence prediction can be interpreted as a channel model. A channel model describes the following general scenario: Someone sent us a transmission $\mathbf{y}$ through a noisy channel which converted it to a new sequence $\mathbf{x}$. The conversion process is nondeterministic and can thus not be reverted in general. Consider for example the following famous example form part-of-speech tagging where given the sequence of words both tag sequences can be considered correct:

Time	flies	like	an	arrow
Noun	Noun	Verb	Det	Noun
Noun	Verb	Conj	Det	Noun

Figure 2.11: Example of a phrase with ambiguous part-of-speech

We thus have to approximate $\mathbf{y}$ by $\hat{\mathbf{y}}$, the most probable explanation for $\mathbf{x}$. We thus define the sequence prediction problem as: Given a sequence of input symbols $\mathbf{x} = x_1 \dots x_T = x_1^T$ with $x_t \in \mathcal{X}$. Find the most probable output symbols $\mathbf{y} = y_1^T$ with $y_t \in \mathcal{Y}$. We denote the values that the y_i can assume by $y^{(j)}$. By convention, we have $y_0 = \mathrm{start}$ and $y_{T+1} = \mathrm{stop}$.

2.3.1 Hidden Markov Models

Hidden Markov models (HMMs) are still a heavily used solution to sequence prediction problems even though they are usually outperformed by the more complex discriminative models discussed later in this section. HMMs can be derived using classical probability theory. We are looking for a model of the conditional probability distribution $P(\mathbf{y}|\mathbf{x})$. Using Bayes' theorem we can rewrite this probability as:

$$P(\mathbf{y}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})}$$

As the input symbols $\mathbf{x}$ are given in sequence prediction, there is no need to model them and we can reduce the problem to:

$$P(\mathbf{y}|\mathbf{x}) \propto P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) = P(\mathbf{x}, \mathbf{y})$$

$P(\mathbf{y})$ is essentially a language model over output symbols and just like in language modeling we model it by applying the chain rule followed by a Markov assumption:

$$\begin{aligned} P(\mathbf{y}) &= \prod_{t=1}^{T+1} P(y_t|y_0^{t-1}) \\ &\approx \prod_{t=1}^{T+1} P(y_t|y_{t-1}) \end{aligned}$$

$P(\mathbf{x}|\mathbf{y})$ is the probability that the input sequence observed has been generated by the output sequence. We model it using the chain rule and by making the independence assumption, that every input symbol only depends on its direct output symbol:

$$P(\mathbf{x}|\mathbf{y}) = \prod_{t=1}^{T+1} P(x_t|\mathbf{y}, x_0^{t-1}) \tag{2.29}$$

$$\approx \prod_{t=1}^{T+1} P(x_t|y_t) \tag{2.30}$$

We thus arrive at the final model:

$$P(\mathbf{y}|\mathbf{x}) \propto P(\mathbf{y}, \mathbf{x}) \approx \prod_{t=1}^{T+1} P(y_t|y_{t-1})P(x_t|y_t) \tag{2.31}$$

HMMs thus have the form of a class-based language model (Eq. 2.26) with the important difference that the assignment of input symbol to output symbol is nondeterministic. The emission and transmission probabilities can be estimated with the smoothed probability distributions known from language modeling (e.g., Witten-Bell smoothing). It is however important to model

the emission probability so that it also makes reasonable predictions for unseen input symbols. This can be done by modeling $P(y|x)$ (after applying Bayes' theorem):

$$P(\mathbf{y}, \mathbf{x}) \approx \prod_{t=1}^{T+1} P(y_t|y_{t-1}) \frac{P(y_t|x_t)}{P(y_t)}$$

In many NLP problems, the input sequence consists of word forms, these can be effectively modeled using morphological and distributional features such as word representations.

Viterbi

Decoding in a HMM is done by enumerating all the possible state sequences and choosing the sequence with the highest probability. The number of possible sequences is exponential in the number of output symbols N (T^N), but can be efficiently enumerated using dynamic programming. We define $\mathrm{v}(t, y)$ as the probability of the most likely output sequence ending at position t and with output symbol y. $\mathrm{v}(1, y)$ is initialized by multiplying the transition probability of y following the start symbol times the emission probability of the first input symbol. Given all the $\mathrm{v}(t, y)$ at a fixed t, we can calculate $\mathrm{v}(t, y)$ by looking for the most probable predecessor state y':

$$\begin{aligned}
\mathrm{v}(t, y) &= p(\hat{\mathbf{y}}_{\mathbf{0}}^{\mathbf{t}}) \\
\hat{\mathbf{y}}_{\mathbf{0}}^{\mathbf{t}} &= \text{ best path from } 0 \text{ to } t \text{ ending in state } y \\
\mathrm{v}(1, y) &= p(y|\,\mathrm{start}) p(x_1|y) \\
\mathrm{v}(t+1, y) &= \max_{y'} \left[\mathrm{v}(t, y') \cdot p(y|y') \cdot p(x_{t+1}|y)\right]
\end{aligned}$$

The probability of the most probable path is then given as $\mathrm{v}(T+1, \mathrm{stop})$. In order to find the most probable output sequence we define a similar matrix holding the most probable predecessor of each state:

$$\begin{aligned}
\mathrm{bt}(t, y) &= \hat{\mathbf{y}}_{\mathbf{0}}^{\mathbf{t}} \\
\mathrm{bt}(0, y) &= \mathrm{start} \\
\mathrm{bt}(t+1, y) &= \arg\max_{y'} \left[\mathrm{v}(t, y') \cdot p(y|y') \cdot p(x_{t+1}|y)\right]
\end{aligned}$$

The most probable state sequence can then be reconstructed using back-tracking starting at the state $\mathrm{bt}(T+1, \mathrm{stop})$. The algorithm has a runtime of $\mathcal{O}(N^2 T)$ as we have to test N^2 possible transitions at every position. The algorithm makes a first-order Markov assumption, but can be easily extended to higher orders, as every Markov chain can be reduced to a first-order Markov chain: We can implement a second-order HMM by using sequence bigrams as underlying states. A second-order model created this way would have N^4 transitions at any position of which only N^3 are consistent (overlapping). A general n^{th} order HMM can thus be decoded at a time complexity of $\mathcal{O}(N^{n+1} T)$.

2.3.2 Hidden Markov Models with Latent Annotations

Hidden Markov models with latent annotations (HMM-LA) were introduced by Huang et al. (2009). They are an adaptation of a similar algorithm developed for probabilistic context-free grammars (Petrov et al., 2006). We already discussed that the Markov assumption made during the training has an important effect on the runtime of the Viterbi algorithm. HMM-LAs soften the Markov assumptions of traditional HMMs by increasing the number of states. The approach is thus similar to increasing the order of the HMM, but differs in that the states are trained in a way to maximize the likelihood of the training data. The procedure consists of two iteratively executed steps. The first step is called "splitting" and splits every HMM state into two similar substates. Then the expectation–maximization (EM) algorithm is applied to optimize the parameters of the new model. In the second step, called "merging" the improvement in likelihood that every single split provides is estimated and a certain fraction of the splits with the lowest gains are reversed. The two steps are then iterated until a certain number of states is reached. As an example from part-of-speech tagging: It might be beneficial to split the PoS tag "Noun" into common nouns and proper nouns in the first iteration and in the second iteration proper nouns could be split into company names (often followed by *corp.* or *inc.*) and person names (often proceeded by *Mr*). On the other hand it makes little sense to have a high number of determiner tags so we would expect that the determiner splits get reversed in the merging step.

EM

The EM algorithm estimates models over unobserved latent variables (Dempster et al., 1977). It was originally proposed as a method of handling incomplete data sets. The algorithm finds a local optimum in the marginal likelihood:

$$\mathrm{LL}_\theta(D) = \sum_{x \in D} p(x|D) = \sum_{x \in D} \sum_{z} p(x, z|\theta) \tag{2.32}$$

where D is a data set of observed items x, the z are the values of the unobserved latent variables and θ is a model. EM iteratively improves θ: In the expectation step, it calculates estimated counts using the model:

$$\hat{c}_{D,\theta_t}(x, z) = c_D(x) \cdot p(z|x, \theta_t) = c_D(x) \cdot \frac{p(x, z|\theta_t)}{\sum_{z'} p(x, z'|\theta_t)} \tag{2.33}$$

From which a new model can be learned in the maximization step:

$$\theta_{t+1} = \arg\max_\theta \sum_{x,z} \hat{c}_{D,\theta_t}(x, z) \cdot p(x, z|\theta) \tag{2.34}$$

It can be shown that this procedure produces models with monotonically increasing likelihood (Dempster et al., 1977). As we already discussed, EM is used in the training of HMM-LAs, where we observe input and output symbols and need to estimate the frequencies of the unobserved substates. In this case $p(z|x, \theta_t)$ cannot be computer by summation over all possible

sequences z as there are exponentially many. We thus use a more efficient dynamic program similar to the Viterbi algorithm.

Forward-Backward

The forward-backward algorithm allows us to calculate the posterior probabilities needed for the EM e-step. Given an input sequence, the forward-backward algorithm calculates the posterior probability of a transition from z to z' at position t. This posterior can be decomposed into a product of three probabilities: The probability of all output sequences ending in z at position t, the probability of a transition from z to z' and the probability of all output sequences starting with z' at position $t+1$. While the transition probability is a simple model parameter, the other two probabilities have to be calculating using dynamic programs. We start with the probability of all sequences ending in z at position t, the corresponding program is known as the forward algorithm and is similar to the Viterbi v matrix except that the paths are combined by summation instead of maximization:

$$\begin{aligned}\alpha_{0,\text{start}} &= 1\\ \alpha_{z,t} &= \sum_{z'} \alpha_{t-1,z'} \cdot p(z|z') \cdot p(x_t|z)\end{aligned}$$

where for the sake of simplicity we ignore the dependencies on θ, $\mathbf{x}$ and $\mathbf{z}$ in the notation. The probability of all sequences starting at position t in symbol z' can be calculated similarly:

$$\begin{aligned}\beta_{T+1,\text{stop}} &= 1\\ \beta_{z,t} &= \sum_{z'} \beta_{t+1,z'} \cdot p(z'|z) \cdot p(x_{t+1}|z)\end{aligned}$$

The probability of a transition at position t is then given by:

$$p(\mathbf{x}, z, z'|t) = \alpha_{t,z} \cdot p(z'|z) \cdot \beta_{z',t+1} \tag{2.35}$$

Normalizing by the sum off all sequences, we obtain:

$$p(z, z'|\mathbf{x}, t) = \frac{\alpha_{t,z} \cdot p(z'|z) \cdot \beta_{z',t+1}}{\alpha_{T+1,\text{stop}}} \tag{2.36}$$

Just as for Viterbi the time complexity of the FB algorithm is dominated by the summation over all possible transitions at a specific position and thus grows polynomially with the number of states and exponentially with the model order. The posterior probability then allows us to estimate the frequencies needed for HMM training: The frequency of a state, following another state and of a state co-occurring with a specific input symbol:

$$\hat{c}_{D,\theta_t}(z, z') = \sum_{\mathbf{x},t} c_D(\mathbf{x}) \cdot p(z, z'|\mathbf{x}, t)$$
$$\hat{c}_{D,\theta_t}(z, x) = \sum_{\mathbf{x},z',t} c_D(\mathbf{x}) \cdot p(z, z'|\mathbf{x}, t)$$

Here we derived the forward-backward computation for an unrestricted HMM. In the case of HMM-LA training we already know the correct output symbols and just need to calculate the probabilities of the possible substates. We thus calculate probabilities of the following form: $p(z_y, z_{y'}|\mathbf{x}, t, y, y')$. This can be easily achieved by updating the forward and backward definitions:

$$\alpha'_{z_y,t} = \sum_{z'_y \in \Omega(y)} \alpha'_{t-1,z'_y} \cdot p(z_y|z'_y) \cdot p(x_t|z_y) \tag{2.37}$$

where $\Omega(y)$ is the set of all substates of y. With EM as the method to adjust the latent substates to the training set, we can continue with our description of split-merge training for HMMs. The procedure starts by collecting frequency counts for a bigram HMM from an annotated corpus. We denote the transition frequency of state y' following y by $c_{y,y'}$ and the emission frequency of symbol x occurring with state y by $c_{y,x}$. The training procedure consists of iteratively splitting tag symbols into two latent subsymbols and adjusting the resulting latent model to the training set using expectation–maximization (EM) training. It then approximates the gain in likelihood (L) of every split and reverts splits that give little increase and needlessly increase the complexity of the model.

Splitting

In the split phase we split every state y into two subtags y_0 and y_1. We set

$$c_{y_0,x} = \frac{c_{y,x}}{2} + r$$
$$c_{y_1,x} = \frac{c_{y,x}}{2} - r$$

where r is a random number $r \in [-\rho c_{y,x}, \rho c_{y,x}]$ and $\rho \in [0, 1]$ controls how much the statistics for y_0 and y_1 differ. The exact value of ρ is of secondary importance, if it is just big enough to break the symmetry (the model could not learn anything if the parameters for y_0 and y_1 were identical). Analogously to the emission frequencies we initialize the transition frequencies as follows:

$$c_{y_0,y'_0} = \frac{c_{y,y'}}{4} + r \quad c_{y_0,y'_1} = \frac{c_{y,y'}}{4} - r$$
$$c_{y_1,y'_0} = \frac{c_{y,y'}}{4} + r' \quad c_{y_1,y'_1} = \frac{c_{y,y'}}{4} - r'$$

We then run EM training to fit the new model to the training data.

Merging

To prevent the model from wasting parameters for splits that only yield small increases in likelihood we revert the splits with the lowest gains. The gain of a split can be calculated analogously to Petrov et al. (2006). To this end we calculate for each split, what we would lose in terms of likelihood if we were to reverse this split. The likelihood contribution at a certain position t with state y can be recovered from the FB probabilities:

$$p(\mathbf{x}, \mathbf{y}|t) = \sum_{z_y \in \Omega(y)} \frac{\alpha_{t,z_y} \cdot \beta_{t,z_y}}{p(x|z_y)}$$

In order to calculate the likelihood of a model that does not split y into y_0 and y_1 we first have to derive the parameters of the new model from the parameters of the old model:

$$\begin{aligned} p(x|y) &= \sum_{i \in \{0,1\}} p_i \cdot p(x|y_i) \\ p(y'|y) &= \sum_{i \in \{0,1\}} p_i \cdot p(y'|y_i) \\ p(y|y') &= \sum_{i \in \{0,1\}} p(y_i|y') \end{aligned}$$

where p_0 and p_1 are the relative frequency of y_0 and y_1 given their parent tag y. We can now derive the approximated forward and backward probabilities by substituting the corresponding parameters. Huang et al. (2009) do not provide the equations or derivation for merging in their paper, but due to personal communication we think that their approximation has the same form:

$$\begin{aligned} \alpha'_{t,y} &\approx p(x_t|y) \sum_{i \in \{0,1\}} \alpha_{t,y_i} / p(x_t|y_i) \\ \beta'_{t,y} &\approx \sum_{i \in \{0,1\}} p_i \cdot \beta_{t,y_i} \end{aligned}$$

where the α values can be added because they do not contain terms that directly depend on t, while the other terms have to be interpolated. This calculation is approximate because it ignores any influence that the merge might have at any other position in the sequence. The likelihood of the new model can then by calculated from the new FB values and a fraction of the states with the lowest likelihood improvements is reversed.

2.3.3 Conditional Random Fields

Conditional random fields are the state-of-the-art sequence models for many NLP processing tasks such as part-of-speech tagging and named entity recognition. In this section we discuss the

formal implementation. We start with a discussion of maximum entropy (ME) models of which conditional random fields are a special case. The general idea behind ME models is to find a principled way to model the distribution of a random variable X given a data set D. Unlike in the derivations of n-gram models and HMMs we do not want to make explicit independence assumptions. The only assumptions we make are introduced via feature functions $\phi_i(x)$ which tell us which aspects of the problem are important. In the case of language modeling for example, where X was the set of all sequences of words of a vocabulary V, we could decide that the ϕ_i count how often certain n-grams occurs in the sentence x. We require that the expected values of these ϕ_i in the density function f of our model equal the expected values of the empirical distribution P_D:

$$\mathbb{E}_f(\phi_i) = \mathbb{E}_{P_D}(\phi_i) \ , 1 \leq i \leq N \tag{2.38}$$

This essentially guarantees that the model memorizes the important aspects of our data set. We now want to set the parameters of our model without making any further assumptions by maximizing the entropy of the model, where we define entropy as the expected Shannon information $I(x)$ of a discrete random variable X[1]:

$$H(X) = \mathbb{E}_X(I) = \sum_{x \in \mathcal{X}} P(x) I(x) = -\sum_{x \in \mathcal{X}} P(x) \log P(x) \tag{2.39}$$

A ME model without constraints corresponds to the uniform probability distribution. If we add a constraint to the model, the model will change in order to satisfy the constraint, but still stay as *uniform* as possible. Consider the following example: We are modeling a discrete random variable with the three values A, B, and C. If we know nothing more then we should assume a uniform distribution with $P(A) = P(B) = P(C) = 1/3$. If we now add the constraint that A and B should make up 50% of our probability mass then the natural change would be to assume $P(A) = P(B) = 1/4$, where we set the probabilities of A and B uniform because we do not want to assume anything.

We now need to find the form of the density function $f(x)$. We do so by solving an optimization problem. We want to define $f(x)$ in a way to maximize the ME and satisfying the features constraints and the constraint that $f(x)$ is properly normalized. Using Lagrange multipliers we arrive at the following updated objective function:

$$\begin{aligned} H'(f, \lambda) &= H(f, \lambda) + \lambda_0 (\sum_{x' \in \mathcal{X}} f(x') - 1) + \sum_{i=1}^{N} \lambda_i [\mathbb{E}_f(\phi_i) - \mathbb{E}_{p_D}(\phi_i)] \\ &= -\sum_{x' \in \mathcal{X}} f(x') \log f(x') + \lambda_0 (\sum_{x' \in \mathcal{X}} f(x') - 1) + \sum_{i=1}^{N} \lambda_i [\mathbb{E}_f(\phi_i) - \mathbb{E}_{p_D}(\phi_i)] \end{aligned}$$

[1] By convention we have $\lim_{x \to 0} x \log x = 0$

H' has the following derivatives:

$$\frac{\partial H'(f,\lambda)}{\partial f(x)} = -\log f(x) - 1 + \lambda_0 + \sum_{i=1}^{N} \lambda_i \phi_i(x) \tag{2.40}$$

$$\frac{\partial H'(f,\lambda)}{\partial \lambda_0} = \sum_{x' \in \mathcal{X}} f(x') - 1 \tag{2.41}$$

$$\frac{\partial H'(f,\lambda)}{\partial \lambda_i} = \mathbb{E}_f(\phi_i) - \mathbb{E}_{p_D}(\phi_i) \tag{2.42}$$

Setting Eq. 2.40 and Eq. 2.41 zero we obtain:

$$f(x,\lambda) = \frac{\exp \sum_{i=1}^{N} \lambda_i \phi_i(x)}{Z(\lambda)} \tag{2.43}$$

$$Z(\lambda) = \sum_{x \in \mathcal{X}} \exp \sum_{i=1}^{N} \lambda_i \phi_i(x) \tag{2.44}$$

Which is the general form of a ME model. The normalization constant Z is called the partition function. The remaining constraints Eq. 2.42 are met during training and it can be shown that this can also been done by maximizing the likelihood of D given the model (see Sudderth (2006) for details and a proof).

The models we are interested in for classification have the following conditional form:

$$p_{\mathrm{ME}}(y|x) = \frac{1}{Z_{\mathrm{ME}}(\vec{\lambda}, x)} \exp \sum_i \lambda_i \phi_i(x,y) \tag{2.45}$$

$$Z_{\mathrm{ME}}(\vec{\lambda}, x) = \sum_y \exp \sum_i \lambda_i \phi_i(x,y) \tag{2.46}$$

As we already discussed in the last subsection, these models can be maximized by optimizing the (conditional) likelihood of D:

$$\begin{aligned} \mathrm{ll}[D](p_{\mathrm{ME}}(\vec{\lambda})) &= \sum_{x,y \in D} \log p_{\mathrm{ME}}(y|x,\vec{\lambda}) \\ &= \sum_{x,y \in D} \sum_i \lambda_i \cdot \phi_i(x,y) - \sum_{x,y \in D} \log Z_{\mathrm{ME}}(\vec{\lambda}, x) \end{aligned} \tag{2.47}$$

$$\frac{\partial\, \mathrm{ll}[D](p_{\mathrm{ME}}(\vec{\lambda}))}{\partial \lambda_i} = \sum_{x,y \in D} \phi_i(x,y) - \sum_{x,y \in D} \sum_{y'} \phi_i(x,y') p_{ME}(y'|x,\vec{\lambda}) \tag{2.48}$$

Given $\mathrm{ll}[D](\vec{\lambda})$ and the gradient $\nabla\,\mathrm{ll}[D](\vec{\lambda})$ we can use general numeric optimization to calculate the optimal $\hat{\lambda}$. Here we just give a simple stochastic gradient descent algorithm used by Tsuruoka et al. (2009). The algorithm receives an initial step width η_0 and a maximal number of iterations N. It then iterates over the data in an online fashion and moves the model parameters in the direction of the gradient. The step width η decays with the number of processed items in order to achieve convergence:

Algorithm 2.2 Stochastic Gradient Descent

$\vec{\lambda} \leftarrow \vec{0}$
$i \leftarrow 0$
for epoch $= 1 \rightarrow N$ **do**
  shuffle D
  for $\vec{x}, \vec{y} \in D$ **do**
    $\eta_i \leftarrow \frac{\eta_0}{1+i/|D|}$
    $\vec{\lambda} \leftarrow \vec{\lambda} + \eta_i \cdot \nabla_{\vec{\lambda}} \log p(\vec{y}|\vec{x})$
    $i \leftarrow i + 1$
  end for
end for

During training, a ME Classifier might tend to use rare features to explain classes that cannot be explained otherwise. These rare features might then obtain a high weight λ_i even though we cannot be sure they will behave in the same way on unseen data. We thus add a penalty term to our likelihood objective which pushes the weight vector towards zero. Every increase in a feature weight has now to be justified by an sufficient increase in training data likelihood. The most common form of regularization is to impose constraints on the norm of the weight vector. The so called l_2-regularization has the following form:

$$ll'_D(\lambda) = ll_D(\lambda) - \mu \sum_i |\lambda_i|^2 \tag{2.49}$$

$$\frac{\partial ll'_D(\lambda)}{\partial \lambda_i} = \nabla ll_D(\lambda) - 2\mu\lambda_i \tag{2.50}$$

where μ is the strength of the regularizes, a hyper parameter that needs to be optimized on held-out data. Another common form is l_1-regularization:

$$ll''_D(\lambda) = ll_D(\lambda) - \mu \sum_i |\lambda_i| \tag{2.51}$$

$$\frac{\partial ll''_D(\lambda)}{\partial \lambda_i} = \nabla ll_D(\lambda) - \mu\,\mathrm{sign}(\lambda_i) \tag{2.52}$$

The difference between the two forms of regularization is that l_2 generates small λ weights while l_1 will set the less important features to zero. l_1-regularization is thus also useful as a feature selection method.

Conditional random fields are ME classifiers over sequences and can be defined in the following way:

$$p_{\mathrm{CRF}}(\mathbf{y}|\mathbf{x}) = \frac{\exp \sum_t \sum_i \lambda_i \cdot \phi_i(y_t, y_{t-1}, \mathbf{x}, t)}{Z_{\mathrm{CRF}}(\vec{\lambda}, \mathbf{x})} \tag{2.53}$$

$$Z_{\mathrm{CRF}}(\vec{\lambda}, \mathbf{x}) = \sum_{\mathbf{y}} \exp \sum_t \sum_i \lambda_i \cdot \phi_i(y_t, y_{t-1}, \mathbf{x}, t) \tag{2.54}$$

The only difference between MEs and CRFs is that we need to define some dynamic programs to make the calculations needed during parameter estimation efficient. We start with the partition function Z_{CRF} which involves a summation over all the possible sequences $\mathbf{y}$. As explained in the definition of the Viterbi algorithm the numbers of sequences rises exponentially with the number of output symbols. However, we have already discussed that the sum of all the paths through a sequence lattice can be efficiently calculated using the forward or backward algorithm:

$$\begin{aligned}
\log Z_{CRF}(\vec{\lambda}, \mathbf{x}) &= \log \sum_{\mathbf{y}} \exp \sum_t \sum_i \lambda_i \cdot \phi_i(y_t, y_{t-1}, \mathbf{x}, t) \\
&= \log \sum_{\mathbf{y}} \exp \sum_t \psi(y_t, y_{t-1}, \mathbf{x}, t) \\
&= \sum_{\mathbf{y}}^{\oplus} \sum_t \psi(y_t, y_{t-1}, \mathbf{x}, t) \\
&= \alpha(T+1, \mathrm{stop}) = \beta(0, \mathrm{start})
\end{aligned}$$

where $\oplus$ denotes the addition of two numbers in log space: $a \oplus b = \log(\exp a + \exp b)$, $\psi(y_t, y_{t-1}, \mathbf{x}, t) = \sum_i \lambda_i \cdot \phi_i(y_t, y_{t-1}, \mathbf{x}, t)$ is the score potential around position t and we define α and β – following our discussion of FB for HMM– as:

$$\begin{aligned}
\alpha(0, \mathrm{start}) &= 0 \\
\alpha(t, y) &= \sum_{y'}^{\oplus} \alpha(t-1, y') + \psi(y', y, \mathbf{x}, t)
\end{aligned}$$

$$\begin{aligned}
\beta(T+1, \mathrm{stop}) &= 0 \\
\beta(t, y) &= \sum_{y'}^{\oplus} \beta(t+1, y') + \psi(y', y, \mathbf{x}, t)
\end{aligned}$$

The difference to our definition HMM is that we now do not calculate probabilities, but unnormalized log-probabilities. In order to estimate parameters we just have to derive $\mathrm{ll}_D(\vec{\lambda})$ and $\nabla \mathrm{ll}_D(\vec{\lambda})$:

$$\begin{aligned}
\mathrm{ll}_D(\vec{\lambda}) &= \sum_{\mathbf{x},\mathbf{y}\in D} \log p_{\mathrm{CRF}}(\mathbf{y}|\mathbf{x},\vec{\lambda}) \\
&= \sum_{\mathbf{x},\mathbf{y}\in D} \sum_t \sum_i \lambda_i \cdot \phi_i(y_t, y_{t-1}, \mathbf{x}, t) - \sum_{\mathbf{x},\mathbf{y}\in D} \log Z_{\mathrm{CRF}}(\vec{\lambda},\mathbf{x}) \\
\frac{\partial \mathrm{ll}_D(\vec{\lambda})}{\partial \lambda_i} &= \sum_{\mathbf{x},\mathbf{y}\in D} \sum_t \phi_i(y_t, y_{t-1}, \mathbf{x}, t) - \sum_{\mathbf{x},\mathbf{y}\in D} \sum_{t,\mathbf{y}'} \phi_i(y'_t, y'_{t-1}, \mathbf{x}, t) \cdot p_{CRF}(\mathbf{y}'|\mathbf{x},\vec{\lambda}) \\
&= \sum_{\mathbf{x},\mathbf{y}\in D} \sum_t \phi_i(y_t, y_{t-1}, \mathbf{x}, t) - \sum_{\mathbf{x},\mathbf{y}\in D} \sum_{t,y',y''} \phi_i(y', y'', \mathbf{x}, t) \cdot p_{\mathrm{CRF}}(y', y''|\mathbf{x}, t, \vec{\lambda}) \quad (2.55)
\end{aligned}$$

where $p_{\mathrm{CRF}}(y', y''|\mathbf{x}, t, \vec{\lambda})$ denotes the posterior probability of a transition from y' to y'' at position t. We already discussed the calculation of this probability for the HMM case and can derive a similar formula using FB:

$$\begin{aligned}
p_{\mathrm{CRF}}(y, y'|\mathbf{x}, t, \vec{\lambda}) &= \frac{\sum\{\psi(\mathbf{y})|\mathbf{y} : \text{path with a } y \to y' \text{ transition at } t\}}{\sum_{\mathbf{y}} \psi(\mathbf{y})} \\
&= \frac{\exp(\alpha(t, y) + \psi(y', y, \mathbf{x}, t) + \beta(t+1, y'))}{Z_{CRF}(\vec{\lambda},\mathbf{x})}
\end{aligned}$$

The log-likelihood can then be optimized using the SGD algorithm we discussed for ME classifiers. The runtime of the parameter estimation is dominated by the FB calculations above and thus in $\mathcal{O}(N^n T)$ where N denotes the number of output states, T the length of the sequence and n the order of the CRF. CRF training is thus slow when high model orders (> 1) or big tagsets are used (> 100). In Chapter 5 we discuss pruning strategies that yield substantial speed ups in these particular cases.

2.4 Word Representations

Word Representations describe the morphologic, syntactic or semantic properties of word forms by mapping them to a small number of nominal values (called **clusters**) or by embedding them in a vector space. In this thesis we are going to discuss two different types of representations. Morphological representations are based on the morphological properties of a word and might also consider special properties of the spelling of a word, such as its capitalization or whether it contains special symbols such as digits or hyphens. Morphological representations developed for language modeling of morphologically complex languages are discussed in Chapter 3. In this section we discuss distributional word representations which are extracted from unlabeled text.

Distributional word representations are motivated by the distributional hypothesis: 'A word is characterized by the company it keeps' (Harris, 1954; Firth, 1957).

In particular, we discuss count vectors reduced by a singular value decomposition (SVD), word clusters induced using the likelihood of a class-based language model, distributed embeddings trained using a neural network and accumulated tag counts, a task-specific representation obtained from an automatically tagged corpus. All representations are trained from unlabeled raw text, except accumulated tag counts which also need automatically assigned morphological tags.

Singular Value Decomposition

The singular value decomposition (SVD) of word-feature cooccurrence matrices (Schütze, 1995) has been found to be a fast and efficient way to obtain distributed embeddings. The approach selects a subset of the vocabulary as so-called feature words, usually by including words up to a certain frequency rank f. Every word form can then be represented by the accumulated counts of feature words occurring to its left and right. In this work we mark whether a word occurred on the left and right as we are more interested in syntactic than semantic properties. Every word is thus represented by a count vector of dimensionality $2f$. These count vectors could in principle already be considered as embedding, but they have a number of properties that make them inadequate for many NLP tasks. First of all they have a high dimensionality and are sparse as most word forms only occur a couple of times. Secondly, their dimensions are correlated: A word that occurs after *a* can also occurs after *the*, while we believe that the vectors contain information that is useful for many tasks we do need a procedure to compress them and reduce the correlation between the different dimensions. Singular value decomposition turns out to be a natural choice for the problem. A SVD is a factorization of the form:

$$M = U\Sigma V^T \tag{2.56}$$

where Σ is a diagonal matrix and its diagonal entries σ_i are called **singular values**. If $M \in \mathcal{R}^{n\times m}$ we have $U \in \mathcal{R}^{n\times n}$, $\Sigma \in \mathcal{R}^{n\times m}$ and $V \in \mathcal{R}^{m\times m}$. The SVD can be used to find the low-rank approximation M_r to M minimizing the Frobenius norm of the difference Matrix $M - M_r$:

$$\|M\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|m_{ij}|^2} \tag{2.57}$$

In order to find the optimal low-rank approximation of rank r. We set the approximated matrix M_r to:

$$M_r = U\Sigma_r V^T \tag{2.58}$$

where Σ_r is derived from Σ by setting all, but the highest r singular values to zero. In our application M is the matrix holding the coocurrence counts of word forms and their left and right features and thus $M \in \mathcal{R}^{|V|\times 2\cdot f}$ where V denotes the vocabulary. We can derive dense vectors of dimensionality r by using the SVD to reduce the rank of M to r. The resulting representation

will still hold similar information as M as they are derived minimizing the distance in terms of the Frobenius norm, but they will also be dense and less correlated. SVD-based representations have been used in English POS induction (Lamar et al., 2010) as well as as features in English POS tagging and syntactic chunking (Huang et al., 2009).

Language model-based

LM-based word clusters were introduced by Brown et al. (1992a) and later found to be helpful in a range of NLP tasks. The basic idea is to find the optimal clustering with respect to the likelihood of a class-based language model:

$$g = \arg\max_{g} \prod_{i=1}^{|D|} p(g(x_i)|g(x_{i-1})) \cdot p(x_i|g(x_i)) \tag{2.59}$$

where $g(x)$ is the cluster assignment function, that maps a word form x to a cluster and $|D|$ denotes the length of the training set. As finding the optimal clustering requires an exponential algorithm the problem is approximated by greedy algorithms. Brown et al. (1992a) propose a bottom-up algorithm that merges the pair of clusters that yields the smallest loss in likelihood. Even after some optimization, the algorithm has a cubic runtime ($\mathcal{O}(|V|^3)$). They also proposed a more efficient approximation of that algorithm that limits the number of clusters under consideration and still works well in practice. This algorithm has a runtime of $\mathcal{O}(|V|C^2)$, where C is the number of cluster to be induced. This algorithm is used by most work in the literature (Liang, 2005; Turian et al., 2010; Koo et al., 2008).

Miller et al. (2004) use tags of different granularity induced from unlabeled text to improve the performance of an averaged perceptron tagger Collins (2002) on an English NER task. The Brown algorithm induces a tree where leaves represent a single word form and the root node the entire vocabulary. Intermediate nodes represent clusters of different sizes and can be addressed by a binary string specifying the path from the root node to the cluster. Efficient open-source implementations of this algorithm have been released by Liang (2005) and Klusácek (2006).[2,3]

Miller et al. (2004) report an error reduction of 25% using path prefixes in combination with active learning. Brown clusters are also used by Koo et al. (2008) to improve dependency parsing for English and Czech. They created new templates by replacing word forms and POS in the feature templates of the baseline parser and found that short paths (4 - 6 bits) were good replacements for POS and longer paths (assigning the word form to one out of 1000 clusters) were a good replacement for word forms. They report improvements in attachments score of > 1.0. Chrupala (2011) compare Brown clusters to a Latent Dirichlet Allocation (LDA) model on Spanish and French morphological tagging and find them to yield similar performance.

Martin et al. (1998) proposed a different induction algorithm similar to K-means clustering in that it starts with an initial clustering and greedily improves the objective function by moving single words to their optimal cluster. In contrast to K-means, it updates the objective function

[2] https://github.com/percyliang/brown-cluster/
[3] https://ufal.mff.cuni.cz/tools/mmic

immediately. The algorithm has a runtime of $\mathcal{O}(|V|NC)$, where N is the number of iterations needed until convergence ($N \ll C$). This algorithm has also been shown to work well in unsupervised POS induction (Blunsom and Cohn, 2011; Clark, 2003). Our implementation of this algorithm is discussed in Appendix B.

Neural Networks

Neural networks have been used by Collobert et al. (2011) to train embeddings for POS tagging as well as other NLP tasks. These embeddings – henceforth *CW embeddings* – are trained by building a neural network that given the context of a word as input is trained to discriminate between the correct center word and a random word.

In the architecture of the network, word forms x in a window around the target position are first mapped to embedding vectors by means of a lookup table W. The embeddings are then concatenated and transformed using a weight matrix M. In order to introduce non-linearity into the model $\tanh$ is applied to the intermediate output. Finally, the resulting vector is multiplied with a weight vector v to produce a score $f_\theta(x)$ (with $\theta = (W, M, v)$) which denotes how well the target word fits the surrounding context. Using a text corpus D with a vocabulary V, the model is trained by optimizing a ranking criterion of the following form:

$$\hat{\theta} = \arg\min_{\theta} = \sum_{x \in D} \sum_{w \in V} \max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\} \tag{2.60}$$

where $x^{(w)}$ is created from x by replacing the center word with w. The criterion effectively sets the parameters θ in a way that the correct $f_\theta(x)$ is at least 1 point higher than for any other $x^{(w)}$. The sum over all w in the vocabulary V is expensive and usually approximated by sampling random w from V. The proposed training algorithm is reported to need several days or even weeks, but has been reimplemented by Al-Rfou et al. (2013), who induced embeddings for the Wikipedias of more than 100 languages. In an NER task, Turian et al. (2010) find the performance of Brown clusters to be competitive with the more training intensive CW embeddings. Mnih and Hinton (2008) and Mikolov et al. (2013) are two additional important Neural Network models that have been used in the literature, but are not discussed in this thesis, because of the lack of pre-trained embeddings.

Accumulated Tag Counts

Accumulated tag counts (ATC) are a form of task-specific sparse representation that was successfully applied in PCFG parsing. The unlabeled corpus is first annotated by an automatic morphological tagger; for each occurring word form, the number of times a specific tag was assigned can then be used as a representation. Goldberg and Elhadad (2013) and Szántó and Farkas (2014) show that using such information in the word-preterminal emission probabilities of PCFGs can improve parsing accuracy. In the context of morphological tagging, ATC representations are interesting because they can be interpreted as an approximated morphological

analyzer. A morphological analyzer (MA) is a manually-created finite-state transducer that produces for every word form the possible morphological readings. MAs feature a high precision, because they rely on carefully checked morphological rules and stem dictionaries. ATCs on the other hand, can produce readings for word forms, with new stems and also carry the notion of how rare or frequent a specific reading is. Szántó and Farkas (2014) find that in Hungarian PCFG parsing ACTs perform as well as morphological analyzers.

2.5 Conclusions

In this chapter we presented the foundations of the remaining chapters of this thesis. Section 2.1 introduced the linguistic terminology used throughout the entire thesis. Section 2.2 discussed the basics of n-gram language modeling and thus prepared the reader for the next Chapter 3, where we describe the design of a morphological language model. In Section 2.3, we reviewed hidden Markov models with latent annotations (HMM-LA). In Chapter 4, HMM-LAs are evaluated as a method to induce linguistically meaningful tagsets, which can be used to improve dependency parser performance. Section 2.3 also introduced conditional random fields (CRF). In Chapter 5, we present a pruned CRF for fast morphological tagging with higher model orders. Section 2.4 introduced several word representations. In Chapter 6 we evaluate their utility in morphological tagging.

Chapter 3

A Morphological Language Model

Erklärung nach §8 Absatz 4 der Promotionsordnung: This chapter covers work already published at international peer-reviewed conferences. The relevant publications are Müller and Schütze (2011) and Müller et al. (2012). The research described in this chapter was carried out in its entirety by myself. The other author(s) of the publication(s) acted as advisor(s) or were responsible for work that was reported in the publication(s), but is not included in this chapter.

Morphologically rich languages (MRLs) pose a problem to traditional language modeling, because their productive morphology introduces additional sparsity that makes n-gram estimation challenging. In this chapter, we present a morphological language model that mitigates these sparsity issues by coupling word forms of similar morphology. In particular, we propose a class-based language model that clusters rare words of similar morphology together. The model combines morphological and shape features with a Kneser-Ney model. We discuss a large cross-lingual study of European languages, where even though the model is generic and the same architecture and features are used for all languages, the model achieves reductions in perplexity for all 21 languages represented in the Europarl corpus, ranging from 3% to 11%. We will also find that almost all of this perplexity reduction can be achieved by identifying suffixes by frequency.

3.1 Introduction

Language models are fundamental to many natural language processing applications. In the most common approach, language models estimate the probability of the next word based on one or more equivalence classes that the history of preceding words is a member of. The inherent productivity of natural language poses a problem in this regard because the history may be rare or unseen or have unusual properties that make assignment to a predictive equivalence class difficult.

In many languages, morphology is a key source of productivity that gives rise to rare and unseen histories. For example, even if a model can learn that words like “large”, “dangerous” and “serious” are likely to occur after the relatively frequent history “potentially”, this knowledge

cannot be transferred to the rare history "hypothetically" without some generalization mechanism like morphological analysis. A second challenge are words that appear in the recognition task at hand, but not in the training set, so called out-of-vocabulary (OOV) words. Especially for productive languages it is often necessary to at least reduce the number of OOVs. Both challenges are even more severe for MRLs, where a single lexeme might occur as one of dozens or hundreds of word forms.

Our primary goal is not to develop optimized language models for individual languages. Instead, we investigate whether a simple generic language model that uses shape and morphological features can be made to work well across a large number of languages.

We also do not want to create a model with a runtime substantially worse than that of a standard word-based n-gram model. Language models are estimated from raw text, which thanks to the web and the digitization of books is available in huge quantities for most languages. This leads to the point, where the amount of training data used is not limited by the text available, but by the training algorithm and by computational resources.

In an extensive evaluation we find our model to achieve considerable perplexity reductions for all 21 languages in the Europarl corpus. We see this as evidence that morphological language modeling should be considered as a standard part of any language model, even for languages like English that are often not viewed as a good application of morphological modeling due to their morphological simplicity.

To understand which factors are important for good performance of the morphological component of a language model, we perform an extensive cross-lingual analysis of our experimental results. We look at three parameters of the morphological model we propose: the frequency threshold θ that divides words subject to morphological clustering from those that are not; the number of suffixes used ϕ; and three different morphological segmentation algorithms. We also investigate the differential effect of morphological language modeling on different word shapes: alphabetical words, punctuation, numbers and other shapes.

Some prior work has used morphological models that require careful linguistic analysis and language-dependent adaptation. We show that simple frequency analysis performs only slightly worse than more sophisticated morphological analysis. This potentially removes a hurdle to using morphological models in cases where sufficient resources to do the extra work required for sophisticated morphological analysis are not available.

The motivation for using morphology in language modeling is similar to distributional clustering (Brown et al., 1992a). In both cases, we form equivalence classes of words with similar distributional behavior. In a preliminary experiment, we find that morphological equivalence classes reduce perplexity as much as traditional distributional classes – a surprising result (Table 3.4).

The main contributions are as follows: We present a language model design and a set of morphological and shape features that achieve reductions in perplexity for all 21 languages represented in the Europarl corpus, ranging from 3% to 11%, compared to a Kneser-Ney model. We show that identifying suffixes by frequency is sufficient for getting almost all of this perplexity reduction. More sophisticated morphological segmentation methods do not further increase perplexity or just slightly. Finally, we show that there is one parameter that must be tuned for good performance for most languages: the frequency threshold θ above which a word is not subject

to morphological generalization because it occurs frequently enough for standard word n-gram language models to use it effectively for prediction.

The chapter is organized as follows. In Section 3.2 we discuss related work. In Section 3.3 we describe the morphological and shape features. Section 3.4 introduces language model and experimental setup. Section 3.5 discusses our results. Section 3.6 summarizes the contributions of this chapter.

3.2 Related Work

Whittaker and Woodland (2000) apply language modeling to morpheme sequences and investigate data-driven segmentation methods. Creutz et al. (2007) propose a similar method that improves speech recognition for highly inflecting languages. They use Morfessor CAT-MAP (Creutz and Lagus, 2005) to split words into morphemes. Both approaches are essentially a simple form of a factored language model (FLM) (Bilmes and Kirchhoff, 2003). In a general FLM a number of different back-off paths are combined by a back-off function to improve the prediction after rare or unseen histories. Vergyri et al. (2004) apply FLMs and morphological features to Arabic speech recognition. These papers and other prior work on using morphology in language modeling have been language-specific and have paid less attention to the question as to how morphology can be useful across languages and what generic methods are appropriate for this goal. Previous work also has concentrated on traditional linguistic morphology whereas we compare linguistically motivated morphological segmentation with frequency-based segmentation and include shape features in our study.

Our initial plan was to use complex language modeling frameworks that allow experimenters to include arbitrary features (including morphological and shape features) in the model. In particular, we looked at publicly available implementations of maximum entropy models (Rosenfeld, 1996; Berger et al., 1996) and random forests (Xu and Jelinek, 2007). However, we found that these methods do not currently scale to running a large set of experiments on a multi-gigabyte parallel corpus of 21 languages. Similar considerations apply to other sophisticated language modeling techniques like Pitman-Yor processes (Teh, 2006), recurrent neural networks (Mikolov et al., 2010) and FLMs in their general, more powerful form.

We therefore decided to conduct our study in the framework of smoothed n-gram models, which currently are an order of magnitude faster and more scalable. More specifically, we adopt a class-based approach, where words are clustered based on morphological and shape features. This approach has the nice property that the number of features used to estimate the classes does not influence the time needed to train the class-based language model, once the classes have been found. This is an important consideration in the context of the questions asked in this chapter as it allows us to use large numbers of features in our experiments.

3.3 Modeling of Morphology and Shape

Our basic approach is to define a number of morphological and shape features and then assign all words with identical feature values to one class. In earlier work on English (Müller and Schütze, 2011), we tried to reduce the number of vectors using clustering, but could not find substantial improvements. An overview of our model is given in Figure 3.1: The model is trained from raw text data. During training we first extract the vocabulary. The vocabulary is used to train an unsupervised segmentation algorithm that provides us with the most frequent suffixes of the language. We also you use the vocabulary to extract a number of shape-based features. The concatenation of the shape features of a word and its suffix define its morphological class. The morphological classes are used to train a class-based morphological language model, which is then interpolated with a word-based language model.

We represent words by a feature vector consisting of three parts that represent information about *suffixes*, *capitalization* and *special characters*. For the *suffixes* group, we define a binary feature for each of the ϕ most frequent suffixes. Suffixes are learned in an unsupervised fashion using one of the segmentation algorithms described below. One additional binary feature is used for all other suffixes, including the empty suffix.

In addition to suffix features, we define features that capture shape properties: capitalization and special characters. These groups are motivated by the analysis shown in Table 3.1: One important goal of our model is to improve OOV modeling. The table shows that most OOV words ($f = 0$) are names or nouns. This distribution is similar to hapax legomena ($f = 1$), but different from the POS distribution of all tokens.

	POS	Types		Tokens
		$f = 0$ (OOV)	$f = 1$	
English	Proper Noun	0.61	0.52	0.15
	Noun	0.23	0.24	0.17
	Adjective	0.08	0.10	0.06
	Verb	0.03	0.08	0.11
	Σ	0.95	0.94	0.49
Hungarian	Noun	0.42	0.46	0.22
	Proper Noun	0.34	0.24	0.10
	Adjective	0.11	0.13	0.11
	Verb	0.05	0.08	0.08
	Σ	0.92	0.91	0.51
Spanish	Proper Noun	0.68	0.44	0.08
	Noun	0.14	0.22	0.17
	Verb	0.08	0.19	0.10
	Adjective	0.06	0.11	0.05
	Σ	0.98	0.97	0.43

Table 3.1: Proportion of dominant POS for types with training set frequencies $f \in \{0, 1\}$ and for tokens for a Wikipedia Corpus.

As proper nouns often do not have meaningful suffixes we add additional features to handle them. Capitalization and special character features are of obvious utility in identifying the POS classes of proper nouns and cardinal numbers since names are usually capitalized and sometimes contain special characters such as comma and period. To capture these "shape" properties of a word, we define the features listed in Table 3.2.

$is\_capital(w)$	first character of w is an uppercase letter
$is\_all\_capital(w)$	$\forall\, c \in w : c$ is an uppercase letter
$capital\_character(w)$	$\exists\, c \in w : c$ is an uppercase letter
$appears\_in\_lowercase(w)$	$\neg capital\_character(w) \vee w' \in \Sigma_T$
$special\_character(w)$	$\exists\, c \in w : c$ is not a letter or digit
$digit(w)$	$\exists\, c \in w : c$ is a digit
$is\_number(w)$	$w \in L([+-\epsilon][0-9]\,(([.,][0-9])\|[0-9])*)$

Table 3.2: Predicates of the capitalization and special character groups. Σ_T is the vocabulary of the training corpus T, w' is obtained from w by changing all uppercase letters to lowercase and $L(expr)$ is the language generated by the regular expression $expr$.

If a word in the test set has a combination of feature values that does not occur in the training set, then it is assigned to the class whose features are most similar. To this end the three parts of the vector (suffixes, capitalization, special characters) are weighted equally by normalizing the subvector of each subgroup to unit length. The most similar class is then defined as the class closest in the resulting vector space.

We investigate three different automatic suffix identification algorithms: REPORTS (Keshava and Pitler, 2006), MORFESSOR (Creutz and Lagus, 2005) and our own baseline algorithm FREQUENCY. The focus of our work is to evaluate the utility of these algorithms for language modeling; we do not directly evaluate the quality of the suffixes. A word is segmented by identifying the longest of the ϕ suffixes that it ends with. Thus, each word has one suffix feature if it ends with one of the ϕ suffixes and none otherwise.

REPORTS

REPORTS (Keshava and Pitler, 2006) is a segmentation algorithm developed for languages similar to English. It identifies possible suffix boundaries by looking at the probability that a character follows the preceding substring. Given a vocabulary V and a word $w = \alpha \mathrm{AB} \beta$, where α and β a substrings and A and B are characters. $\mathrm{B}\beta$ is a suffix of w, if

1. $\alpha \mathrm{A} \in V$
2. $P_f(B|\alpha A) < 1$.
3. $P_f(A|\alpha) \approx 1$.

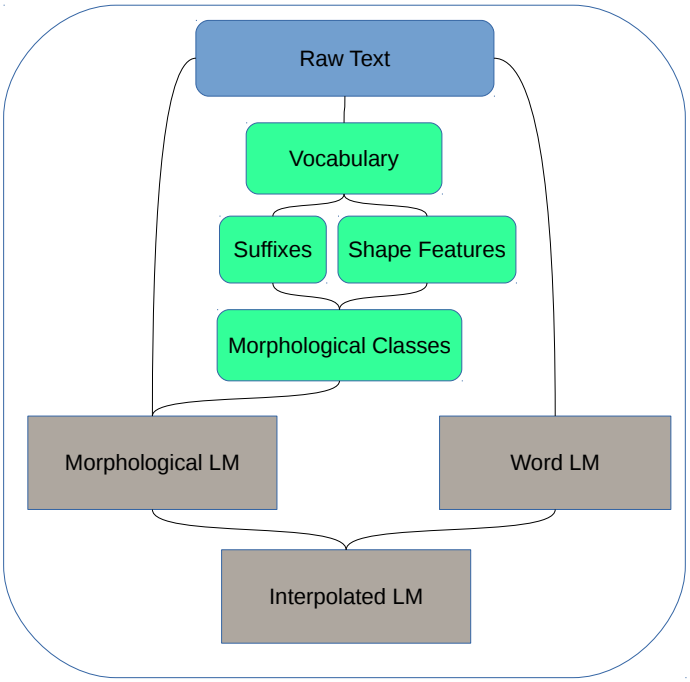

Figure 3.1: System overview of the morphological language model

where $P_f(A|\alpha)$ is the probability of a character A following the string α. P_f can be estimated from V. The first condition requires that the stem αA is a valid word of the language. The second condition states that αA can be followed by other suffixes than Bβ and the third condition assures that ABβ is not a possible suffix. As an example, given a vocabulary consisting of *work*, *works* and *working* and the target word *working*, *ing* is a valid suffix, because *work* is a valid word, $P_f(k|wor)$ equals one and $P_f(i|work)$ is smaller than one. A major drawback of the algorithm is that it cannot handle stem alternations. That is, given the German word *bücher* 'books' (singular form: *buch*) it cannot find the correct suffix *-er*, as *büch* is not a valid German word.

MORFESSOR CAT-MAP

MORFESSOR CAT-MAP (Creutz and Lagus, 2005) is an extension of Morfessor Baseline (Creutz and Lagus, 2002) and Morfessor CAT (Creutz and Lagus, 2004) and comprises ideas of previous work (de Marcken, 1995; Goldsmith, 2001). The Morfessor Baseline algorithm creates a lexicon of morphemes that can then be used to produce any word in the corpus. The lexicon is constructed in a way that frequently occurring morphemes can be reused to build rare words, it is thus essentially trying to compress the corpus. A major disadvantage of the algorithm is that it tends to oversplit rare word forms such as *s-ed-it-ious* and simply memorizes frequent words such as *having* (Creutz and Lagus, 2005). Another issue it that its notion of morpheme is type-less and morphemes might thus be used in the wrong places such as in *ed-ward*. Morfessor CAT tries to resolve these issues by representing words using Hidden Markov Models (HMM). The states

of the HMM correspond to four morpheme types: prefix, stem and suffix. The transition probabilities of the HMM are constrained so that prefixes and suffixes can only occur at the beginning or at the end of a word and need to have at least one stem state in between. The algorithm also involves a number of heuristics that deal with noisy segmentation such as the *seditious* example above. Morfessor CAT-MAP changes the Maximum-Likelihood HMM objective of Morfessor CAT into a maximum a posteriori (MAP) objective by adding a prior on the morpheme lexicon that rewards the reusage of frequent morphemes.

FREQUENCY

FREQUENCY simply selects the most frequent word-final letter sequences as suffixes. The 100 most frequent suffixes found by Frequency for English are given in the following Figure 3.2. We use the longest suffix if a word has multiple suffixes.

s, e, d, ed, n, g, ng, ing, y, t, es, r, a, l, on, er, ion, ted, ly, tion, rs, al, o, ts, ns, le, i, ation, an, ers, m, nt, ting, h, c, te, sed, ated, en, ty, ic, k, ent, st, ss, ons, se, ity, ble, ne, ce, ess, ions, us, ry, re, ies, ve, p, ate, in, tions, ia, red, able, is, ive, ness, lly, ring, ment, led, ned, tes, as, ls, ding, ling, sing, ds, ded, ian, nce, ar, ating, sm, ally, nts, de, nd, ism, or, ge, ist, ses, ning, u, king, na, el

Figure 3.2: The 100 most frequent English suffixes in Europarl, ordered by frequency.

Reports as well as Morfessor generate complete segmentations of words, but in this study we only use the algorithms to produce a list of the most frequent suffixes.

3.3.1 Morphological Class-based Language Model

Our model is a class-based language model that groups words into classes and replaces the word transition probability by a class transition probability and a word emission probability:

$$P_C(w|h = w_1^k) = P(g(w)|g(w_1)\dots g(w_k)) \cdot P(w|g(w)) \qquad \text{(Eq. 2.26 revisited)}$$

where $g(w)$ is the class of word w. Our approach specifically targets rare and unseen histories. We therefore exclude all frequent words from clustering on the assumption that enough training data is available for them. That means, that clustering of words is restricted to those below a certain token frequency threshold θ. As described above, we simply group all words with identical feature values into one class. Frequent words with a training set frequency above θ are added as singletons. The class transition probability $P(g(w_i)|g(w_{i-N+1}^{i-1}))$ is estimated using Witten-Bell smoothing. Witten-Bell smoothing outperformed modified Kneser-Ney (KN) and Good-Turing (GT) in preliminary experiments. The word emission probability is defined as follows:

$$P(w|k) = \begin{cases} 1 & , \ c(w) > \theta \\ \frac{c(w)}{\sum_{w' \in k} c(w')} - \frac{\epsilon(k)}{|k|-1}, & \theta \geq c(w) > 0 \\ \epsilon(k) & , \ c(w) = 0 \end{cases} \qquad (3.1)$$

where $k = g(w)$ is w's class, $c(w)$ is the frequency of w in the training set and $|k|$ is the number of members of c. That is, the emission probability is 1 for singletons, the class-dependent out-of-vocabulary rate for unseen forms and otherwise a discounted relative frequency. The out-of-vocabulary (OOV) rate $\epsilon(k)$ for each class k is estimated on held-out data. Our final model P_M interpolates P_C with a modified KN model:

$$P_M(w_i|w_{i-1}^{i-N+1}) = \lambda(g(w_{i-1})) \cdot P_{\mathrm{C}}(w_i|w_{i-1}^{i-N+1}) + (1 - \lambda(g(w_{i-1}))) \cdot P_{\mathrm{KN}}(w_i|w_{i-1}^{i-N+1}) \quad (3.2)$$

This model can be viewed as a generalization of the simple interpolation $\alpha P_{\mathrm{C}} + (1-\alpha)P_{\mathrm{W}}$ used by Brown et al. (1992a) (where P_{W} is a word n-gram model and P_{C} a class n-gram model). For the setting $\theta = \infty$ (clustering of all words), our model is essentially a simple interpolation of a word n-gram and a class n-gram model except that the interpolation parameters are optimized for each class instead of using the same interpolation parameter α for all classes. We have found that $\theta = \infty$ is never optimal; it is always beneficial to assign the most frequent words to their own singleton classes.

3.4 Experimental Setup

Experiments are performed using SRILM (Stolcke, 2002), in particular the Kneser-Ney (KN) and generic class model implementations. Estimation of optimal interpolation parameters is based on Bahl et al. (1991). The baseline system is a word-based modified KN model (Chen and Goodman, 1999).

Following Yuret and Biçici (2009), we evaluate models on the task of predicting the next word from a vocabulary that consists of all words that occur more than once in the training set and the unknown word UNK. Performing this evaluation for KN is straightforward: we map all words with frequency one in the training set to UNK and then compute $P_{\mathrm{KN}}(\mathrm{UNK}\,|h)$ in testing.

In contrast, computing probability estimates for P_{C} is more complicated. We define the vocabulary of the morphological model as the set of all words found in the training set, including frequency-1 words, and one unknown word for each class. We do this because – as we argued above – morphological generalization is only expected to be useful for rare words, so we are likely to get optimal performance for P_{C} if we include all words in clustering and probability estimation, including hapax legomena. Since our testing setup only evaluates on words that occur more than once in the training set, we ideally would want to compute the following estimate when predicting the unknown word:

$$P_C(\mathrm{UNK}_{\mathrm{KN}}\,|h) = \sum_{\{w:N(w)=1\}} P_C(w|h) + \sum_k P_C(\mathrm{UNK}_k\,|h) \quad (3.3)$$

where we distinguish the unknown words of the morphological classes from the unknown word used in evaluation and by the KN model by giving the latter the subscript KN.

However, Eq. 3.3 cannot be computed efficiently and we would not be able to compute it in practical applications that require fast language models. For this reason, we use the modified

class model P'_C in Eq. 3.2 that is defined as follows:

$$P'_C(w|h) = \begin{cases} P_C(w|h) & , \ c(w) \geq 1 \\ P_C(\text{UNK}_{g(w)}|h), & c(w) = 0 \end{cases} \tag{3.4}$$

P'_C and – by extension – P_M are deficient. This means that the evaluation of P_M we present below is pessimistic in the sense that the perplexity reductions would probably be higher if we were willing to spend additional computational resources and compute Eq. 3.3 in its full form.

3.4.1 Distributional Class-based Language Model

The most frequently used type of class-based language model is the distributional model introduced by Brown et al. (1992a). To understand the differences between distributional and morphological class-based language models, we compare our morphological model P_{M} with a distributional model P_{D} that has exactly the same form as P_{M}; in particular, it is defined by Equations (1) and (2). The only difference is that the classes are morphological for P_{M} and distributional for P_{D}.

The algorithm that was used by Brown et al. (1992a) has long running times for large corpora in standard implementations like SRILM. It is thus difficult to conduct the large number of clusterings necessary for an extensive study like ours using standard implementations.

We therefore induce the distributional classes as clusters in a whole-context distributional vector space model (Schütze and Walsh, 2011), a model similar to the ones described by Schütze (1992) and Turney and Pantel (2010) except that dimension words are immediate left and right neighbors (as opposed to neighbors within a window or specific types of governors or dependents). Schütze and Walsh (2011) present experimental evidence that suggests that the resulting classes are competitive with Brown classes.

3.4.2 Data

Our experiments are performed on the Europarl corpus Koehn (2005), a parallel corpus of proceedings of the European Parliament in 21 languages. The languages are members of the following families: Baltic languages (Latvian, Lithuanian), Germanic languages (Danish, Dutch, English, German, Swedish), Romance languages (French, Italian, Portuguese, Romanian, Spanish), Slavic languages (Bulgarian, Czech, Polish, Slovak, Slovene), Uralic languages (Estonian, Finnish, Hungarian) and Greek. In order to make the data sets more comparable, we only use the part of the corpus that can be aligned to English sentences. All 21 corpora are divided into training set (80%), validation set (10%) and test set (10%).

The training set is used for morphological and distributional clustering and estimation of class and KN models. The validation set is used to estimate the OOV rates ϵ and the optimal parameters λ, θ and ϕ.

Table 3.3 gives basic statistics about the corpus.

	Language	T/T	ϵ	#Sentences
S	bg Bulgarian	.0183	.0094	**181,415**
S	cs Czech	.0185	.0097	369,881
S	pl Polish	.0189	.0096	358,747
S	sk Slovak	.0187	.0088	368,624
S	sl Slovene	.0156	.0090	365,455
G	da Danish	.0086	.0077	1,428,620
G	de German	.0091	.0073	1,391,324
G	en English	**.0028**	**.0023**	**1,460,062**
G	nl Dutch	.0061	.0048	1,457,629
G	sv Swedish	.0090	.0095	1,342,667
E	el Greek	.0081	.0079	851,636
R	es Spanish	.0040	.0031	1,429,276
R	fr French	**.0029**	**.0024**	**1,460,062**
R	it Italian	.0040	.0030	1,389,665
R	pt Portuguese	.0042	.0032	1,426,750
R	ro Romanian	.0142	.0079	**178,284**
U	et Estonian	**.0329**	**.0198**	375,698
U	fi Finnish	.0231	**.0183**	1,394,043
U	hu Hungarian	**.0312**	.0163	364,216
B	lt Lithuanian	.0265	.0147	365,437
B	lv Latvian	.0182	.0086	363,104

Table 3.3: Statistics for the 21 languages. S = Slavic, G = Germanic, E = Greek, R = Romance, U = Uralic, B = Baltic. Type/token ratio (T/T) and # sentences for the training set and OOV rate ϵ for the validation set. The two smallest and largest values in each column are bold.

The sizes of the corpora of languages whose countries have joined the European community more recently are smaller than for countries who have been members for several decades. We see that English and French have the lowest type/token ratios and OOV rates; and the Uralic languages (Estonian, Finnish, Hungarian) and Lithuanian the highest. The Slavic languages have higher values than the Germanic languages, which in turn have higher values than the Romance languages except for Romanian.

Type/token ratio and OOV rate are one indicator of how much improvement we would expect from a language model with a morphological component compared to a non-morphological language model.

The relatively low OOV rates can be explained by the tokenization scheme: The tokenization of the Europarl corpus has a preference for splitting tokens in unclear cases. OOV rates would be higher for more conservative tokenization strategies.

		$\mathrm{PP_{KN}}$	θ^*_{M}	ϕ^*	M^*	$\mathrm{PP_C}$	$\mathrm{PP_M}$	Δ_{M}	θ^*_{D}	$\mathrm{PP_{WC}}$	$\mathrm{PP_D}$	Δ_{D}
S	bg	74	200	50	f	103	69	0.07	500	141	71	0.04
S	cs	141	500	100	f	217	129	0.08	1000	298	134	0.04
S	pl	148	500	100	m	241	134	0.09	1000	349	141	0.05
S	sk	123	500	200	f	186	111	0.10	1000	261	116	0.06
S	sl	118	500	100	m	177	107	0.09	1000	232	111	0.06
G	da	69	1000	100	r	89	65	0.05	2000	103	65	0.05
G	de	100	2000	50	m	146	94	0.06	2000	150	94	0.06
G	en	55	2000	50	f	73	53	**0.03**	5000	87	53	0.04
G	nl	70	2000	50	r	100	67	0.04	5000	114	67	0.05
G	sv	98	1000	50	m	132	92	0.06	2000	154	92	0.06
E	el	80	1000	100	f	108	73	0.08	2000	134	74	**0.07**
R	es	57	2000	100	m	77	54	0.05	5000	93	54	0.05
R	fr	**45**	1000	50	f	**56**	**43**	0.04	5000	**71**	**42**	0.05
R	it	69	2000	100	m	101	66	0.04	2000	100	66	0.05
R	pt	62	2000	50	m	88	59	0.05	2000	87	59	0.05
R	ro	76	500	100	m	121	70	0.07	1000	147	71	**0.07**
U	et	256	500	100	m	**422**	230	0.10	1000	668	248	**0.03**
U	fi	**271**	1000	500	f	410	**240**	**0.11**	2000	**706**	**261**	0.04
U	hu	151	200	200	m	222	136	0.09	1000	360	145	**0.03**
B	lt	175	500	200	m	278	161	0.08	1000	426	169	**0.03**
B	lv	154	500	200	f	237	142	0.08	1000	322	147	0.05

Table 3.4: Perplexities on the test set for $n = 4$. S = Slavic, G = Germanic, E = Greek, R = Romance, U = Uralic, B = Baltic. θ^*_x, ϕ^* and M^* denote frequency threshold, suffix count and segmentation method optimal on the validation set. The letters f, m and r stand for the frequency-based method, MORFESSOR and REPORTS. $\mathrm{PP_{KN}}$, $\mathrm{PP_C}$, $\mathrm{PP_M}$, $\mathrm{PP_{WC}}$, $\mathrm{PP_D}$ are the perplexities of KN, morphological class model, interpolated morphological class model, distributional class model and interpolated distributional class model, respectively. Δ_x denotes relative improvement: $(\mathrm{PP_{KN}} - \mathrm{PP_x})/\,\mathrm{PP_{KN}}$. Bold numbers denote maxima and minima in the respective column.

3.5 Results and Discussion

We now discuss our major findings. We performed all our experiments with an n-gram order of 4; this was the order for which the KN model performs best for all languages on the validation set.

3.5.1 Morphological Model

Using grid search, we first determined on the validation set the optimal combination of three parameters: (i) $\theta \in \{100, 200, 500, 1000, 2000, 5000\}$, (ii) $\phi \in \{50, 100, 200, 500\}$ and (iii)

segmentation method. Recall that we only cluster words whose frequency is below θ and only consider the ϕ most frequent suffixes. An experiment with the optimal configuration was then run on the test set. The results are shown in Table 3.4.

The KN perplexities vary between 45 for French and 271 for Finnish. The main result is that the morphological model P_{M} consistently achieves better performance than KN (columns PP_{M} and Δ_{M}), in particular for Slavic, Uralic and Baltic languages and Greek. Improvements range from 0.03 for English to 0.11 for Finnish.

Column θ^*_{M} gives the threshold that is optimal for the validation set. Values range from 200 to 2000. Column ϕ^* gives the optimal number of suffixes. It ranges from 50 to 500. The morphologically complex language Finnish benefits more from more suffixes than morphologically simple languages like Dutch, English and German, but there are a few languages that do not fit this generalization, e.g., Estonian for which 100 suffixes are optimal.

The optimal morphological segmenter is given in column M^*: f = FREQUENCY, r = REPORTS, m = MORFESSOR. The most sophisticated segmenter, MORFESSOR is optimal for about half of the 21 languages, but FREQUENCY does surprisingly well. REPORTS is optimal for two languages, Danish and Dutch. In general, MORFESSOR has an advantage for complex morphologies, but is beaten by FREQUENCY for Finnish and Latvian.

3.5.2 Distributional Model

Columns PP_{D} and Δ_{D} show the performance of the distributional class-based language model. As one would perhaps expect, the morphological model is superior to the distributional model for morphologically complex languages like Estonian, Finnish and Hungarian. These languages have many suffixes that have high predictive power for the distributional contexts in which a word can occur. A morphological model can exploit this information even if a word with an informative suffix did not occur in one of the linguistically licensed contexts in the training set. For a distributional model it is harder to learn this type of generalization.

What is surprising about the comparative performance of morphological and distributional models is that there is no language for which the distributional model outperforms the morphological model by a wide margin. Perplexity reductions are lower than or the same as those of the morphological model in most cases, with only four exceptions – English, French, Italian, and Dutch – where the distributional model is better by one percentage point than the morphological model (0.05 vs. 0.04 and 0.04 vs. 0.03).

Column θ^*_{D} gives the frequency threshold for the distributional model. The optimal threshold ranges from 500 to 5000. This means that the distributional model benefits from restricting clustering to less frequent words – and behaves similarly to the morphological class model in that respect. We know of no previous work that has conducted experiments on frequency thresholds for distributional class models and shown that they increase perplexity reductions.

3.5.3 Sensitivity Analysis of Parameters

Table 3.4 shows results for parameters that were optimized on the validation set. We now want to analyze how sensitive performance is to the three parameters θ, ϕ and segmentation method. To

this end, we present in Table 3.5 the best and worst values of each parameter and the difference in perplexity improvement between the two.

		$\Delta_{\theta^+} - \Delta_{\theta^-}$	θ^+	θ^-	$\Delta_{\phi^+} - \Delta_{\phi^-}$	ϕ^+	ϕ^-	$\Delta_{\mathrm{M}^+} - \Delta_{\mathrm{M}^-}$	M^+	M^-
S	bg	0.03	200	5000	0.01	50	500		f	m
S	cs	0.03	500	5000		100	500		f	r
S	pl	0.03	500	5000	0.01	100	500		m	r
S	sk	0.02	500	5000		200	500	0.01	f	r
S	sl	0.03	500	5000	0.01	100	500		m	r
G	da	0.02	1000	100		100	50		r	f
G	de	0.02	2000	100		50	500		m	f
G	en	0.01	2000	100		50	500		f	r
G	nl	0.01	2000	100		50	500		r	f
G	sv	0.02	1000	100		50	500		m	f
E	el	0.02	1000	100		100	500	0.01	f	r
R	es	0.02	2000	100		100	500		m	r
R	fr	0.01	1000	100		50	500		f	r
R	it	0.01	2000	100		100	500		m	r
R	pt	0.02	2000	100		50	500		m	r
R	ro	0.03	500	5000		100	500		m	r
U	et	0.02	500	5000	0.01	100	50	0.01	m	r
U	fi	0.03	1000	100	0.03	500	50	0.02	f	r
U	hu	0.03	200	5000	0.01	200	50		m	r
B	lt	0.02	500	5000		200	50		m	r
B	lv	0.02	500	5000		200	500		f	r

Table 3.5: Sensitivity of perplexity values to the parameters (on the validation set). S = Slavic, G = Germanic, E = Greek, R = Romance, U = Uralic, B = Baltic. Δ_{x^+} and Δ_{x^-} denote the relative improvement of P_{M} over the KN model when parameter x is set to the best (x^+) and worst value (x^-), respectively. The remaining parameters are set to the optimal values of Table 3.4. Cells with differences of relative improvements that are smaller than 0.01 are left empty.

Differences of perplexity improvement between best and worst values of θ_{M} range between 0.01 and 0.03. The four languages with the smallest difference 0.01 are morphologically simple (Dutch, English, French, Italian). The languages with the largest difference (0.03) are morphologically more complex languages. In summary, the frequency threshold θ_{M} has a comparatively strong influence on perplexity reduction. The strength of the effect is correlated with the morphological complexity of the language.

In contrast to θ, the number of suffixes ϕ and the segmentation method have negligible effect on most languages. The perplexity reductions for different values of ϕ are 0.03 for Finnish, 0.01 for Bulgarian, Estonian, Hungarian, Polish and Slovene, and smaller than 0.01 for the other languages. This means that, with the exception of Finnish, we can use a value of $\phi = 100$ for all languages and be close to the optimal perplexity reduction – either because 100 is optimal

or because perplexity reduction is not sensitive to choice of ϕ. Finnish is the only language that clearly benefits from a large number of suffixes.

Surprisingly, the performance of the morphological segmentation methods is close for 17 of the 21 languages. For three of the four where there is a difference in improvement of ≥ 0.01, FREQUENCY (f) performs best. This means that FREQUENCY is a good segmentation method for all languages, except perhaps for Estonian.

		Δ_W	Δ_P	Δ_N	Δ_O
S	bg	0.07	0.04	0.28	0.16
S	cs	0.09	0.04	0.26	0.33
S	pl	0.10	0.05	0.23	0.22
S	sk	0.10	0.05	0.25	0.28
S	sl	0.10	0.04	0.28	0.28
G	da	0.05	0.05	0.31	0.18
G	de	0.06	0.05	0.40	0.18
G	en	0.03	0.04	0.33	0.14
G	nl	0.04	0.05	0.31	0.26
G	sv	0.06	0.05	0.31	0.35
E	el	0.08	0.05	0.33	0.14
R	es	0.05	0.04	0.26	0.14
R	fr	0.04	0.04	0.29	0.01
R	it	0.04	0.05	0.33	0.02
R	pt	0.05	0.05	0.28	0.39
R	ro	0.08	0.04	0.25	0.17
U	et	0.11	0.05	0.26	0.26
U	fi	0.12	0.06	0.38	0.36
U	hu	0.10	0.04	0.32	0.23
B	lt	0.08	0.06	0.27	0.05
B	lv	0.08	0.05	0.26	0.19

Table 3.6: Relative improvements of P_{M} on the validation set compared to KN for histories w_{i-N+1}^{i-1} grouped by the type of w_{i-1}. The possible types are alphabetic word (W), punctuation (P), number (N) and other (O).

3.5.4 Example Clusters

The Tables 3.7, 3.8 and 3.9 show example clusters with their interpolation weight λ for English, German and Finnish, respectively. λ can be interpreted as a quality measure of a cluster as it shows the trade-off between the word-based KN model and the class-based model after a word of the respective cluster. A weight > 0.5 means that the interpolated model puts more weight on the class-based model than on the KN model. All three languages show a mixture of traditional suffixes such as *-based*, *-ungen* (German, derivational noun suffix + inflectional plural marking) and *-lla* (Finnish, adessive case marking) and special cases such as ranges of numbers, acronyms and file references (such as *B5-321*)).

λ	cluster size	examples
0.85	1858	B5-321 B4-0470 A4-0216 H-0162 A4-0307 H-0563 A5-0103 O-0055
0.42	146	Somal**ia** Goriz**ia** Scan**ia** Bavar**ia** Nad**ia** Amazon**ia** Oualid**ia**
0.40	83	tree-grow**ers** road-us**ers** non-pap**ers** sub-suppli**ers** ring-bind**ers**
0.36	260	18-25 1998-99 1992-2000 180-200 1994-1998 1993-94 17-18 2001-2006
0.30	170	liberal**ism** conform**ism** natural**ism** fantastic**ism** egalitarian**ism**
0.27	83	copper**-based** sex**-based** consensus**-based** health**-based** maize**-based**
0.24	79	democracy-re**lated** community-re**lated** age-re**lated** performance-re**lated**
0.22	61	Cultur**ally** Paradoxic**ally** Materi**ally** Institution**ally** Politic**ally**
0.22	77	MED-MEDA PHARE-CBC M. H. C. PLO- P. Z. EC- V. II-C

Table 3.7: English clusters with their interpolation weight λ, size and some examples at $\theta = 1000, \phi = 100, m =$ FREQUENCY. The average weight is 0.10.

λ	cluster size	examples
0.82	1998	B4-0262 B1-4050 B4-1002 C4-0428 A4-0347 B4-0363 C5-0033
0.39	59	Air-Mitarbei**ter** EU-Finanzminis**ter** Ad-hoc-Charak**ter**
0.37	62	portugiesisch-brit**ische** finnisch-schwed**ische** französisch-span**ische**
0.33	62	UN-Generalversamm**lung** Hardware-Entwick**lung** Software-Entwick**lung**
0.32	878	fakturi**ert** moki**ert** demobilisi**ert** signi**ert** kassi**ert** gemart**ert**
0.25	251	salbad**ern** näh**ern** anzuford**ern** alt**ern** trau**ern** zuzusteu**ern** mod**ern**
0.23	121	UCLAF-Untersuch**ungen** EU-Entscheid**ungen** Ad-hoc-Lös**ungen**
0.22	1267	0014 0300 6119 3975 14000 0149 640 05 820 2299 275 136 652 0027 8014
0.22	190	EURES-Netzwerk**e** UNO-Friedenstrupp**e** Dehaene-Gruppe

Table 3.8: German clusters with their interpolation weight λ, size and some examples at $\theta = 1000, \phi = 100, m =$ FREQUENCY. The average weight is 0.12.

λ	cluster size	examples
0.85	1881	C5-0250 BBC1- B4-0391 A5-0024 O-0045 C4-0171 C4-0559
0.55	77	Macartney**lla** Keskustelu**lla** Korfu**lla** Urheilu**lla** Arvio**lla**
0.49	55	vireillepano-oikeu**tta** -periaate**tta** epätasa-arvoisuu**tta**
0.47	59	Kongo-Brazzaville**ssä** Eurochambre-yhdistykse**ssä** Riski-iä**ssä**
0.42	60	esi-yhdentymisstrateg**ia** tasa-arvoneuvonantaj**ia** -ohjelm**ia**
0.39	86	offshore-al**aa** sikarutto-ongelm**aa** johdanto-os**aa** läsnäolo-ongelm**aa**
0.39	307	Käytännölli**siä** Estéve**ziä** Schmid**iä** Flemming**iä** Tämänhetki**siä**
0.36	339	13.40 15,5 6.3 11.41 11.35 13.17 19,3 11,5 11.00 13.02 18,6 1,55
0.34	69	offshore-al**aan** suunnitelma-asiakirj**aan** -asi**aan** ennalta-arvaamattom**aan**

Table 3.9: Finnish clusters with their interpolation weight λ, size and some examples at $\theta = 1000, \phi = 100, m =$ FREQUENCY. The average weight is 0.08.

3.5.5 Impact of Shape

The basic question we are asking is to what extent the sequence of characters a word is composed of can be exploited for better prediction in language modeling. In the final analysis in Table 3.6 we look at four different types of character sequences and their contributions to perplexity reduction. The four groups are alphabetic character sequences (W), numbers (N), single special characters (P = punctuation), and other (O). Examples for O would be "751st" and words containing special characters like "O'Neill". The parameters used are the optimal ones of Table 3.4. Table 3.6 shows that the impact of special characters on perplexity is similar across languages: $0.04 \leq \Delta_P \leq 0.06$. The same is true for numbers: $0.23 \leq \Delta_N \leq 0.33$, with two outliers that show a stronger effect of this class: Finnish $\Delta_N = 0.38$ and German $\Delta_N = 0.40$.

The fact that special characters and numbers behave similarly across languages is encouraging as one would expect less cross-linguistic variation for these two classes of words.

In contrast, "true" words (those exclusively composed of alphabetic characters) show more variation from language to language: $0.03 \leq \Delta_W \leq 0.12$. The range of variation is not necessarily larger than for numbers, but since most words are alphabetical words, class W is responsible for most of the difference in perplexity reduction between different languages. As before we observe a negative correlation between morphological complexity and perplexity reduction; e.g., Dutch and English have small Δ_W and Estonian and Finnish large values.

We provide the values of Δ_O for completeness. The composition of this catch-all group varies considerably from language to language. For example, many words in this class are numbers with alphabetic suffixes like "2012-ben" in Hungarian and words with apostrophes in French.

3.6 Conclusion

We have investigated an interpolation of a KN model with a class-based language model whose classes are defined by morphology and shape features. We tested this model in a large cross-lingual study of European languages.

Even though the model is generic and we use the same architecture and features for all languages, the model achieves reductions in perplexity for all 21 languages represented in the Europarl corpus, ranging from 3% to 11%, when compared to a KN model. We found perplexity reductions across all 21 languages for histories ending with four different types of word shapes: alphabetical words, special characters, and numbers.

We looked at the sensitivity of perplexity reductions to three parameters of the model: θ, a threshold that determines for which frequencies words are given their own class; ϕ, the number of suffixes used to determine class membership; and morphological segmentation. We found that θ has a considerable influence on the performance of the model and that optimal values vary from language to language. This parameter should be tuned when the model is used in practice.

In contrast, the number of suffixes and the morphological segmentation method only had a small effect on perplexity reductions. This is a surprising result since it means that simple identification of suffixes by frequency and choosing a fixed number of suffixes ϕ across languages is sufficient for getting most of the perplexity reduction that is possible.

Chapter 4

HMM-LAs for Dependency Parsing

Erklärung nach §8 Absatz 4 der Promotionsordnung: This chapter covers work already published at an international peer-reviewed conference. The relevant publication is Müller et al. (2014). The research described in this chapter was carried out in its entirety by myself. The other author(s) of the publication(s) acted as advisor(s) or were responsible for work that was reported in the publication(s), but is not included in this chapter.

In this chapter we propose a method to increase dependency parser performance without using additional labeled or unlabeled data by refining the layer of predicted part-of-speech (POS) tags. This procedure is interesting in the context of transfer learning (Hwa et al., 2005), where the morpho-syntactic annotation of one language is transferred to a second language by means of a parallel corpus. This projection often involves a reduced universal part-of-speech (POS) tagset (Petrov et al., 2012). These coarse POS can be more easily projected, but also lack many of the often morphological distinctions that a language independent tagset would make and are thus less informative features for syntactic parsers. The induction procedure in this chapter could be used to extend the projected POS of an MRL and thus yield better results on a downstream task such as syntactic parsing.

In particular, we show that induced tagsets can yield better performance than treebank tagsets when evaluated on a universal POS (Petrov et al., 2012) tagging task, which proves that automatically induced POS can be as informative as manually derived POS. In parsing experiments on English and German, we show significant improvements for both languages. Our refinement is based on hidden Markov models with latent annotations (HMM-LA) (Huang et al., 2009), for which we propose a modified training procedure that significantly improves the tagging accuracy of the resulting HMM taggers.

4.1 Introduction

Generative split-merge training for probabilistic context-free grammars (PCFGs) has been shown (Petrov et al., 2006) to yield phrase structure parsers with state-of-the-art accuracy and linguistically comprehensible latent annotations. While split-merge training can also be applied to

hidden Markov models (Huang et al., 2009), the resulting taggers stay somewhat behind the performance of state-of-the-art discriminative taggers (Eidelman et al., 2010). In this chapter we address the question of whether the resulting latent POS tags are linguistically meaningful and useful for upstream tasks such as syntactic parsing. We find that this is indeed the case, leading to a procedure that significantly increases the performance of dependency parsers. The procedure is attractive because the refinement of predicted part-of-speech sequences using a coarse-to-fine strategy (Petrov and Klein, 2007) is fast and efficient. The contributions of this chapter are as follows:

1. We propose several enhancements to split-merge training and show that they give better results than the basic form of the method proposed by (Huang et al., 2009).

2. We explain the linguistic and practical properties of the induced POS tagsets.

3. We show that incorporating the induced POS into a state-of-the-art dependency parser (Bohnet, 2010) gives substantial increases in accuracy (increasing LAS from 90.34 to 90.57 for English and from 87.92 to 88.24 when not using morphological features and from 88.35 to 88.51 when using morphological features for German)

We first discuss prior work on latent sequence modeling in section 4.2. In section 4.3 we propose a number of enhancements which we show in section 4.4.1 to lead to significant improvements in POS accuracy. In section 4.4.2 we give an overview of the properties of the induced subtags and the linguistic phenomena that the subtags capture. In section 4.4.3, we discuss our experiments on dependency parsing.

4.2 Related Work

Petrov et al. (2006) introduce generative split-merge training for PCFGs and provide a fully automatic method for training state-of-the-art phrase structure parsers. They also argue that the resulting latent annotations are linguistically meaningful. Sun et al. (2008) induce latent substates into CRFs and show that noun phrase (NP) recognition can be improved, especially if no part-of-speech features are available. Huang et al. (2009) apply split-merge training to create HMMs with latent annotations (HMM-LA) for Chinese POS tagging. They report that the method outperforms standard generative bigram and trigram tagging, but do not compare to discriminative methods. Eidelman et al. (2010) show that a bidirectional variant of latent HMMs with incorporation of prosodic information can yield state-of-the-art results in POS tagging of conversational speech. Finkel et al. (2007) split POS using a generative model based on gold dependency trees and also find improvements over a baseline parser.

4.3 Enhancements

We remind the reader that we introduced the basic functionality of HMM-LAs as proposed by Huang et al. (2009) in Section 2.3.2. In this section, we propose a number of modifications which

result in more efficient training as well as more accurate models. Section 4.4.1 evaluates these modifications experimentally.

Smoothing of estimated frequencies

To prevent the specialized tags from moving too far from their base tags and thus to increase the robustness of the method, we smooth the estimated frequencies using recursive Witten-Bell (WB) smoothing in the direction of their parents. We introduced Witten-Bell smoothing for n-gram modeling in Chapter 2. We can easily get a version for a simple emission probability by replacing the history with the tag:

$$P_{\mathrm{WB}}(x|t) = (1 - \gamma_{\mathrm{WB}}(t)) \cdot P_{\mathrm{ML}}(x|t) + \gamma_{\mathrm{WB}}(t) \cdot P_{\mathrm{BO}}(x|t) \tag{4.1}$$

$$\gamma_{\mathrm{WB}}(t) = \frac{N_{1+}(t\bullet)}{N_{1+}(t\bullet) + c_{t,\bullet}} \tag{4.2}$$

where $c_{t,\bullet} = \sum_x c_{t,x}$ and $N_{1+}(t\bullet)$ is the number of different word types tag t has been observed with. In our model we want to smooth counts instead of probabilities, we can derive a count version by multiplying with the tag frequency $c_{t,\bullet}$:

$$\begin{aligned} c_{t,\bullet} \cdot P_{\mathrm{WB}}(x|t) &= c_{t,\bullet} \cdot [(1 - \gamma_{\mathrm{WB}}(t)) \cdot P_{\mathrm{ML}}(x|t) + \gamma_{\mathrm{WB}}(t) \cdot P_{\mathrm{BO}}(x|t)] \\ &= (1 - \gamma_{\mathrm{WB}}(t)) \cdot c_{t,x} + c_{t,\bullet} \cdot \gamma_{\mathrm{WB}}(t) \cdot P_{\mathrm{BO}}(x|t) \\ &\propto c_{t,x} + \frac{c_{t,\bullet} \cdot \gamma_{\mathrm{WB}}(t)}{1 - \gamma_{\mathrm{WB}}(t)} \cdot P_{\mathrm{BO}}(x|t) \\ &= c_{t,x} + N_{t+1}(t\bullet) \cdot P_{\mathrm{BO}}(x|t) \end{aligned} \tag{4.3}$$

We can thus set:

$$c^{\mathrm{WB}}_{t,x} = c_{t,x} + N_{1+}(t\bullet) \cdot P_{\mathrm{BO}}(x|t) \tag{4.4}$$

However, $N_{1+}(t\bullet)$ was defined for hard integer counts, while during EM training we obtain soft counts. We thus use a soft version of $N_{1+}(t\bullet)$:

$$N'_{1+}(t\bullet) = \sum_x \min(1,\, c_{t,x}) \tag{4.5}$$

As back-off distribution we use the distribution of the parent tag in the tag hierarchy:

$$P_{\mathrm{BO}}(x|t) = \frac{c^{\mathrm{WB}}_{\pi(t),x}}{\sum_{x'} c^{\mathrm{WB}}_{\pi(t),x'}} \tag{4.6}$$

where $\pi(t)$ denotes the parent of t in the induced tag hierarchy. The recursion stops at base tags for which we set $c^{\mathrm{WB}}_{t,x} = c_{t,x}$. Petrov et al. (2006) propose linear smoothing that backs off to the base tag immediately.

Smoothing of emission probabilities

We also smooth emission probabilities, primarily to be able to handle unknown words. We model emission probabilities $P(w|t)$ by applying Bayes theorem: $P(w|t) \propto_{w=const} \frac{P(t|w)}{P(t)}$. Using Witten-Bell, the probabilities $P(t|w)$ are smoothed between the actual frequencies and a signature-based back-off model:

$$P_{\mathrm{WB}}(t|w) = \frac{c_{t,w} + N_t \cdot P(t|g(w))}{c_w + N_t} \tag{4.7}$$

As signature $g(w)$ of a word w we use the signatures from Petrov et al. (2006).

Sampling

In our initial experiments, we observed that the model starts to overfit held-out data at huge tagset sizes. In order to dampen this effect we run the E-step on *uniform* samples of the training set. The sample strategy selects each sentence randomly with a probability equal to the sampling rate. The intuition behind this is that every sample has general properties and specific properties. An optimal learner fits the model according to the general properties and ignores the sample-specific ones. By doing E-steps on different samples we expect the special properties to average out and force the model to better generalize to new data. After the EM training we run one additional EM-step on the complete training set to make use of all the training data. We call the percentage of sentences that are sampled from the training set at each EM-iteration *sampling rate* and use a sampling rate of 0.1 throughout the chapter.

4.4 Experiments

4.4.1 POS Tagging

In this section, we evaluate the basic training as well as the enhancements we just introduced in section 4.3. All experiments are performed on the English and German parts of the CoNLL 2009 data (Hajič et al., 2009). We evaluate our taggers on the universal POS tagset (Petrov et al., 2012) to make results comparable between languages, as well as on the treebank tagsets to show the effects on more fine-grained tagsets. We compare our numbers to the latest version of the Stanford tagger (Toutanova et al., 2003) (v3.1.4) using all the features except the distributional similarity features which are based on additional unlabeled data. All experiments are performed with 10 EM iterations after each split and merge phase and all tagging accuracies of latent models are averages over 10 independent runs.

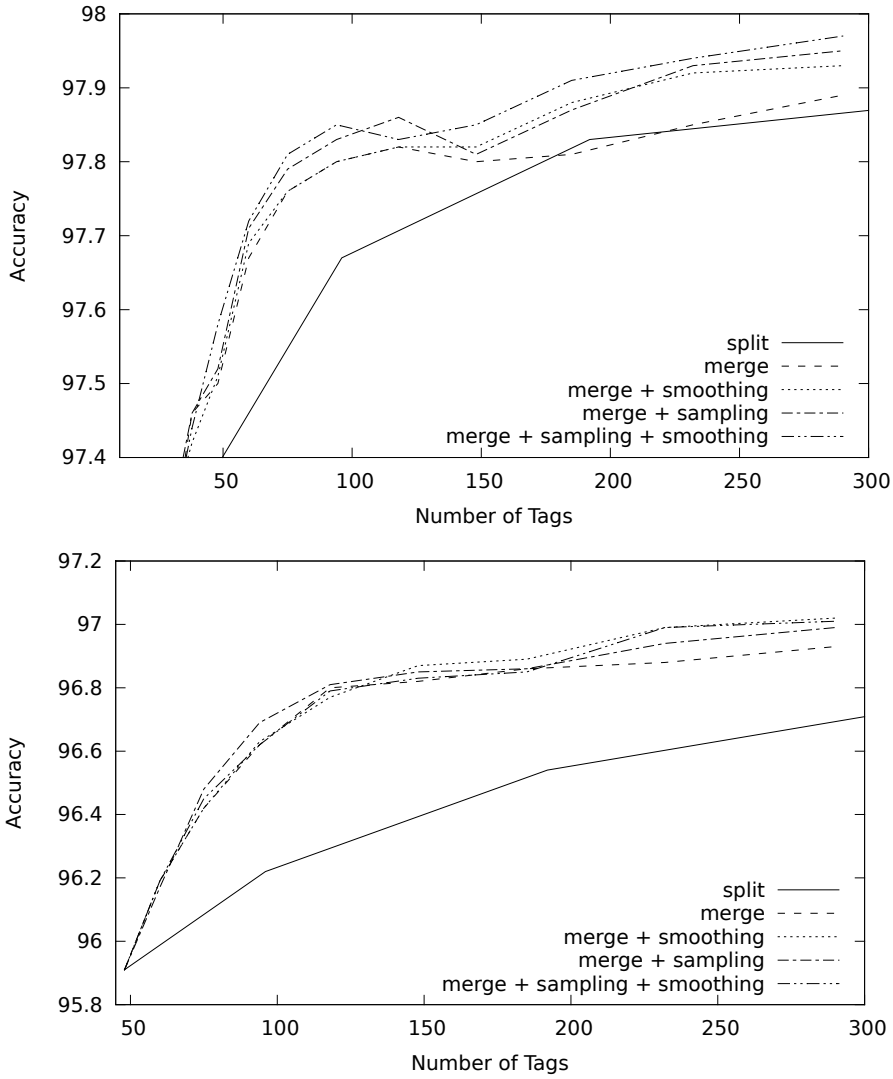

Figure 4.1: Training on English universal POS data (top) and Penn-Treebank POS data (bottom) over several iterations. We compare HMM-LAs with split training (split), split-merge training (merge), WB smoothing (merge + smoothing), EM sampling (merge + sampling) and both (merge + sampling + smoothing).

We first look at the development of the accuracy over an increasing number of latent tags, that is as training progresses. Figure 4.1 shows the training for English universal and Penn-Treebank POS tagging. We compare an HMM-LA that only splits (split) a standard HMM-LA (Huang et al., 2009) (merge) and models that use WB smoothing (merge + smoothing), sampling (merge + sampling), or both (merge + sampling + smoothing). For the case of universal POS tagging, we see that the baseline models *split* and *merge* are outperformed by our enhanced models. Smoothing gives a lower improvement than sampling, but both can be combined to yield the most accurate model. In the Penn-Treebank results we see a similar picture, but the improvement

due to the enhancements is smaller and combining both techniques is slightly worse than the WB smoothed model.

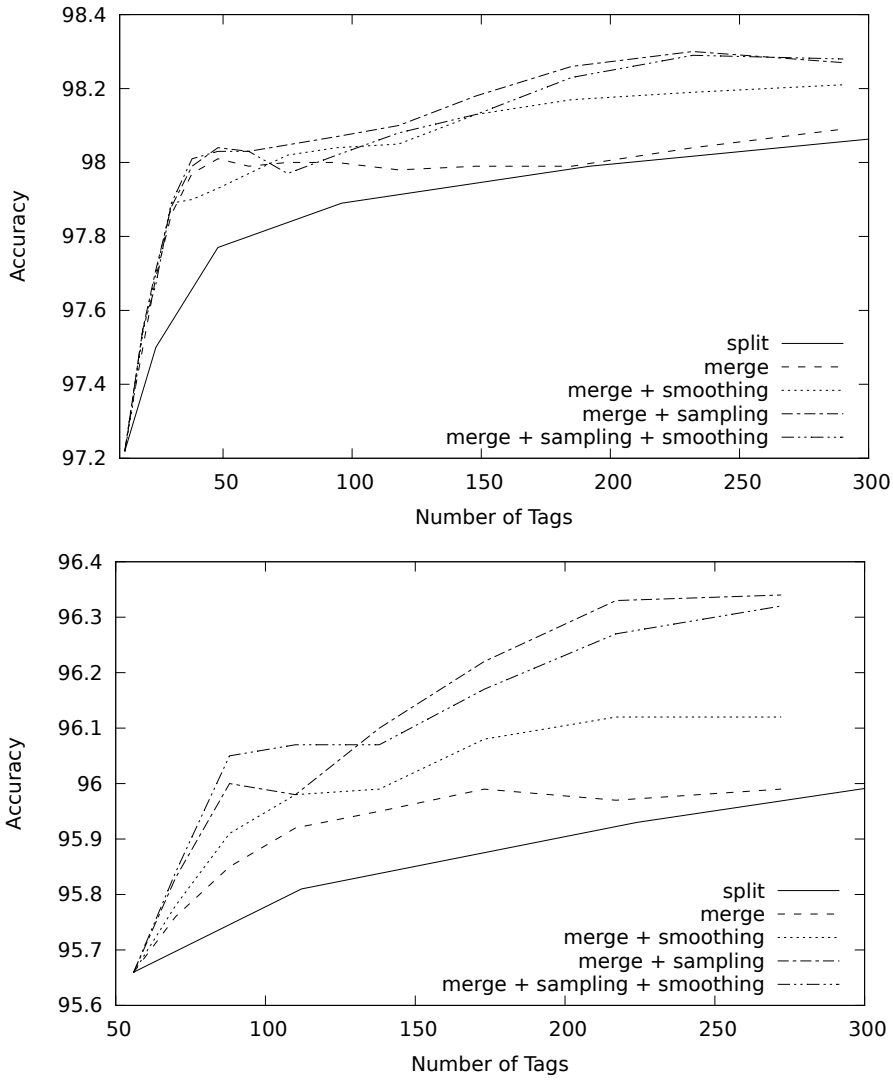

Figure 4.2: Training on German universal POS data (top) and Tiger POS data (bottom) over several iterations. We compare HMM-LAs with split training (split), split-merge training (merge), WB smoothing (merge + smoothing), EM sampling (merge + sampling) and both (merge + sampling + smoothing).

For German (Figure 4.2), we get a consistent improvement of the enhanced models over the baseline models. However, combining the two enhancements is never substantially better than the sampling model on its own.

Figure 4.1 and 4.2 show that – given a large enough tagset – the two innovations introduced here (sampling and smoothing) improve tagging accuracy. Overall, WB smoothing has a smaller impact than sampling.

In a second experiment we want to demonstrate that a HMM-LA can find a tagset of similar size as the treebank tagset, but with higher accuracy. HMM taggers rely heavily on preceding and following POS tags as features for the prediction at the current position. As a result, a HMM trained on the universal POS tagset (consisting of 12 tags) achieves a lower accuracy than a HMM trained on a more fine-grained treebank tagset, which after decoding maps the predicted tags to the coarse tagset. For English we get an accuracy of 96.21 against an accuracy of 97.20 for a tagger trained on the Penn Treebank tagset (48 tags) and for German an accuracy of 97.22 against 98.01 for the Tiger tagset (57). For English (en) and German (de) universal POS data we compare the HMM models based on treebank tagsets (tree) with HMM-LAs with 48 latent tags, trained using: split-merge (m), WB smoothing (wb), sampling (sa) and WB smoothing and sampling (wb+sa):

	tree	m	wb	sa	wb+sa
en	97.20	97.50	97.51	97.52	**97.58***
de	98.01	98.01	97.93	**98.04**	98.03

Table 4.1: English and German universal POS tagging accuracies for HMMs based on treebank tagsets (tree), split-merge training (m), split-merge training with smoothing (wb) and split-merge training with sampling (sa) using 48 latent tags. The best numbers are bold. Numbers significantly better then the baselines models (tree, m) are marked (*).

The highest values for both languages lie above the values we obtain for the treebank tagset. However, the difference is only significant for English. Throughout the chapter we establish significance by running approximate randomization tests (Yeh, 2000). We consider p-values < 0.05 significant. Still we see that a HMM-LA can produce a tagset that at a similar size is as accurate as the Penn Treebank tagset (48 tags) and the Tiger tagset (56 tags). To demonstrate that our enhancements outperform the simple form of split-merge training we further increase the tagset sizes to 290 tags:

	tree	m	wb	sa	wb+sa
en	97.20	97.89	97.93	97.95	**97.96**
de	98.01	98.09	98.21	98.27*	**98.28***

Table 4.2: English and German universal POS tagging accuracies for HMMs based on treebank tagsets (tree), split-merge training (m), split-merge training with smoothing (wb) and split-merge training with sampling (sa) using 290 latent tags. The best numbers are bold. Numbers significantly better then the baselines models (tree, m) are marked (*).

As for the smaller tag sizes we see that sampling gives a small improvement for English; for German, however, the difference is substantial (.18) and also significant. Note that for German the accuracy of the simple training stagnates between approximately 50 and 170 tags (cf. Figure 4.2). When training and evaluating on the treebank tagsets we see a similar picture:

	m	wb	sa	wb+sa
en	96.93	**97.02**	96.99	97.01
de	95.99	96.12	**96.34***	96.32*

Table 4.3: English and German treebank POS tagging accuracies for split-merge training (m), split-merge training with smoothing (wb) and split-merge training with sampling (sa) and optimal latent tagset sizes. The best numbers are bold. Numbers significantly better then the baselines models (tree, m) are marked (*).

Again the improvement for sampling is significant for German, but not for English. Another observation is that there still remains a significant gap between generative and discriminative POS tagging: for the four tasks the corresponding Stanford tagger accuracies are:

	English		German	
	UPOS	Penn	UPOS	Tiger
HMM-LA	97.96	97.01	98.28	96.32
Stanford	**98.19***	**97.34***	**98.55***	**97.02***

Table 4.4: Tagging accuracies for the best HMM-LA models and the Stanford Tagger on different tagset. The best numbers are bold. Significant improvements are marked (*).

We conclude that sampling gives consistent improvements for both languages, but significant improvements only for German. WB smoothing also gives consistent improvements over simple split-merge training, but never significant improvements. Combining smoothing and sampling does not lead to additional improvements. One explanation why the results for sampling are not significantly better for English might be that sampling especially helps unknown word tagging: For the German Tiger experiments, for example, the unknown word accuracy improves from 76.78 (with 12 tags) to 78.39 using simple split-merge training and to 79.71 using split-merge training with sampling – much larger improvements than for known words. This might explain the impact of sampling, because the unknown word rate for German is approximately four times higher than for English. Though the Stanford tagger significantly outperforms the HMM-LAs on all four tasks, the magnitude of the differences is remarkably small taking the simplicity of the features used by the HMM-LA into account. We have shown that split merge training yields tagsets that are at least as informative for HMM POS tagging as sophisticated treebank tagsets. We next investigate (i) to what extent the induced tags are linguistically interpretable and (ii) if they are useful as features for a downstream task, namely dependency parsing.

4.4.2 Properties of the Induced Tags

In this section we analyze the split decisions of the first level made by one particular training run on English and German universal POS data. We compare the split by their form distributions,

the corresponding gold treebank tags and the distributions of preceding universal tags. Table 4.5 and Table 4.6 show the statistics used for this analysis.

English		
Universal Tag	Tag_0	Tag_1
Adjectives	more (0.05) many (0.03) last (0.03)	new (0.03) other (0.03) first (0.02)
(ADJ)	JJ (0.85) JJR (0.11) JJS (0.04)	JJ (0.95) JJR (0.03) JJS (0.03)
	VERB (0.32) ADV (0.27) NOUN (0.14)	**DET** (0.39) ADP (0.17) ADJ (0.10)
Particles	's (0.93) ' (0.07)	to (0.89) up (0.04) out (0.02) off (0.01)
(PRT)	**POS** (1.00)	**TO** (0.89) **RP** (0.10)
	NOUN (0.99)	VERB (0.43) NOUN (0.34) ADJ (0.07)
Prepositions	that (0.11) in (0.10) by (0.09)	of (0.43) in (0.19) for (0.11)
(ADP)	IN (1.0)	IN (1.0)
	VERB (0.46) NOUN (0.15) . (0.13)	**NOUN** (0.84) NUM (0.06) ADJ (0.03)
Pronouns	its (0.30) their (0.15) his (0.14)	it (0.21) he (0.16) they (0.12)
(PRON)	**PRP$** (0.68) PRP (0.26) WP (0.05)	**PRP** (0.87) WP (0.11) PRP$ (0.02)
	VERB (0.46) ADP (0.38) PRT (0.05)	. (0.37) ADP (0.17) VERB (0.16)
Verbs	be (0.06) been (0.02) have (0.02)	is (0.10) said (0.08) was (0.05)
(VERB)	VB (0.42) VBN (0.29) VBG (0.20)	VBD (0.37) VBZ (0.29) VBP (0.14) MD (0.13)
	VERB (0.38) PRT (0.22) ADV (0.11)	**NOUN** (0.52) **PRON** (0.20) . (0.12)

Table 4.5: English induced subtags and their statistics. The three rows in each cell contain word forms (row 1), treebank tags (row 2) and preceding universal tags (row 3). Statistics pointing to linguistically interesting differences are highlighted in bold.

Interesting observations are: The universal POS tagset puts the three Penn Treebank tags RP (particle), POS (possessive marker) and TO into one particle tag (see "Particles (PRT)" in Table 4.5). The split-merge training essentially reverses this by splitting particles first into possessive and non-possessive markers and in a subsequent split the non-possessives into TO and particles.

For German we have a similar split into verb particles, negation particles such as *nicht* 'not' and the infinitive marker *zu* 'to' ("Particles (PRT)" in Table 4.6). English prepositions get split by proximity to verbs or nouns ("Prepositions (ADP)"). Subordinate conjunctions such as *that*, which in the Penn Treebank annotation are part of the preposition tag IN, get assigned to the subclass next to verbs. For German we also see a separation of "Conjunctions (CONJ)" into predominantly subordinate conjunctions (Tag 0) and predominantly coordinating conjunctions (Tag 1).

German		
Universal Tag	Tag_0	Tag_1
Conjunctions (CONJ)	daß (0.26) wenn (0.08) um (0.06) **KOUS** (0.58) KON (0.30) KOUI (0.06) . (0.93) ADV (0.03) CONJ (0.03	und (0.76) oder (0.07) als (0.06) **KON** (0.88) KOKOM (0.10) APPR (0.02) NOUN (0.56) VERB (0.15) ADJ (0.09)
Particles (PRT)	an (0.13) aus (0.10) ab (0.09) PTKVZ (0.92) ADV (0.04) ADJD (0.01) NOUN (0.65) PRON (0.08) VERB (0.07)	**nicht** (0.49) **zu** (0.46) Nicht (0.01) PTKNEG (0.52) PTKZU (0.44) PTKA (0.02) NOUN (0.43) ADV (0.14) VERB (0.14)
Pronouns (PRON)	sich (0.13) die (0.08) es (0.07) **PPER** (0.33) PRF (0.14) PRELS (0.14) VERB (0.32) . (0.28) CONJ (0.12)	ihre (0.06) seine (0.05) seiner (0.05) **PPOSAT** (0.40) PIAT (0.34) PDAT (0.16) ADP (0.33) NOUN (0.18) VERB (0.11)
Verbs (VERB)	werden (0.04) worden (0.02) ist (0.02) VVPP (0.31) VVINF (0.24) VVFIN (0.17) NOUN (0.46) VERB (0.22) PRT (0.10)	ist (0.07) hat (0.04) sind (0.03) VVFIN (0.50) VAFIN (0.36) VMFIN (0.12) NOUN (0.49) **.** (0.19) PRON (0.16)

Table 4.6: German induced subtags and their statistics. The three rows in each cell contain word forms (row 1), treebank tags (row 2) and preceding universal tags (row 3). Statistics pointing to linguistically interesting differences are highlighted in bold.

For both languages adjectives get split by predicative and attributive use. For English the predicative subclass also seems to hold rather atypical adjectives such as “such” and “last”. For English, verbs (“Verbs (VERB)”) get split into a predominantly infinite tag (Tag 0) and a predominantly finite tag (Tag 1) while for German we get a separation by verb position. In German we get a separation of pronouns (“Pronouns (PRON)”) into possessive and non-possessive; in English, pronouns get split by predominant usage in subject position (Tag 0) and as possessives (Tag 1). We can conclude that split-merge training generates annotations that are to a considerable extent linguistically interpretable. In the next section we evaluate the utility of these annotations for dependency parsing.

4.4.3 Dependency Parsing

	#Tags	μ_{LAS}	$\max_{\text{LAS}}$	σ_{LAS}	μ_{UAS}	$\max_{\text{UAS}}$	σ_{UAS}
English	Baseline	88.43			91.46		
	58	88.52	88.59	0.06	91.52	91.61	0.08
	73	88.55	88.61	0.05	91.54	91.59	0.04
	92	88.60	88.71	0.08	**91.60**	**91.72**	0.08
	115	**88.62**	**88.73**	0.07	91.58	91.71	0.08
	144	88.60	88.70	0.07	**91.60**	91.71	0.07
German (no feat.)	Baseline	87.06			89.54		
	85	87.09	87.18	0.06	89.61	89.67	0.04
	107	**87.23**	**87.36**	0.09	89.74	89.83	0.08
	134	87.22	87.31	0.09	**89.75**	**89.86**	0.09
German (feat.)	Baseline	87.35			89.75		
	85	87.33	87.47	0.11	89.76	89.88	0.09
	107	**87.43**	**87.73**	0.16	**89.81**	**90.14**	0.17
	134	87.38	87.53	0.08	89.75	89.89	0.08

Table 4.7: Labeled (LAS) and Unlabeled Attachment Score (UAS), mean, best value and standard deviation for the development set for English and German dependency parsing with (feat.) and without morphological features (no feat.). The best numbers are bold.

In this section we investigate the utility of induced POS as features for dependency parsing. We run our experiments on the CoNLL-2009 data sets (Hajič et al., 2009) for English and German. As a baseline system we use the latest version of the mate-tools parser[1] (Bohnet, 2010). It was the highest scoring syntactic parser for German and English in the CoNLL 2009 shared task evaluation. The parser gets automatically annotated lemmas, POS and morphological features as input which are part of the CoNLL-2009 data sets. The automatically annotated tagsets are slightly smaller than the gold tagsets: For English 47 and for German 55.

In this experiment we want to examine the benefits of tag refinements isolated from the improvements caused by using two taggers in parallel, thus we train the HMM-LA on the automatically tagged POS sequences of the training set and use it to add an additional layer of refined POS to the input data of the parser. We do this by calculating the forward-backward charts that are also used in the E-steps during training – recall that in these charts base tags of the refined tags are constrained to be identical to the automatically predicated tags.

We integrate the tags by adding one additional feature for every edge: the conjunction of latent tags of the two words connected by the edge.

In preliminary experiments we found the latent models to produce better results when trained with more EM iterations (we use 100) and without WB smoothing. We think that this is because WB smoothing and few EM iterations generate less specific or more similar subtags. For the HMM-LA tagger accuracy, less or more specific tags do not seem to make a big difference

[1] We use v3.3 of the graph-based parser.

because the decoder sums over all possible subtags at a certain position. For the parser, however, we just extract the most probable tag, thus small differences between the most probable and second most probable tag lead to more noisy training data.

The results of our experiments are shown in Table 4.7. All numbers are averages of five independent runs. For English the smaller models with 58 and 73 tags achieve improvements of approximately 0.1. The improvements for the larger tagsets are approximately 0.2. The best individual model improves LAS by 0.3. For the German experiments without morphological features we get only marginal average improvements for the smallest tagset and improvements of approx. 0.15 for the bigger tagsets. The average ULA scores for 107 and 134 tags are at the same level as the ULA scores of the baseline with morphological features. The best model improves LAS by 0.3. For German with morphological features the absolute differences are smaller: The smallest tagset does not improve the parser on average. For the tagset of 107 tags the average improvement is 0.08. The best model improves LAS by 0.38. In all experiments we see the highest improvements for tagset sizes of roughly the same size (115 for English, 107 for German). While average improvements are low especially for the German models with morphological features, the peak improvements are substantial.

Running the best English system on the test set gives an improvement in LAS from 90.34 to 90.57 (Table 4.8); this improvement is significant ($p < .02$). For German we get an improvement from 87.92 to 88.24 without and from 88.35 to 88.51 with morphological features. The difference between the values without morphological features is significant ($p < .05$), but the difference between models with morphological features is not ($p = .26$). However, the difference between the baseline system with morphological features and the best system without morphological features is also not significant ($p = .49$).

	Baseline	HMM-LA
English	90.34	**90.75***
German (feat)	88.35	**88.51**
German (no feat)	87.92	**88.24***

Table 4.8: LAS for the test set for English and German dependency parsing with (feat.) and without morphological features (no feat.). The best numbers are bold. Significant improvements are marked (*).

We can conclude that HMM-LA tags can significantly improve parsing results. For German we see that HMM-LA tags can substitute morphological features up to an insignificant difference. We also see that morphological features and HMM-LA seem to be correlated as combining the two gives only insignificant improvements.

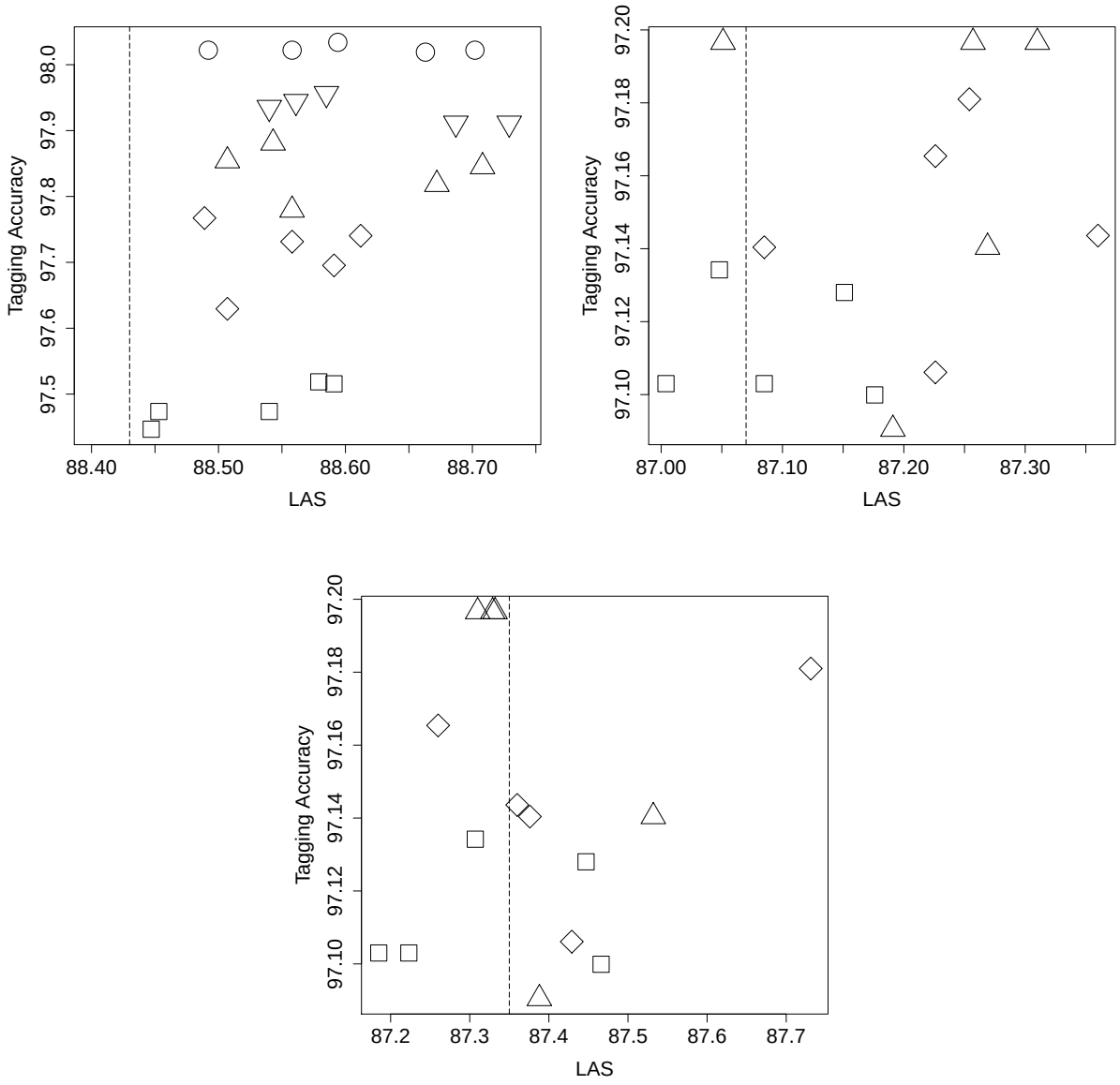

Figure 4.3: Scatter plots of LAS vs tagging accuracy for English (top left) and German without (top right) and with (bottom) morphological features. English tagset sizes are 58 (squares), 73 (diamonds), 92 (triangles), 115 (triangles pointing downwards) and 144 (circles). German tagset sizes are 85 (squares), 107 (diamonds) and 134 (triangles). The dashed lines indicate the baselines.

4.4.4 Contribution Analysis

In this section we try to find statistical evidence for why a parser using a fine-grained tagset might outperform a parser based on treebank tags only.

The results indicate that an induced latent tagset as a whole increases parsing performance. However, not every split made by the HMM-LA seems to be useful for the parser. The scatter plots in Figure 4.3 show that there is no strict correlation between tagging accuracy of a model and the resulting LAS. This is expected as split-merge training optimizes a tagging objective function, which does not directly translate into better parsing performance. An example is lexicalization. Most latent models for English create a subtag for the preposition "of". This is useful for a HMM as "of" is frequent and has a very specific context. A lexicalized syntactic parser, however, does not benefit from such a tag.

We base the remainder of our analysis on the results of the baseline parser on the English development set and the results of the best performing latent model. The best performing model has a LAS score of 88.73 vs 88.43 for the baseline, a difference of 0.3. If we just look at the LAS of words with incorrectly predicted POS we see a difference of 1.49. A look at the data shows that the latent model helps the parser to identify words that might have been annotated incorrectly. As an example consider plural nouns (NNS) and two of their latent subtags NNS_1 and NNS_2 and how often they get classified correctly and misclassified as proper nouns (NNPS):

	Gold NNS	Gold NNPS
Predicted NNS	**2019**	**104**
Latent NNS_1	90	72
Latent NNS_2	**1100**	13
Latent NNS_3	...	...

Table 4.9: Coocurrences of gold POS (columns) and predicted POS (NNS) and latent POS ($\mathrm{NNS}_1, \mathrm{NNS}_2, \mathrm{NNS}_3$).

We see that NNS_1 is roughly equally likely to be a NNPS or NNS while NNS_2 gives much more confidence of the actual POS being NNS. So one benefit of HMM-LA POS tagsets are tags of different levels of confidence.

Another positive effect is that latent POS tags have a higher correlation with certain dependency relations. Consider proper nouns (NNP):

	Gold NAME	Gold NMOD	Gold SBJ
Predicted NNP	**962**	**662**	**468**
Latent NNP_1	10	27	**206**
Latent NNP_2	24	50	**137**
Latent NNP_3	...	...	...

Table 4.10: Coocurrences of correct dependency relations Name (Name), Noun Modifier (NMOD), subject (SBJ) and predicted POS and latent POS ($\mathrm{NNP}_1, \mathrm{NNP}_2, \mathrm{NNP}_3$).

We see that NNP_1 and NNP_2 are more likely to appear in subject relations. NNP_1 contains surnames; the most frequent word forms are *Keating*, *Papandreou* and *Kaye*. In contrast, NNP_2

contains company names such as *Sony*, *NBC* and *Keystone*. This explains why the difference in LAS is twice as high for NNPs as on average.

For German we see similar effects and the anticipated correlation with morphology. The 5 determiner subtags, for example, strongly correlate with grammatical case:

	Nom.	Gen.	Dat.	Acc.
ART	**1185**	**636**	**756**	**961**
ART_1	**367**		7	38
ART_2	11	28	**682**	21
ART_3	6	**602**	7	3
ART_4	39		43	**429**
ART_5	**762**	6	17	**470**

Table 4.11: Coocurrences of correct case and predicted POS and latent POS (ART_i).

4.5 Conclusion

We have not only shown that split-merge training for HMMs is a method for increasing performance of generative taggers, but also that the generated latent annotations are linguistically interpretable and can be used to improve dependency parsing. Our best systems improve an English parser from a LAS of 90.34 to 90.57 and a German parser without morphological features from 87.92 to 88.24 and with morphological features from 88.35 to 88.51. Our analysis of the parsing results shows that the major reasons for the improvements are: the separation of POS tags into more and less trustworthy subtags, the creation of POS subtags with higher correlation to certain dependency labels and for German a correlation of tags and morphological features such as case.

Chapter 5

Morphological Tagging with Higher-Order CRFs

Erklärung nach §8 Absatz 4 der Promotionsordnung: This chapter covers work already published at international peer-reviewed conferences. The most relevant publication is Müller et al. (2013). The chapter also covers a small part of Björkelund et al. (2013). The research described in this chapter was carried out in its entirety by myself. The other author(s) of the publication(s) acted as advisor(s) or were responsible for work that was reported in the publication(s), but is not included in this chapter.

In morphological tagging, complex morphological readings are represented as tags in order to allow for the application of standard sequence prediction methods (Section 2.3). For morphologically-rich languages this leads to tagsets of hundreds of tags, which makes the training of higher-order conditional random fields prohibitive. We present an approximated conditional random field using coarse-to-fine decoding and early updating. We show that our implementation MarMoT yields fast and accurate morphological taggers across six languages with different morphological properties and that across languages higher-order models give significant improvements over 1^{st}-order models.

5.1 Introduction

Most natural language processing (NLP) tasks can be better solved if a preprocessor tags each word in the natural language input with a label like "noun, singular" or "verb, past tense" that gives some indication of the syntactic role that the word plays in its context. The most common form of such preprocessing is part-of-speech (POS) tagging. However, for morphologically rich languages, a large subset of the languages of the world, POS tagging in its original form – where labels are syntactic categories with little or no morphological information – does not make much sense. The reason is that POS and morphological properties are mutually dependent, so solving only one task or solving the tasks sequentially is inadequate. The most important dependence

of this type is that POS can be read off morphology in many cases; e.g., the morphological suffix "-iste" is a reliable indicator of the informal second person singular preterite indicative form of a verb in Spanish. In what follows, we use the term "morphological tagging" to refer to "morphological and POS tagging" since morphological tags generally include POS information.

Conditional Random Fields (CRFs) (Lafferty et al., 2001) are arguably one of the best performing sequence prediction models for many Natural Language Processing (NLP) tasks. As we already discussed in Section 2.3, during CRF training forward-backward computations – a form of dynamic programming – dominate the asymptotic runtime. The training and also decoding times thus depend polynomially on the size of the tagset and exponentially on the order of the CRF. This probably explains why CRFs, despite their outstanding accuracy, normally only are applied to tasks with small tagsets such as Named Entity Recognition and Chunking; if they are applied to tasks with bigger tagsets – e.g., to part-of-speech (POS) tagging for English – then they generally are used as 1^{st}-order models.

In this chapter, we demonstrate that fast and accurate CRF training and tagging is possible for large tagsets of even thousands of tags by approximating the CRF objective function using coarse-to-fine decoding (Charniak and Johnson, 2005; Rush and Petrov, 2012). Our pruned CRF model MarMoT has a much smaller runtime than higher-order CRF models and may thus lead to an even broader application of CRFs across NLP tagging tasks.

We use POS tagging and combined POS and morphological (POS+MORPH) tagging to demonstrate the properties and benefits of our approach. Morphological disambiguation is an important preprocessing step for syntactic parsing. It is usually tackled by applying sequence prediction. POS+MORPH tagging is also a good example of a task where CRFs are rarely applied as the tagsets are often so big that even 1^{st}-order dynamic programming is too expensive. A workaround is to restrict the possible tag candidates per position by using either morphological analyzers (MAs), dictionaries or heuristics (Hajič, 2000). In this chapter, however we show that when using pruning (i.e., MarMoT), CRFs can be trained in reasonable time, which makes hard constraints unnecessary.

In this chapter, we run successful experiments on six languages with different morphological properties; we interpret this as evidence that our approach is a general solution to the problem of POS+MORPH tagging. The tagsets in our experiments range from small sizes of 12 to large sizes of up to 1811. We will see that even for the smallest tagset, MarMoT models need only 40% of the training time of standard CRFs. For the bigger tagset sizes we can reduce training times from several days to several hours. We will also show that training higher-order MarMoT models takes only several minutes longer than training 1^{st}-order models and – depending on the language – may lead to substantial accuracy improvements. For example in German POS+MORPH tagging, a 1^{st}-order model (trained in 32 minutes) achieves an accuracy of 88.96 while a 3^{rd}-order model (trained in 35 minutes) achieves an accuracy of 90.60.

The remainder of the chapter is structured as follows: Section 5.3 describes our CRF implementation and the feature set used. Section 5.2 summarizes related work on tagging with CRFs, efficient CRF tagging and coarse-to-fine decoding. Section 5.4 describes experiments on POS tagging and POS+MORPH tagging and Section 5.6 summarizes the main contributions of the chapter.

5.2 Related Work

Morphological tagging (Oflazer and Kuruöz, 1994; Hajič and Hladká, 1998) is the task of assigning a morphological reading to a token in context. The morphological reading consists of features such as case, gender, person and tense and is represented as a single tag. This allows for the application of standard sequence labeling algorithms such as Conditional Random Fields (CRF) (Lafferty et al., 2001), but also puts an upper bound on the accuracy as only readings occurring in the training set can be produced. It is still the standard approach to morphological disambiguation as the number of readings that cannot be produced is usually small. The related work can be divided into branches. Some papers try to exploit certain properties of a language such as Habash and Rambow (2005) and Roth et al. (2008) (Arabic), Yuret and Türe (2006) (Turkish) , Adler and Elhadad (2006) (Hebrew) and Spoustová et al. (2009) and Straková et al. (2014) (Czech). MorphoDiTa, the implementation of the last two papers has been released as open-source.[1] Other papers implement language-independent systems (Hajič, 2000; Smith et al., 2005; Schmid and Laws, 2008; Müller et al., 2013), with our model, we follow a language-independent approach.

Despite their dominance in many NLP areas CRFs are usually not applied to morphological tagging. Smith et al. (2005) use CRFs for morphological tagging, but use a morphological analyzer for candidate selection. They report training times of several days and that they had to use simplified models for Czech. Several methods have been proposed to reduce CRF training times. Stochastic gradient descent can be applied to reduce the training time by a factor of 5 (Tsuruoka et al., 2009) and without drastic losses in accuracy. Lavergne et al. (2010) make use of feature sparsity to significantly speed up training for moderate tagset sizes (< 100) and huge feature spaces. It is unclear if their approach would also work for huge tagsets (> 1000).

Coarse-to-fine decoding has been successfully applied to CYK parsing where full dynamic programming is often intractable when big grammars are used (Charniak and Johnson, 2005). Weiss and Taskar (2010) develop cascades of models of increasing complexity in a framework based on perceptron learning and an explicit trade-off between accuracy and efficiency.

Kaji et al. (2010) propose a modified Viterbi algorithm that is still optimal, but depending on task and especially for big tagsets might be several orders of magnitude faster. While their algorithm can be used to produce fast decoders, there is no such modification for the forward-backward algorithm used during CRF training.

5.3 Methodology

5.3.1 Standard CRF Training

We already discussed standard CRF training in Chapter 2. We know that in a standard CRF, we model our sentences using a globally normalized log-linear model. The probability of a tag sequence $\vec{y}$ given a sentence $\vec{x}$ is then given as:

[1] http://ufal.mff.cuni.cz/morphodita

$$p(\vec{y}|\vec{x}) = \frac{\exp \sum_{t,i} \lambda_i \cdot \phi_i(\vec{y}, \vec{x}, t)}{Z(\vec{\lambda}, \vec{x})}$$
$$Z_{\text{CRF}}(\vec{\lambda}, \vec{x}) = \sum_{\vec{y}} \exp \sum_{t,i} \lambda_i \cdot \phi_i(\vec{y}, \vec{x}, t)$$

where t and i are token and feature indexes, ϕ_i is a feature function, λ_i is a feature weight and Z is a normalization constant. During training the feature weights λ are set to maximize the conditional log-likelihood of the training data D:

$$\text{ll}_D(\vec{\lambda}) = \sum_{(\vec{x},\vec{y}) \in D} \log\, p(\vec{y}|\vec{x}, \vec{\lambda})$$

In order to use numerical optimization we have to calculate the gradient of the log-likelihood, which is a vector of partial derivatives $\partial ll_D(\vec{\lambda})/\partial \lambda_i$. For a training sentence $\vec{x}, \vec{y}$ and a token index t the derivative with respect to feature i is given by:

$$\phi_i(\vec{y}, \vec{x}, t) - \sum_{\vec{y}'} \phi_i(\vec{y}', \vec{x}, t)\, p(\vec{y}'|\vec{x}, \vec{\lambda})$$

This is the difference between the empirical feature count in the training data and the estimated count in the current model $\vec{\lambda}$. For a 1^{st}-order model, we can replace the expensive sum over all possible tag sequences $\vec{y}'$ by a sum over all pairs of tags:

$$\phi_i(y_t, y_{t+1}, \vec{x}, t) - \sum_{y,y'} \phi_i(y, y', \vec{x}, t)\, p(y, y'|\vec{x}, \vec{\lambda})$$

The probability of a tag pair $p(y, y'|\vec{x}, \vec{\lambda})$ can then be calculated efficiently using the forward-backward algorithm. If we further reduce the complexity of the model to a 0-order model, we obtain simple maximum entropy model updates:

$$\phi_i(y_t, \vec{x}, t) - \sum_{y} \phi_i(y, \vec{x}, t)\, p(y|\vec{x}, \vec{\lambda})$$

5.3.2 Pruned CRF Training

As we discussed in the introduction, we want to decode sentences by applying a variant of coarse-to-fine tagging. Naively, to later tag with n^{th}-order accuracy we would train a series of n CRFs of increasing order. We would then use the CRF of order $n-1$ to restrict the input of the CRF of order n. In this chapter we approximate this approach, but do so while training only

one integrated model. This way we can save both memory (by sharing feature weights between different models) and training time (by saving lower-order updates).

The main idea of our approach is to create increasingly complex lattices and to filter candidate states at every step to prevent a polynomial increase in lattice size. The first step is to create a 0-order lattice, which as discussed above, is identical to a series of independent local maximum entropy models $p(y|\vec{x}, t)$. The models base their prediction on the current word x_t and the immediate lexical context. We then calculate the posterior probabilities and remove states y with $p(y|\vec{x}, t) < \tau_0$ from the lattice, where τ_0 is a parameter. The resulting reduced lattice is similar to what we would obtain using candidate selection based on an MA.

We can now create a 1^{st}-order lattice by adding transitions to the pruned lattice and pruning with threshold τ_1. The only difference to 0-order pruning is that we now have to run forward-backward to calculate the probabilities $p(y|\vec{x}, t)$. Note that in theory we could also apply the pruning to transition probabilities of the form $p(y, y'|\vec{x}, t)$; however, this does not seem to yield more accurate models and is less efficient than state pruning.

For higher-order lattices we merge pairs of states into new states, add transitions and prune with threshold τ_i.

We train the model using l_1-regularized Stochastic Gradient Descent (SGD) (Tsuruoka et al., 2009). We would like to create a cascade of increasingly complex lattices and update the weight vector with the gradient of the last lattice. The updates, however, are undefined if the gold sequence is pruned from the lattice. A solution would be to simply reinsert the gold sequence, but this yields poor results as the model never learns to keep the gold sequence in the lower-order lattices. As an alternative we perform the gradient update with the highest lattice still containing the gold sequence. This approach is similar to "early updating" (Collins and Roark, 2004) in perceptron learning, where during beam search an update with the highest scoring partial hypothesis is performed whenever the gold candidate falls out of the beam. Intuitively, we are trying to optimize an n^{th}-order CRF objective function, but apply small lower-order corrections to the weight vector when necessary to keep the gold candidate in the lattice. Algorithm 5.1 illustrates the lattice generation process. The lattice generation during decoding is identical, except that we always return a lattice of the highest order n.

The savings in training time of this integrated approach are large; e.g., training a maximum entropy model over a tagset of roughly 1800 tags and more than half a million instances is slow as we have to apply 1800 weight vector updates for every token in the training set and every SGD iteration. In the integrated model we only have to apply 1800 updates when we lose the gold sequence during filtering. Thus, in our implementation training a 0-order model for Czech takes roughly twice as long as training a 1^{st}-order model.

Algorithm 5.1 Lattice generation during training

```
1:  function GetSumLattice(sentence, τ⃗)
2:      gold-tags ← getTags(sentence)
3:      candidates ← getAllCandidates(sentence)
4:      lattice ← ZeroOrderLattice(candidates)
5:      for i = 1 → n do
6:          candidates ← lattice.prune(τ_{i−1})
7:          if gold-tags ∉ candidates then
8:              return lattice
9:          end if
10:         if i > 1 then
11:             candidates ← mergeStates(candidates)
12:         end if
13:         candidates ← addTransitions(candidates)
14:         lattice ← SequenceLattice(candidates, i)
15:     end for
16:     return lattice
17: end function
```

5.3.3 Threshold Estimation

Our approach would not work if we were to set the parameters τ_i to fixed predetermined values; e.g., the τ_i depend on the size of the tagset and should be adapted during training as we start the training with a uniform model that becomes more specific. We therefore set the τ_i by *specifying μ_i, the average number of tags per position that should remain in the lattice after pruning*. This also guarantees stable lattice sizes and thus stable training times. We achieve stable average number of tags per position by setting the τ_i dynamically during training: we measure the real average number of candidates per position $\hat{\mu}_i$ and apply corrections after processing a certain fraction of the sentences of the training set. The updates are of the form:

$$\tau_i = \begin{cases} (1.0 + \epsilon) \cdot \tau_i & \text{if } \hat{\mu}_i < \mu_i \\ (1.0 - \epsilon) \cdot \tau_i & \text{if } \hat{\mu}_i > \mu_i \end{cases}$$

where we use $\epsilon = 0.1$. Figure 5.1 shows an example training run for German with $\mu_0 = 4$. Here the 0-order lattice reduces the number of tags per position from 681 to 4 losing roughly 15% of the gold sequences of the development set, which means that for 85% of the sentences the correct candidate is still in the lattice. This corresponds to more than 99% of the tokens. We can also see that after two iterations only a very small number of 0-order updates have to be performed.

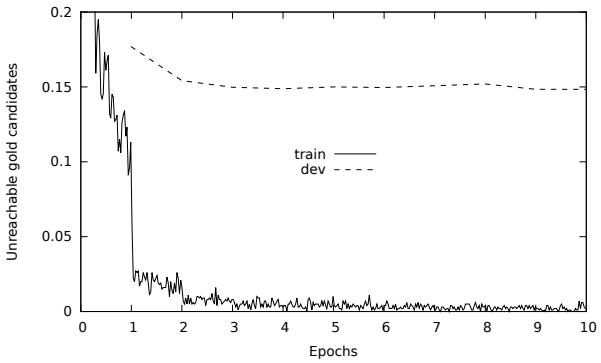

Figure 5.1: Example training run of a pruned 1^{st}-order model on German showing the fraction of pruned gold sequences (= sentences) during training for training (train) and development sets (dev).

5.3.4 Tag Decomposition

As we discussed before for the very large POS+MORPH tagsets, most of the decoding time is spent on the 0-order level. To decrease the number of tag candidates in the 0-order model, we decode in two steps by separating the fully specified tag into a coarse-grained part-of-speech (POS) tag and a fine-grained MORPH tag containing the morphological features. We then first build a lattice over POS candidates and apply our pruning strategy. In a second step we expand the remaining POS tags into all the combinations with MORPH tags that were seen in the training set. We thus build a sequence of lattices of both increasing order and increasing tag complexity.

5.3.5 Feature Set

We use the features of Ratnaparkhi (1996) and Manning (2011): the current, preceding and succeeding words as unigrams and bigrams and for rare words prefixes and suffixes up to length 10, and the occurrence of capital characters, digits and special characters. We define a rare word as a word with training set frequency ≤ 10. We concatenate every feature with the POS and MORPH tag and every morphological feature. For example, for the word "der", the POS tag `art` (article) and the MORPH tag `gen|sg|fem` (genitive, singular, feminine) we get the following features for the current word template: `der+art`, `der+gen|sg|fem`, `der+gen`, `der+sg` and `der+fem`.

We also use an additional binary feature, which indicates whether the current word has been seen with the current tag or – if the word is rare – whether the tag is in a set of open tag classes. The open tag classes are estimated by 10-fold cross-validation on the training set: We first use the folds to estimate how often a tag is seen with an unknown word. We then consider tags with a relative frequency $\geq 10^{-4}$ as open tag classes. While this is a heuristic, it is safer to use a "soft" heuristic as a feature in the lattice than a hard constraint.

For some experiments we also use the output of a morphological analyzer (MA). In that case we simply use every analysis of the MA as a simple nominal feature. This approach is

attractive because it does not require the output of the MA and the annotation of the treebank to be identical; in fact, it can even be used if treebank annotation and MA use completely different features.

Manning (2011) reports shape features to be helpful for POS tagging. A limitation of their shapes is that they make certain assumptions about typological properties; thus, the features are not language independent. We propose to induce word shapes based on POS distributions. The method is similar to Schmid (1994), but allows for arbitrary features instead of only suffixes. The principal idea is to train a decision tree and to use the leaves as word shapes. For this purpose we represent every word form as a binary feature vector. Our features include the length of the word, whether the word contains a certain character or any digit, lowercase or uppercase letter, whether the lowercased word form occurred in the training corpus and whether one of the 10 leading or trailing characters is a particular character. During the decision tree training these features are concatenated to form signatures. As an example: applied to the English Penn Treebank the method generates a signature that groups words without uppercase characters and digits that end in "ness" into one signature; examples include "creditworthiness", "comprehensiveness", "fickleness", "heavy-handedness" and "warmheartedness". We apply the induction method to all rare words in the training corpus and use information gain as the splitting criterion. We further constrain the split nodes to contain at least 50 words and stop when the number of leaf nodes reaches k, where we set $k = 100$. The learned signatures are used as a replacement for rare word forms. The shape features are evaluated in the next section.

Because the weight vector dimensionality is high for large tagsets and productive languages, we use a hash kernel (Shi et al., 2009) to keep the dimensionality constant. We investigate the impact of the dimensionality in the next section. Details of our implementation are explained in Appendix A.

5.4 Experiments

We run POS+MORPH tagging experiments on Arabic (ar), Czech (cs), Spanish (es), German (de) and Hungarian (hu). Table 5.1 shows the type-token (T/T) ratio, the average number of tags of every word form that occurs more than once in the training set ($\overline{A}$) and the number of tags of the most ambiguous word form ($\hat{A}$).

	T/T	$\overline{A}$	$\hat{A}$
Arabic	0.06	2.06	17
Czech	0.13	1.64	23
Spanish	0.09	1.14	9
German	0.11	2.15	44
Hungarian	0.11	1.11	10

Table 5.1: Type-token (T/T) ratio, average number of tags per word form ($\overline{A}$) and the number of tags of the most ambiguous word form ($\hat{A}$)

Arabic is a Semitic language with nonconcatenative morphology. An additional difficulty is that vowels are often not written in Arabic script. This introduces a high number of ambiguities; on the other hand it reduces the type-token ratio, which generally makes learning easier. In this chapter, we work with the transliteration of Arabic provided in the Penn Arabic Treebank. Czech is a highly inflecting Slavic language with a large number of morphological features. Spanish is a Romance language. Based on the statistics above we can see that it has few POS+MORPH ambiguities. It is also the language with the smallest tagset and the only language in our setup that – with a few exceptions – does not mark case. German is a Germanic language and – based on the statistics above – the language with the most ambiguous morphology. The reason is that it only has a small number of inflectional suffixes. The total number of nominal inflectional suffixes for example is five. A good example for a highly ambiguous suffix is "en", which is a marker for infinitive verb forms, for the 1^{st} and 3^{rd} person plural and for the polite 2^{nd} person singular. Additionally, it marks plural nouns of all cases and singular nouns in genitive, dative and accusative case.

Hungarian is a Finno-Ugric language with an agglutinative morphology; this results in a high type-token ratio, but also the lowest level of word form ambiguity among the selected languages.

POS tagging experiments are run on all the languages above and also on English.

5.4.1 Resources

For Arabic we use the Penn Arabic Treebank (Maamouri et al., 2004), parts 1–3 in their latest versions (LDC2010T08, LDC2010T13, LDC2011T09). As training set we use parts 1 and 2 and part 3 up to section `ANN20020815.0083`. All consecutive sections up to `ANN20021015.0096` are used as development set and the remainder as test set. We use the unvocalized and pretokenized transliterations as input. For Czech and Spanish, we use the CoNLL 2009 data sets (Hajič et al., 2009); for German, the TIGER treebank (Brants et al., 2002) with the split from Fraser et al. (2013); for Hungarian, the Szeged treebank (Csendes et al., 2005) with the split from Farkas et al. (2012). For English we use the Penn Treebank (Marcus et al., 1993) with the split from Toutanova et al. (2003).

We also compute the possible POS+MORPH tags for every word using MAs. For Arabic we use the AraMorph reimplementation of Buckwalter (2002), for Czech the "free" morphology (Hajič, 2001), for Spanish Freeling (Padró and Stanilovsky, 2012), for German DMOR (Schiller (1995)) and for Hungarian Magyarlanc 2.0 (Zsibrita et al., 2013).

5.4.2 Setup

To compare the training and decoding times we run all experiments on the same test machine, which features two Hexa-Core Intel Xeon X5680 CPUs with 3,33 GHz and 6 cores each and 144 GB of memory. The baseline tagger and our MarMoT implementation are run single threaded. Note that our tagger might actually use more than one core because the Java garbage collection is run in parallel. The taggers are implemented in different programming languages and with different degrees of optimization; still, the run times are indicative of comparative performance to be expected in practice.

Our Java implementation is always run with 10 SGD iterations and a regularization parameter of 0.1, which for German was the optimal value out of $\{0, 0.01, 0.1, 1.0\}$. We follow Tsuruoka et al. (2009) in our implementation of SGD and shuffle the training set between epochs. All numbers shown are averages over 5 independent runs, where not noted otherwise, we use $\mu_0 = 4$, $\mu_1 = 2$ and $\mu_2 = 1.5$. We found that higher values do not consistently increase performance on the development set, but result in much higher training times.

Language	Sentences	Tokens	POS Tags	MORPH Tags	POS+MORPH Tags	OOV Rate
ar (Arabic)	15,760	614,050	38	516	516	4.58%
cs (Czech)	38,727	652,544	12	1,811	1,811	8.58%
en (English)	38,219	912,344	45		45	3.34%
es (Spanish)	14,329	427,442	12	264	303	6.47%
de (German)	40,472	719,530	54	255	681	7.64%
hu (Hungarian)	61,034	1,116,722	57	1,028	1,071	10.71%

Table 5.2: Training set statistics. Out-Of-Vocabulary (OOV) rate is regarding the development sets.

5.4.3 POS Experiments

In a first experiment we evaluate the speed and accuracy of CRFs and MarMoT models on the POS tagsets. As shown in Table 5.2 the tagset sizes range from 12 for Czech and Spanish to 54 and 57 for German and Hungarian, with Arabic (38) and English (45) in between. The results of our experiments are given in Table 5.3. For the 1^{st}-order models, we observe speed-ups in training time from 2.3 to 31 at no loss in accuracy. For all languages, training pruned higher-order models is faster than training unpruned 1^{st}-order models and yields more accurate models. Accuracy improvements range from 0.08 for Hungarian to 0.25 for German. We can conclude that for small and medium tagset sizes MarMoT gives substantial improvements in both training and decoding speed (cf. Table 5.4) and thus allows for higher-order tagging, which for all languages leads to significant accuracy improvements. Throughout the chapter we establish significance by running approximate randomization tests on sentences (Yeh, 2000).

	ar		cs		es		de		hu		en	
n	TT	ACC	TT	ACC	TT	ACC	TT	ACC	TT	ACC	TT	ACC
CRF 1	106	96.21	10	98.95	7	98.51	234	97.69	374	97.63	154	97.05
MarMoT 1	5	96.21	4	98.96	3	98.52	7	97.70	12	97.64	5	97.07
MarMoT 2	6	96.43*	5	99.01*	3	98.65*	9	97.91*	13	97.71*	6	97.21*
MarMoT 3	6	96.43*	6	99.03*	4	98.66*	9	97.94*	14	97.69	6	97.19*

Table 5.3: POS tagging experiments with pruned (MarMoT) and unpruned CRFs with different orders n. For every language the training time in minutes (TT) and the POS accuracy (ACC) are given. * indicates models significantly better than CRF (first line).

	n	ar	cs	es	de	hu	en
CRF	1	101	2041	1095	119	96	219
MarMoT	1	1150	2746	1377	1851	1593	2647
MarMoT	2	762	1720	1175	1552	1207	1715
MarMoT	3	604	1617	861	1375	1042	1419

Table 5.4: Decoding speed at order n for POS tagging. Speed is measured in sentences / second.

5.4.4 POS+MORPH Oracle Experiments

		ar	cs	es	de	hu
1	Oracle $\mu_0 = 4$	90.97	92.59	97.91	89.33	96.48
2	Model $\mu_0 = 4$	90.90	92.45*	97.95	88.96*	96.47
3	Model $\mu_0 = 8$	90.89	92.48*	97.94	88.94*	96.47

Table 5.5: Accuracies for 1^{st}-order models with and without oracle pruning. * indicates models significantly worse than the oracle model.

Ideally, for the full POS+MORPH tagset we would also compare our results to an unpruned CRF, but our implementation turned out to be too slow to do the required number of experiments. For German, the model processed ≈ 0.1 sentences per second during training; so running 10 SGD iterations on the 40,472 sentences would take more than a month. We therefore compare our model against models that perform oracle pruning, which means we perform standard pruning, but always keep the gold candidate in the lattice. The oracle pruning is applied during training and testing on the development set. The oracle model performance is thus an upper bound for the performance of an unpruned CRF.

The most interesting pruning step happens at the 0-order level when we reduce from hundreds of candidates to just a couple. Table 5.5 shows the results for 1^{st}-order CRFs.

We can roughly group the five languages into three groups: for Spanish and Hungarian the damage is negligible, for Arabic we see a small decrease of 0.07 and only for Czech and German we observe considerable differences of 0.14 and 0.37. Surprisingly, doubling the number of candidates per position does not lead to significant improvements.

We can conclude that except for Czech and German losses due to pruning are insignificant.

5.4.5 POS+MORPH Higher-Order Experiments

One argument for MarMoT is that while it might be less accurate than standard CRFs it allows to train higher-order models, which in turn are more accurate than their standard lower-order counterparts. In this section, we investigate how big the improvements of higher-order models are. The results are given in Table 5.6:

n	ar	cs	es	de	hu
1	90.90	92.45	97.95	88.96	96.47
2	91.86*	93.06*	98.01	90.27*	96.57*
3	91.88*	92.97*	97.87	90.60*	96.50

Table 5.6: POS+MORPH accuracies for models of different order n.

We see that 2^{nd}-order models give improvements for all languages. For Spanish and Hungarian we see minor improvements ≤ 0.1.

For Czech we see a moderate improvement of 0.61 and for Arabic and German we observe substantial improvements of 0.96 and 1.31. An analysis on the development set revealed that for all three languages, case is the morphological feature that benefits most from higher-order models. A possible explanation is that case has a high correlation with syntactic relations and is thus affected by long-distance dependencies.

German is the only language where fourgram models give an additional improvement over trigram models. The reason seem to be sentences with long-range dependencies, e.g., "Die Rebellen haben kein Lösegeld verlangt" (The rebels have not demanded any ransom); "verlangt" (demanded) is a past participle that is separated from the auxiliary verb "haben" (have). The 2^{nd}-order model does not consider enough context and misclassifies "verlangt" as a finite verb form, while the 3^{rd}-order model tags it correctly.

We can also conclude that the improvements for higher-order models are always higher than the loss we estimated in the oracle experiments. More precisely we see that if a language has a low number of word form ambiguities (e.g., Hungarian) we observe a small loss during 0-order pruning, but we also have to expect less of an improvement when increasing the order of the model. For languages with a high number of word form ambiguities (e.g., German) we must anticipate some loss during 0-order pruning, but we also see substantial benefits for higher-order models.

Surprisingly, we found that higher-order MarMoT models can also avoid the pruning errors of lower-order models. Here is an example from the German data. The word "Januar" (January) is ambiguous: in the training set, it occurs 108 times as dative, 9 times as accusative and only 5 times as nominative. The development set contains 48 nominative instances of "Januar" in datelines at the end of news articles, e.g., "TEL AVIV, 3. Januar". For these 48 occurrences, (i) the oracle model in Table 5.5 selects the *correct* case nominative, (ii) the 1^{st}-order MarMoT model selects the *incorrect* case accusative, and (iii) the 2^{nd}- and 3^{rd}-order models select – unlike the 1^{st}-order model – the *correct* case nominative. Our interpretation is that the correct nominative reading is pruned from the 0-order lattice. However, the higher-order models can put less weight on 0-order features as they have access to more context to disambiguate the sequence. The lower weights of order-0 result in a more uniform posterior distribution and the nominative reading is not pruned from the lattice.

5.4.6 Experiments with Morphological Analyzers

In this section we compare the improvements of higher-order models when used with MAs. The results are given in Table 5.7:

	n	ar	cs	es	de	hu
	1	90.90^{-}	92.45^{-}	97.95^{-}	88.96^{-}	96.47^{-}
	2	91.86^{+}	93.06	98.01^{-}	90.27^{+}	96.57^{-}
	3	91.88^{+}	92.97^{-}	97.87^{-}	90.60^{+}	96.50^{-}
MA	1	91.22	93.21	98.27	89.82	97.28
MA	2	92.16^{+}	93.87^{+}	98.37^{+}	91.31^{+}	97.51^{+}
MA	3	92.14^{+}	93.88^{+}	98.28	91.65^{+}	97.48^{+}

Table 5.7: POS+MORPH accuracy for models of different orders n and models with and without morphological analyzers (MA). +/- indicate models significantly better/worse than MA 1.

Plus and minus indicate models that are significantly better or worse than MA 1. We can see that the improvements due to higher-order models are orthogonal to the improvements due to MAs for all languages. This was to be expected as MAs provide additional lexical knowledge while higher-order models provide additional information about the context. For Arabic and German the improvements of higher-order models are bigger than the improvements due to MAs.

5.4.7 Comparison with Baselines

We use the following baselines: SVMTool (Giménez and Màrquez, 2004), an SVM-based discriminative tagger; RFTagger (Schmid and Laws, 2008), an n-gram Hidden Markov Model (HMM) tagger developed for POS+MORPH tagging; Morfette (Chrupala et al., 2008), an averaged perceptron with beam search decoder; CRFSuite (Okazaki, 2007), a fast CRF implementation; and the Stanford Tagger (Toutanova et al., 2003), a bidirectional Maximum Entropy Markov Model. For POS+MORPH tagging, all baselines are trained on the concatenation of POS tag and MORPH tag. We run SVMTool with the standard feature set and the optimal c-values $\in \{0.1, 1, 10\}$. Morfette is run with the default options. For CRFSuite we use l_2-regularized SGD training. We use the optimal regularization parameter $\in \{0.01, 0.1, 1.0\}$ and stop after 30 iterations where we reach a relative improvement in regularized likelihood of at most 0.01 for all languages. The feature set is identical to our model except for some restrictions: we only use concatenations with the full tag and we do not use the binary feature that indicates whether a word-tag combination has been observed. We also had to restrict the combinations of tag and features to those observed in the training set.[2] Otherwise the memory requirements would exceed the memory of our test machine (144 GB) for Czech and Hungarian. The Stanford Tagger is used as a bidirectional 2^{nd}-order model and trained using OWL-BFGS (Andrew and Gao, 2007), a modified version of BFGS for l1-regularized objective functions. For Arabic, German and En-

[2]We set the CRFSuite option $\mathrm{possible\_states} = 0$

glish we use the language specific feature sets and for the other languages the English feature set.

Development set results for POS tagging are shown in Table 5.8.

	ar		cs		es		de		hu		en	
	TT	ACC	TT	ACC	TT	ACC	TT	ACC	TT	ACC	TT	ACC
SVMTool	178	96.39	935	98.94	64	98.42	899	97.29	2653	97.42	253	97.09
Morfette	9	95.91	6	99.00	3	98.43	16	97.28	30	97.53	17	96.85
CRFSuite	4	96.20	2	99.02	2	98.40	8	97.57	15	97.48	8	96.80
Stanford	29	95.98	8	**99.08**	7	98.53	51	97.70	40	97.53	65	**97.24**
MarMoT 1	5	96.21*	4	98.96*	3	98.52	7	97.70	12	97.64*	5	97.07*
MarMoT 2	6	**96.43**	5	99.01*	3	98.65*	9	97.91*	13	**97.71***	6	97.21
MarMoT 3	6	**96.43**	6	99.03	4	**98.66***	9	**97.94***	14	97.69*	6	97.19

Table 5.8: Development results for POS tagging. Given are training times in minutes (TT) and accuracies (ACC). Best baseline results are underlined and the overall best results bold. * indicates a significant difference (positive or negative) between the best baseline and a MarMoT model.

We can observe that Morfette, CRFSuite and the MarMoT models for different orders have training times in the same order of magnitude. For Arabic, Czech and English, the MarMoT accuracy is comparable to the best baseline models. For the other languages we see improvements of 0.13 for Spanish, 0.18 for Hungarian and 0.24 for German. Evaluation on the test set confirms these results, see Table 5.9.[3] We can conclude that the training times of pruned high-order models are on a par with the fastest discriminative baseline taggers. Furthermore, we see comparable accuracies for Arabic and Czech and significant improvements for Spanish, Hungarian and German.

	ar	cs	es	de	hu	en
SVMTool	**96.19**	98.82	98.44	96.44	97.32	97.12
Morfette	95.55	98.91	98.41	96.68	97.28	96.89
CRFSuite	95.97	98.91	98.40	96.82	97.32	96.94
Stanford	95.75	**98.99**	98.50	97.09	97.32	**97.28**
MarMoT 1	96.03*	98.83*	98.46	97.11	97.44*	97.09*
MarMoT 2	96.11	98.88*	**98.66***	97.36*	**97.50***	97.23
MarMoT 3	96.14	98.87*	**98.66***	**97.44***	97.49*	97.19*
Manning (2011)						97.29
Shen et al. (2007)						97.33

Table 5.9: Test results for POS tagging. Best baseline results are underlined and the overall best results bold. * indicates a significant difference between the best baseline and a MarMoT model.

[3] Giménez and Màrquez (2004) report an accuracy of 97.16 instead of 97.12 for SVMTool for English and Manning (2011) an accuracy of 97.29 instead of 97.28 for the Stanford tagger.

The POS+MORPH tagging development set results are presented in Table 5.10.

	ar		cs		es		de		hu	
	TT	ACC	TT	ACC	TT	ACC	TT	ACC	TT	ACC
SVMTool	454	89.91	2454	89.91	64	97.63	1649	85.98	3697	95.61
RFTagger	4	89.09	3	90.38	1	97.44	5	87.10	10	95.06
Morfette	132	89.97	539	90.37	63	97.71	286	85.90	540	95.99
CRFSuite	309	89.33	9274	91.10	69	97.53	1295	87.78	5467	95.95
MarMoT 1	22	90.90*	301	92.45*	25	97.95*	32	88.96*	230	96.47*
MarMoT 2	26	91.86*	318	**93.06***	32	**98.01***	37	90.27*	242	**96.57***
MarMoT 3	26	**91.88***	318	92.97*	35	97.87*	37	**90.60***	241	96.50*

Table 5.10: Development results for POS+MORPH tagging. Given are training times in minutes (TT) and accuracies (ACC). Best baseline results are underlined and the overall best results bold. * indicates a significant difference between the best baseline and a MarMoT model.

Morfette is the fastest discriminative baseline tagger. In comparison with Morfette the speed up for 3^{rd}-order models lies between 1.7 for Czech and 5 for Arabic. Morfette gives the best baseline results for Arabic, Spanish and Hungarian and CRFSuite for Czech and German. The accuracy improvements of the best MarMoT models over the best baseline models range from 0.27 for Spanish over 0.58 for Hungarian, 1.91 for Arabic, 1.96 for Czech to 2.82 for German. The test set experiments in Table 5.11 confirm these results.

	ar	cs	es	de	hu
SVMTool	89.58	89.62	97.56	83.42	95.57
RFTagger	88.76	90.43	97.35	84.28	94.99
Morfette	89.62	90.01	97.58	83.48	95.79
CRFSuite	89.05	90.97	97.60	85.68	95.82
MarMoT 1	90.32*	92.31*	97.82*	86.92*	96.22*
MarMoT 2	**91.29***	92.94*	**97.93***	88.48*	**96.34***
MarMoT 3	91.22*	**92.99***	97.82*	**88.58***	96.29*

Table 5.11: Test results for POS+MORPH tagging. Best baseline results are underlined and the overall best results bold. * indicates a significant difference between the best baseline and a MarMoT model.

5.4.8 Weight Vector Size

In this section we investigate the impact of weight vector dimensionality on model performance. The reason for this is that the large tagsets can give rise to huge feature vectors as we have a weight for every possible combination of feature and tag. In the worst case setup for Czech we observe $\approx 10^6$ feature values and 1800 POS+MORPH tags. Thus, we need a weight vector of length $1.8 \cdot 10^9$. For the other languages we also obtain theoretical vector lengths in the range of 10^8 to 10^9. Assuming 8 byte floating point precision and that for implementing the l_1-regularized

SGD we need a second copy of the weight vector, we need at least 26 GB to store the weight vectors. As described in Section 5.3, we therefore use a hash kernel to decrease the length of the weight vector. In our experiments we use a vector length of 10^7. Thus for Czech we have to expect more than 180 collisions per position. In the following experiment (Table 5.12) we show how this effects model performance.

$\|w\|$	ar	cs	es	de	hu
10^5	89.07	88.03	97.59	85.22	94.34
10^7	90.90*	92.45*	97.95*	88.96*	96.47*
10^9	90.95*	92.54*	97.93*	88.98*	96.49*

Table 5.12: POS+MORPH accuracies at different weight vector dimensions.

The results show that we can reduce the size of the vector by a factor of 100 and lose at most 0.09 in accuracy. All 10^7 and 10^9 models significantly outperform the 10^5 models, but no 10^9 is significantly better than a 10^7 model.

5.4.9 Word Shapes

We evaluate our shape features described in Section 5.3 on a POS tagging task. The Table 5.13 shows results for models with and without shape features.

	ar	cs	es	de	hu	en
–	96.11	98.92	98.45	97.65	97.60	97.05
+	96.21*	98.96	98.52*	97.70*	97.64*	97.07

Table 5.13: POS tagging accuracies for 1^{st}-order models with (+) and without shape features (–).

The shape features give small, but consistent improvements across all languages. For Arabic, Spanish, German and Hungarian we observe significant ("*") improvements.

5.5 An Application to Constituency Parsing

In this section we use our CRF model to improve the unknown word handling of a state-of-the-art phrase structure parser (Petrov et al., 2006). The experiments presented in this section were part of a contribution (Björkelund et al., 2013) to the Shared Task (ST) for (Statistical) Parsing of Morphologically Rich Languages (SPMRL) 2013 (Seddah et al., 2013) and lead to the best performing constituency parsing system in the ST.

Probabilistic Context-Free Grammars with Latent Annotations (PCFG-LAs) (Petrov et al., 2006) are arguably the state of the art in multi-lingual constituency parsing. PCFG-LA avoid the need of manually annotating a grammar. During training an annotated grammar is automatically

built by splitting nonterminals, refining parameters using EM-training and reversing splits that only cause small increases in likelihood.

In this study we use the Berkeley Parser (Petrov et al., 2006), the reference implementation of PCFG-LAs. The Berkeley Parser has state-of-the art results for many languages (such as English and German), but uses a simple signature-based unknown word handling that is not sufficient for many MRLs.

We improve the parser by replacing rare words (frequency < 20) with the morphological reading assigned by our CRF tagger. The intuition is that our discriminative tagger has a more sophisticated unknown word treatment than the Berkeley parser, taking for example prefixes, suffixes and the immediate lexical context into account. Furthermore, the morphological tag contains most of the necessary syntactic information. An exception, for instance, might be the semantic information needed to disambiguate prepositional attachment. We think that replacing rare words by tags has an advantage over the common practice of constraining the pre-terminal layer of the parser, because the parser can still decide to assign a different tag, for example in cases were the tagger produces errors due to long-distance dependencies.

We ran experiments on the nine shared task languages: Arabic, Basque (eu), French (fr), German, Hebrew (he), Hungarian, Korean (ko), Polish (pl) and Swedish (sv). We preprocessed the ST treebank trees by removing the morphological annotation of the POS tags and the function labels of all nonterminals. We also reduced the 177 compositional Korean POS tags to their first atomic tag, which results in a POS tagset of 9 tags.

The data sets result in the PARSEVAL scores given in Table 5.14 (Berkeley). As we already discussed, the Berkeley parser only implements a simple signature-based unknown word model that seems to be ineffective for some of the languages, especially Basque and Korean.

The replacement method (replaced) yields improvements for all languages except for French where we observe a drop of 0.06. The improvements range from 0.46 for Arabic to 1.02 for Swedish, 3.1 for Polish and more than 10 for Basque and Korean.

	ar	eu	fr	de	he	hu	ko	pl	sv
Berkeley	78.24	69.17	**79.74**	81.74	87.83	83.90	70.97	84.11	74.50
Replaced	**78.70**	**84.33**	79.68	**82.74**	**89.55**	**89.08**	**82.84**	**87.12**	**75.52**
Δ	0.46	15.16	- 0.06	1.00	1.72	5.65	11.87	3.01	1.02

Table 5.14: PARSEVAL scores on the SPMRL-2013 development set for the baseline model (Berkeley) and a model that replaces rare word forms by morphological tags (replaced).

The used frequency threshold of 20 results in relatively high token replacement rates of 18% (French) to 55% (Polish) (cf. Table 5.15). This corresponds to huge reductions of the initial vocabulary. For example for Korean the vocabulary is reduced from more than 85 thousand to 1462. Surprisingly, a lower replacement threshold (10) did not yield consistent improvements. The resulting parser is thus almost delexicalized.

	$\lvert V_i \rvert$	$\lvert V_r \rvert$	ρ
Arabic	36,906	3,506	18.99
Basque	25,136	418	50.48
French	27,470	2,096	<u>17.80</u>
German	77,220	**3,683**	24.59
Hebrew	15,975	653	32.51
Hungarian	40,782	707	45.71
Korean	**85,671**	1,462	53.65
Polish	21,793	<u>219</u>	**55.10**
Swedish	<u>14,097</u>	381	39.59

Table 5.15: Size of the initial vocabulary V_i and the vocabulary after replacement V_r and the token replacement rate ρ. The maximum and minimum in each column are bold and underlined respectively.

5.6 Conclusion

We presented a pruned CRF model for very large tagsets. The model is based on coarse-to-fine decoding and stochastic gradient descent training with early updating. We showed that for moderate tagset sizes of ≈ 50, our implementation MarMoT gives significant speed-ups over a standard CRF with negligible losses in accuracy. Furthermore, we showed that training and tagging for approximated trigram and fourgram models is still faster than standard 1^{st}-order tagging, but yields significant improvements in accuracy.

In oracle experiments with POS+MORPH tagsets we demonstrated that the losses due to our approximation depend on the word level ambiguity of the respective language and are moderate (≤ 0.14) except for German where we observed a loss of 0.37. We also showed that higher order tagging – which is prohibitive for standard CRF implementations – yields significant improvements over unpruned 1^{st}-order models. Analogous to the oracle experiments we observed big improvements for languages with a high level of POS+MORPH ambiguity such as German and smaller improvements for languages with less ambiguity such as Hungarian and Spanish.

In experiments on the SPMRL-ST 2013 data sets we showed that the model can be used to improve the results of a state-of-the art parser (Petrov et al., 2006) on eight languages, where we see absolute improvements of more than 10 point in F-score for Basque and Korean.

Chapter 6

Morphological Tagging with Word Representations

Erklärung nach §8 Absatz 4 der Promotionsordnung: This chapter covers work already published at international peer-reviewed conferences. The most relevant publication is Müller and Schütze (2015)[1]. The research described in this chapter was carried out in its entirety by myself. The other author(s) of the publication(s) acted as advisor(s) or were responsible for work that was reported in the publication(s), but is not included in this chapter.

In this chapter, we present a comparative investigation of word representations for part-of-speech and morphological tagging (POS+MORPH tagging), focusing on scenarios with considerable differences between training and test data where a *robust* approach is necessary. Instead of adapting the model towards a specific domain we aim to build a robust model across domains. To this end, we developed a test suite for robust tagging consisting of six morphologically rich languages and different domains. In extensive experiments, we find that representations similar to Brown clusters perform best for part-of-speech tagging and that word representations based on linguistic morphological analyzers perform best for morphological tagging.

6.1 Introduction

The importance of morphological tagging as part of the computational linguistics processing pipeline motivated us to conduct the research reported in this chapter. The specific setting that we address is increasingly recognized as the setting in which most practical NLP takes place: we look at scenarios with considerable differences between the training data and the application data, i.e., between the data that the tagger is trained on and the data that it is applied to. This type of scenario is frequent because of the great diversity and variability of natural language and because of the high cost of annotation – which makes it impossible to create large training sets for each new domain. For this reason, we address morphological tagging in a setting in which training and application data differ.

The most common approach to this setting is domain adaptation. Domain adaptation has been demonstrated to have good performance in scenarios with differently distributed training/test data. However, it has two disadvantages. First, it requires the availability of data from the target domain. Second, we need to do some extra work in domain adaptation – consisting of taking target domain data and using it to adapt our NLP system to the target domain – and we end up with a number of different versions of our NLP system, each an adaptation for a different domain. The extra work required and the proliferation of different versions increase the possibility of error and generally increase the complexity of deploying NLP technology.

Similar to other recent work (Zhang and Wang, 2009), we therefore take an approach that is different from domain adaptation. We build a system that is *robust across domains without any modification*. As a result, no extra work is required when the system is applied to a new domain: there is only one system and we can use it for all domains.

The key to making NLP components robust across domains is *the use of powerful domain-independent representations for words*. One of the main contributions of this chapter is that we compare the performance of the most important representations that can be used for this purpose. We find that two of these are best suited for robust tagging. MarLiN (Martin et al., 1998) clusters – a derivative of Brown clusters – perform best for POS tagging. MarLiN clusters are also an order of magnitude more efficient to induce than the original Brown clusters. We provide an open source implementation of MarLiN clustering as part of this the research conducted for this dissertation. Our implementation is discussed in Appendix B.

Linguistic Morphological Analyzers (MAs) produce the best results in our experiments on morphological tagging. Our initial expectation was that differences between domains and lack of coverage would put resources manually compiled by linguists at a disadvantage in robust tagging when compared to learning algorithms that are run on very large text corpora. However, our results clearly show that representations produced by MAs are the best representations to use for robust morphological tagging.

The motivation for our work is that both morphological tagging and the "robust" application setting are important areas of research in NLP. To support this research, we created an extensive evaluation set for six languages. This involved identifying morphologically rich languages in which usable data sets with different distributional properties were available, designing mappings between different tagsets, organizing a manual annotation effort for one of the six languages and preparing large "general" (not domain-specific) data sets for unsupervised learning of word representations. The preparation and publication of this test suite is in itself a significant contribution.

The remainder of this chapter is structured as follows. Section 6.2 discusses related work. Section 6.3 presents the representations we tested. Section 6.4 describes data sets and the annotation and conversion efforts required to create the in-domain (ID) and out-of-domain (OOD) data sets. In Section 6.5, we describe the experiments and discuss our findings. In Section 6.6, we provide an analysis of our results. Section 6.7 summarizes our findings and contributions.

6.2 Related Work

Semi-supervised learning

Semi-supervised learning attempts to increase the accuracy of a machine learning system by using additional unlabeled data. Word representations, especially Brown clusters, have been extensively used for named entity recognition (NER) (Miller et al., 2004; Ratinov and Roth, 2009; Turian et al., 2010), parsing (Koo et al., 2008; Suzuki et al., 2009) and POS tagging (Collobert and Weston, 2008; Huang et al., 2009, 2014; Manning, 2011; Täckström et al., 2012; Owoputi et al., 2013; Schnabel and Schütze, 2014). In these papers, word representations were shown to yield consistent improvements and to often outperform traditional semi-supervised methods such as self-training. Semi-supervised learning by means of word representations has also been applied to French and Spanish morphological tagging (Chrupala, 2011).

Our work is similar to standard semi-supervised learning in training, but it also evaluates the methods on data sets that are from domains different from the domains of labeled and unlabeled training data. In contrast to earlier work on morphological tagging, we study a number of morphologically more complex and diverse languages. We also compare learned representations to representations obtained from MAs.

Domain Adaptation

Domain adaptation (DA) attempts to adapt a model trained on a source domain to a target domain. DA can be broadly divided into supervised and unsupervised approaches depending on whether labeled target domain data is available or not. Among unsupervised approaches to DA, representation learning (Ando and Zhang, 2005; Blitzer et al., 2006) uses the unlabeled target domain data to induce a structure that is suitable for transferring information from the labeled source domain to the target domain. Similar to representation learning for DA, we attempt to include word representations into the model that move feature weights from more domain specific features (word forms) to more general features (shared by multiple word forms) and evaluate the models by looking at their OOD performance. However, we induce the representation from a *general domain* in an attempt to obtain a model that has robust high accuracy across domains, including the source domain and target domains for which neither labeled nor unlabeled training data are available.

6.3 Representations

We investigate the distributional representations discussed in Chapter 2 with respect to their utility for POS tagging and MORPH tagging. (i) count vectors reduced by a Singular Value Decomposition (SVD), (ii) word clusters induced using the likelihood of a class-based language model, (iii) distributed embeddings trained using a neural network and (iv) accumulated tag counts, a task-specific representation obtained from an automatically tagged corpus.

6.4 Data Preparation

Our test suite consists of data sets for six different languages: Czech (cs), English (en), German (de), Hungarian (hu), Spanish (es) and Latin (la). Czech, German, Hungarian and Latin are morphologically rich. We chose these languages because they represent different families: Germanic (English, German), Romance (Latin, Spanish), Slavic (Czech) and Finno-Ugric (Hungarian) and different degrees of morphological complexity and syncretism. For example, English and Spanish rarely mark case while the other languages do; and as an agglutinative language, Hungarian features a low amount of possible readings for a word form while languages like German can have more than 40 different readings for a word form.

An additional criterion was to have a sufficient amount of labeled OOD data. The data sets also feature an interesting selection of domain differences. For example, for Latin we have texts from different epochs while the English data consist of canonical and non-canonical text.

6.4.1 Labeled Data

This section describes the annotation and conversion we performed to create consistent ID and OOD data sets. We first discuss Hungarian, English and Latin where no conversion was required as the data was already annotated in a consistent way.

Hungarian

For Hungarian we use the Szeged Dependency Treebank (Vincze et al., 2010), which consists of a number of different domains. We use the part that was used in the SPMRL 2013 shared task (Seddah et al., 2013) for training and as ID data (news-wire) and an excerpt from the novel *1984* and a *Windows 2000* manual as OOD data.

Latin

For Latin we use the PROIEL treebank (Haug and Jøhndal, 2008). It consists of data from the *Vulgate* (bible text from $\approx$ 380 AD), *Commentarii de Bello Gallico* (Caesar's notes on the Gallic War from $\approx$ 50 BC), Letters from Cicero to his friend Atticus ($\approx$ 50 BC) and an account of the Pilgrimage of Aetheria ($\approx$ 380 AD). We use the biggest text source *(Vulgate)* as ID data and the remainder as OOD data.

English

For English we use the SANCL shared task data (Petrov and McDonald, 2012), which consists of OntoNotes 4.0 as ID data and five OOD domains from the Google Web treebank: Yahoo! Answers, weblogs, news groups, business reviews and emails.

Czech

For Czech we use the part of the Prague Dependency Treebank (PDT) (Böhmová et al., 2003) that was used in the CoNLL 2009 shared tasks (Hajič et al., 2009) for training and as ID data. We use the Czech part of the Multext East (MTE) corpus (Erjavec, 2010) as OOD data. MTE consists of translations of the novel *1984* that have been annotated morphologically. PDT and MTE have been annotated using two different guidelines that without further annotation effort could only be merged by reducing them to a common subset. Specifically we removed features such as sub POS tags as well as markers for (in)animacy. The PDT features a number of tags that are ambiguous and could not always be resolved. The gender feature Q for example can mean feminine or neuter. If we could not disambiguate such a tag, we removed it; this results in morphological tags that are not present in the MTE corpus and a relatively high number of unseen tags. We give a more detailed discription of the conversion procedure in Appendix C. Our conversion code has been made available.[1]

Spanish

For Spanish we use the part of the AnCora corpus (Taulé et al., 2008) of CoNLL 2009 and the IULA treebank (Marimon et al., 2012), which consists of five domains: law, economics, medicine, computer science and environment. We use the AnCora corpus as ID data set and IULA as OOD data set. The two treebanks have been annotated using the same annotation scheme, but slightly different guidelines. Similar to Czech we merged the data sets by deleting features that could not be merged or were not present in one of the treebanks. In the AnCora corpus for example proper nouns are explicitly annotated with common gender and invariable number, while in IULA proper nouns do not have morphological features. As for Czech, we give a more detailed description of the conversion procedure in Appendix D. Our conversion code has been made available.[1]

German

For German we use the Tiger treebank (Brants et al., 2002) in the same split as Müller et al. (2013) as ID data and the Smultron corpus (Volk et al., 2010) as OOD data. Smultron consists of four parts: a description of Alpine hiking routes, a DVD manual, an excerpt of Sophie's World and economics texts. It has been annotated with POS and syntax, but not with morphological features. We annotated Smultron following the Tiger guidelines. The annotation process was similar to Marimon et al. (2012) in that the data sets where automatically tagged with the MORPH tagger MarMoT (Müller et al., 2013) and then manually corrected by two annotators. The baseline tagger was relatively strong as we could include features based on gold lemma, part-of-speech and syntax and the morphological analyzer SMOR (Schmid et al., 2004). The syntactic features were similar to Seeker and Kuhn (2013). The annotators were trained on the task for several weeks by annotating parts of the Tiger corpus and evaluating and discussing their annotations. The agreement of the annotators was 96.28%. For calculating the κ agreement, we

[1] http://cistern.cis.lmu.de/marmot/naacl2015/

assume that random agreement occurs when both annotators agree with the reading proposed by the tagger. We can now estimate the probability of random agreement by multiplying the individual estimated probabilities of changing the proposed tagging. This yields a random agreement probability of 89.65% and a κ of 0.64. As most of the differences between the annotators were cases where only one of the annotators had corrected an obvious error that the other had overlooked, the differences were resolved by the annotators themselves. The annotated data set haven been released to the public.[1]

Statistics

We used the provided segmentation if available and otherwise split ID data 8/1/1 into training, development and test sets and OOD data 1/1 into development and test sets if not mentioned otherwise. We thus have a classical setup of in-domain news paper text versus prose, medical, law, economic or technical texts for Czech, German, Spanish and Hungarian. For English we have canonical versus non-canonical data and for Latin data of different epochs (ca. 400 AD vs 50 BC). Additionally, for German one of the test domains is written in Swiss German. Table 6.2 summarizes basic statistics of the labeled data sets. Table 6.1 shows the fraction of out-of-vocabulary (OOV) tokens (word), unknown tags (tag) and of known tokens that occur with a tag they have not been seen with in the training set (word-tag).

		word		tag		word-tag	
		ID	OOD	ID	OOD	ID	OOD
morph	Czech	8.58	13.02	0.01	5.19	2.79	16.18
	English	2.72	9.50	0.00	0.40	0.61	2.47
	German	7.64	13.45	0.01	0.04	3.84	5.89
	Hungarian	19.94	26.98	0.09	0.29	0.42	2.40
	Latin	17.59	36.59	0.28	0.88	2.45	4.33
	Spanish	6.47	13.44	0.01	0.58	0.37	1.91

Table 6.1: Rates of unknown words, tags and word-tag combinations in ID and OOD development sets.

We see that Hungarian and Latin are the languages with the highest OOV rates. Hungarian has a productive agglutinative morphology while the high number of Latin OOVs can be explained by the small size of the training data. Czech features the highest unknown tag rate as well as the highest unseen word-tag rate. This can be explained by the limits of the conversion procedure we discussed above, e.g., ambiguous features such as Q.

	POS	MORPH	train	ID	OOD
Czech	12	450	652,544	87,988	27,350
English	48	48	731,678	32,092	53,156
German	54	681	719,530	76,704	24,622
Hungarian	22	673	170,141	29,989	83,087
Latin	23	749	59,992	9,475	41,432
Spanish	12	288	427,442	50,368	56,638

Table 6.2: Labeled data set statistics. Number of part of speech tags (POS) and morphological tags (MORPH); number of tokens in training set (train), ID development set and OOD development set.

6.4.2 Unlabeled Data

As unlabeled data we use Wikipedia dumps from 2014 for all languages except for Latin for which we use the Patrologia Latina, a collection of clerical texts from ca. 100 AD to 1200 AD from Corpus Corporum (Roelli, 2014). We do not use the Latin version of Wikipedia because it is written by enthusiasts, not by native speakers, and contains many errors. We preprocessed the Wikipedia dumps with Wikipedia Extractor (Attardi and Fuschetto, 2013) and performed sentence boundary detection using the NLTK (Bird et al., 2009) implementation of Punkt (Kiss and Strunk, 2006).

	Tool	Citation
Czech	CZECHTOK	Kveton (2013)
English	STANFORD TOKENIZER	Manning et al. (2014)
German		Schmid (2000)
Hungarian	MAGYARLANC	Zsibrita et al. (2013)
Spanish	FREELING	Padró and Stanilovsky (2012)

Table 6.3: Tokenizers used for the different languages. For Latin we used the in-house implementation discusses in the text.

Tokenization was performed using the tools discussed in Table 6.3 For Latin, we removed punctuation because the PROIEL treebank does not contain punctuation. We also split off the clitics “ne”, “que” and “ve” if the resulting token was accepted by LATMOR (Springmann et al., 2014). Following common practice, we normalized the text by replacing digits with 0s.

	articles	tokens	types
Czech	270,625	93,515,197	1,607,183
German	1,568,644	682,311,227	7,838,705
English	4,335,341	1,957,524,862	7,174,661
Spanish	1,004,776	432,596,475	6,033,105
Hungarian	245,558	95,305,736	2,776,681
Latin	5,316	88,636,268	713,162

Table 6.4: Number of articles, tokens and types in the unlabeled data sets.

Table 6.4 gives statistics for the unlabeled data sets.[2] Every language has at least 80 million tokens. The Latin vocabulary size is small because clerical texts cover fewer topics than an encyclopedia like Wikipedia.

	Tool	Citation
Czech	FREE morphology	Hajič (2001)
English	FREELING	Padró and Stanilovsky (2012)
German	SMOR	Schmid et al. (2004)
Hungarian	MAGYARLANC	Zsibrita et al. (2013)
Latin	LATMOR	Springmann et al. (2014)
Spanish	FREELING	Padró and Stanilovsky (2012)

Table 6.5: Morphological analyzers used for the different languages.

We also extract the morphological dictionaries using the morphological analyzers discussed in Table 6.5. We extract one feature for each cluster id or MA reading of the current word form. We also experimented with cluster indexes of neighboring uni/bigrams, but obtained no consistent improvement. For the dense embeddings we analogously extract the vector of the current word form.

	ID		OOD	
	ALL	OOV	ALL	OOV
Czech	4.7	42.8	6.5	45.6
English	0.9	23.8	2.1	22.7
German	7.7	55.0	8.4	50.6
Hungarian	9.9	37.6	11.3	38.0
Latin	2.0	8.0	6.8	17.6
Spanish	5.5	37.5	5.4	29.5

Table 6.6: Percentage of tokens not covered by the representation vocabulary.

In our experiments, we extract representations for the 250,000 most frequent word types. This vocabulary size is comparable to other work; e.g., Turian et al. (2010) use 269,000 word

[2]For Latin 105,997,019 tokens with punctuation included

types. Table 6.6 shows that these vocabularies of 250,000 have a low fraction of uncovered tokens for English and Latin. For the other languages, this fraction rises to $> 4\%$. The OOV numbers are most important as they tell us for how many of the probably hard-to-tag OOVs we will not be able to rely on the induced word representations.

		MarMoT (1)		MarMoT (2)		MarMoT (3)		Morfette		SVMTool	
		ID	OOD	ID	OOD	ID	OOD	ID	OOD	ID	OOD
morph	cs	93.27	77.83	**93.89**	78.52	93.86	**78.55**	91.48	76.56	91.06	75.41
	de	88.90	82.74	90.26	84.19	**90.54***	**84.30**	85.89	80.28	85.98	78.08
	es	98.21	93.24	**98.22**	93.62	98.16	93.42	97.95	**93.97***	97.96	91.36
	hu	**96.11**	89.78	96.07	**89.83**	95.92	89.70	95.47	89.18	94.72	88.44
	la	86.09	**67.90***	86.44	67.47	**86.47**	67.40	83.68	65.06	84.09	65.65

Table 6.7: Baseline experiments comparing MarMoT models of different orders with Morfette and SVMTool. Numbers denote average accuracies on ID and OOD development sets on the full morphological tagging task. A result significantly better than the other four ID (resp. OOD) results in its row is marked with $*$.

6.5 Experiments

For all our experiments we use MarMoT, the CRF-based tagger introduced in the last chapter. We already showed MarMoT to be a competitive POS and morphological tagger across six languages. In order to make sure that it is also robust in an OOD setup we compare it to the two popular taggers SVMTool (Giménez and Màrquez, 2004) and Morfette (Chrupala et al., 2008). As MarMoT uses stochastic gradient descent and thus produces different results for different training runs we always report the average of 5 independent runs. The OOD numbers are macro-averages over the OOD data sets of a language. The results are summarized in Table 6.7. Throughout this chapter we use the approximate randomization test (Yeh, 2000) to establish significance. To this end we compare the output of the medians of the five independent models. We regard p-values < 0.05 as significant. The tables in this chapter are based on the development sets; the only exception to this is Table 6.11, which is based on the test set. MarMoT outperforms SVMTool and Morfette on every language and setup (ID/ OOD) except for the Spanish OOD data set. For Czech, German and Latin the improvements over the best baseline are > 1. Different orders of MarMoT behave as expected: higher-order models (order > 1) outperform first-order models. The only exception to this is Latin. This suggests a drastic difference of the tag transition probabilities between the Latin ID and OOD data sets.

Given the results in Table 6.7 and for the sake of simplicity we use a 2nd order MarMoT model in all subsequent experiments.

		Brown_flat		Brown_path		MarLiN		mkcls	
		ID	OOD	ID	OOD	ID	OOD	ID	OOD
pos	cs	99.19	97.25	99.18	97.21	99.19	**97.26**	**99.21**	**97.26**
	de	98.08	93.42	98.07	93.47	98.10	93.44	**98.11**	**93.64***
	en	96.99	91.67	97.02	91.71	97.01	91.71	**97.03**	**91.86***
	es	98.84	97.91	98.84	**97.97**	**98.87**	**97.97**	98.84	97.90
	hu	97.95	93.40	97.89	93.39	97.98	93.36	**97.99**	**93.42**
	la	96.78	86.49	96.62	86.60	96.91	**87.24**	**96.95**	87.19
morph	cs	94.20	78.95	94.23	79.01	**94.35**	**79.14**	94.32	79.11
	de	90.71	85.39	90.75	85.44	**90.78**	**85.58**	90.68	85.47
	es	98.47	95.08	98.47	95.12	**98.48**	**95.15**	**98.48**	95.13
	hu	96.60	90.57	96.52	90.54	96.60	90.64	**96.61**	**90.66**
	la	87.53	71.69	87.44	71.60	**87.87**	**72.08**	87.67	71.88

Table 6.8: Tagging results for LM-based models.

6.5.1 Language Model-Based Clustering

We first compare different implementations of LM-based clustering. The implementation of Brown clustering by Liang (2005) is most commonly used in the literature. Its hierarchical binary structure can be used to extract clusterings of varying granularity by selecting different prefixes of the path from the root to a specific word form. Following other work (Ratinov and Roth, 2009; Turian et al., 2010), we induce 1000 clusters and select path lengths 4, 6, 10 and 20. We call this representation Brown_path.

We compare these Brown clusterings to mkcls (Och, 1999) and MarLiN. We also tested the implementation of Clark (2003), but it only supports ASCII and is considerably slower than the other implementations. mkcls implements a similar training algorithm as MarLiN, but uses simulated annealing instead of greedy maximization of the objective.

These implementations just induce flat clusterings of a certain size; we thus run them for cluster sizes 100, 200, 500 and 1000 to also obtain cluster ids of different sizes. The cluster sizes are chosen to roughly resemble the granularity obtained in Brown_path. We use these cluster sizes for all flat clusterings and call the corresponding models Brown_flat, mkcls and MarLiN.

The runtime of the Brown algorithm depends quadratically on the number of clusters while mkcls and MarLiN have linear complexity. This is reflected in the training times where for German the Brown algorithm takes $\approx 5000\,\mathrm{min}$, mkcls $\approx 2000\,\mathrm{min}$ and MarLiN $\approx 500\,\mathrm{min}$.

For these experiments as well as for other nominal features we just extract features from the current word form. We also experimented with the cluster indexes of neighboring words and bigrams, but could not obtain consistent improvements.

Table 6.8 shows that the absolute differences between systems are small, but overall MarLiN and mkcls are better (Brown_path reaches the same performance as MarLiN in one case: pos/es/OOD). We conclude that systems based on the algorithm of Martin et al. (1998) are slightly more accurate for tagging and are several times faster than the more frequently used

version of Brown et al. (1992a). We thus use MarLiN for the remainder of this chapter.

6.5.2 Neural Network Representations

We compare MarLiN with the implementation of CW by Al-Rfou et al. (2013). They extracted 64-dimensional representations for only the most frequent 100,000 word forms. To make the comparison fair, we use the intersection of our and their representation vocabularies. We also extract the representations for Latin from Wikipedia, not from Corpus Corporum as in the rest of the chapter.

We thus compare representations for $\approx$ 90,000 word forms all obtained from similar, but still slightly different Wikipedia dumps.

		Baseline		MarLiN		CW	
		ID	OOD	ID	OOD	ID	OOD
pos	cs	99.00	96.80	**99.16***	**97.06**	99.12	97.00
	de	97.87	92.21	**98.03**	**93.35***	**98.03**	93.02
	en	96.92	91.12	**97.05**	91.72	97.00	**91.86***
	es	98.62	96.70	98.79	**97.82***	**98.80**	97.31
	hu	97.49	92.79	**97.94**	93.30	97.88	**93.40**
	la	95.80	81.92	**96.35***	**85.52***	95.88	84.50
morph	cs	93.89	78.52	**94.23***	**78.91**	94.10	78.80
	de	90.26	84.19	90.54	85.08	**90.59**	**85.21**
	es	98.22	93.62	**98.44**	**94.97***	**98.44**	94.32
	hu	96.07	89.83	96.47	90.60	**96.48**	**90.95***
	la	86.44	67.47	**86.95**	**70.30***	86.76	69.32

Table 6.9: Tagging results for the baseline, MarLiN and CW.

The results in Table 6.9 show that the MarLiN result is best in 15 out of 22 cases and significantly better in ten cases. CW is best in nine out of 22 cases and significantly better in four cases. We conclude that LM-based representations are more suited for tagging as they can be induced faster, are smaller and give better results.

6.5.3 SVD and ATC Representations

For the SVD-based representation we use feature ranks out of $\{500, 1000\}$ and dimensions out of $\{50, 100, 200, 500\}$. We found that l1-normalizing the vectors before and after the SVD improved results slightly. The dense vectors are used directly as real valued features. For the accumulated tag counts (ATC) we annotate the data with our baseline model and extract word-tag probabilities. The probabilities are then used as sparse real-valued features.

		Baseline		ATC		MarLiN		MA		SVD	
		ID	OOD	ID	OOD	ID	OOD	ID	OOD	ID	OOD
pos	cs	99.00	96.80	99.11	97.03	**99.19**	**97.26**	99.18	97.25	99.11	97.09
	de	97.87	92.21	98.00	92.92	**98.10**	**93.44***	98.00	92.87	98.09	92.88
	en	96.92	91.12	96.97	91.47	**97.01**	91.71	96.99	91.57	97.00	**91.75**
	es	98.62	96.70	98.79	97.09	**98.87**	**97.97**	**98.87**	97.89	98.80	97.16
	hu	97.49	92.79	97.84	93.15	97.98	93.36	**98.12***	**93.77***	97.86	93.30
	la	95.80	81.92	96.17	83.40	**96.91**	**87.24***	96.81	86.31	96.36	85.01
morph	cs	93.89	78.52	94.16	78.75	94.35	79.14	**94.48***	**79.41***	94.14	78.94
	de	90.26	84.19	90.56	84.78	**90.78**	85.58	90.75	**85.75**	90.69	85.15
	es	98.22	93.62	98.38	93.92	98.48	95.15	**98.56***	**95.43***	98.40	94.18
	hu	96.07	89.83	96.25	90.07	96.60	90.64	**96.83***	**91.14***	96.46	90.50
	la	86.44	67.47	86.96	68.61	87.87	72.08	**88.40***	**73.23***	87.45	70.81

Table 6.10: Tagging results for the baseline and four different representations.

Table 6.10 shows that all representations outperform the baseline. Improvements are biggest for Latin. Overall SVD outperforms ATC and is outperformed by MarLiN and MA. MarLiN gives the best representations for POS tagging while MA outperforms MarLiN in MORPH tagging. Table 6.11 shows that the findings for the baseline, MarLiN and MA also hold for the test set.

		Baseline		MarLiN		MA	
		ID	OOD	ID	OOD	ID	OOD
pos	cs	98.88	96.43	**99.11***	96.94	99.06	**96.95**
	de	97.32	91.10	**97.73***	**92.00***	97.60	91.49
	en	97.36	89.81	**97.58***	**90.65***	97.47	90.51
	es	98.66	97.94	**98.94***	98.33	98.87	**98.38**
	hu	96.84	92.11	97.08	92.95	**97.46***	**93.25***
	la	93.02	81.35	**95.20**	**87.58***	95.11	86.45
morph	cs	93.93	77.50	94.33	78.12	**94.50***	**78.37***
	de	88.41	82.78	89.18	83.91	**89.32***	**84.09**
	es	98.30	95.65	98.53	95.92	**98.54**	**96.33***
	hu	94.82	88.82	95.46	89.98	**95.85***	**90.46***
	la	82.09	65.59	84.67	71.25	**85.91***	**72.42***

Table 6.11: Tagging results for the baseline, MarLiN and MA on the test set.

6.6 Analysis

We now analyze why MarLiN and MA have better performance than the baseline. First we compare the improvements in absolute error rate over the baseline by grouping word forms by their training set frequency f. Table 6.12 shows that most of the improvement comes from OOV words.

			$f = 0$	$0 < f < 10$	$f >= 10$
morph	cs	MarLiN	0.29	0.22	0.11
		MA	0.37	0.35	0.16
	de	MarLiN	1.02	0.17	0.19
		MA	0.85	0.29	0.42
	es	MarLiN	1.36	0.15	0.02
		MA	1.50	0.27	0.04
	hu	MarLiN	0.62	0.18	0.00
		MA	1.07	0.20	0.03
	la	MarLiN	3.76	0.80	0.06
		MA	4.98	0.69	0.09

Table 6.12: Improvement compared to the baseline for different frequency ranges of words on OOD.

Rare words show a smaller, but still important contribution while the contribution of frequent words can be almost neglected for four languages. An interesting exception is German where frequent words contribute more to the error reduction than rare words. This could be caused by syncretisms such as in plural noun phrases where the gender is not marked in determiner and adjective and can only be derived from the head noun; e.g., the adjectives in "schwere Schulfächer" 'difficult school subjects' and "verdächtige Personen" 'suspect persons' are unmarked for gender and the correct genders (neuter vs. feminine) cannot be inferred from distributional information or suffixes for the nouns (although gender is easy to infer distributionally for the singular forms of the nouns).

In Table 6.13, we list the morphological features with the highest improvements in absolute error rate.

morph	cs	MarLiN	gen	0.70	cas 0.41	pos 0.35
		MA	gen	0.85	cas 0.51	pos 0.31
	de	MarLiN	gen	1.23	pos 1.14	num 0.62
		MA	gen	1.37	pos 0.63	num 0.59
	es	MarLiN	sub	1.49	gen 1.21	pos 1.07
		MA	sub	1.34	gen 1.24	pos 1.10
	hu	MarLiN	cas	0.71	sub 0.66	pos 0.52
		MA	cas	0.88	sub 0.84	pos 0.76
	la	MarLiN	pos	5.19	cas 3.46	gen 3.25
		MA	pos	4.68	gen 3.85	cas 3.01

Table 6.13: Features with highest absolute improvement in error rate: Gender (gen), Case (case), POS (pos), Sub-POS (sub) and Number (num).

The most important features are pos (POS), sub (a finer division of POS, e.g., nouns are split into proper nouns and common nouns), gen (gender), cas (case) and num (number). For all languages pos and – if part of the annotation – sub are among the three features with the highest improvements. Gender is also always among the three features with the highest improvements for the four languages that have gender (es, de, la, cs). We just discussed an example for German where gender could not be derived from context or inflectional suffixes. Other languages also have word forms that do not mark gender, e.g., in Spanish masculine "ave" 'bird' versus feminine "llave" 'key' and in Latin masculine "mons" 'mountain' versus feminine "pars" 'part.' The gender can, however, easily be derived if the word representation encodes whether a word form has been seen with a specific determiner or adjective on its right or left.

Lastly, we use Jaccard similarity to compare the sets of gold and predicted morphological features.

$$\text{Jaccard}(U, V) = |U \cap V|/|U \cup V| \tag{6.1}$$

Jaccard can be interpreted as a soft variant of accuracy: If the two tags are identical it yields 1 and otherwise it corresponds to the number of correctly predicted features divided by the size of the union of gold and predicted features.

		cs	de	es	hu	la
morph	accuracy	79.41	85.72	95.43	91.14	73.23
	Jaccard	89.89	90.71	96.77	93.52	83.68

Table 6.14: Comparison between a Jaccard-based and accuracy-based evaluation.

Table Table 6.14 demonstrates that the evaluation measure we have used throughout this chapter – a tag counts as completely wrong if a single feature was misidentified even though all others are correct – is conservative. On a feature-by-feature basis accuracy would be much higher. The difference is largest for Czech and Latin.

6.7 Conclusion

We have presented a test suite for morphological tagging consisting of in-domain (ID) and out-of-domain (OOD) data sets for six languages: Czech, English, German, Hungarian, Latin and Spanish. We converted some of the data sets to obtain a reasonably consistent annotation and manually annotated the German part of the Smultron treebank.

We surveyed four different word representations: SVD-reduced count vectors, language model-based clusters, accumulated tag counts and distributed word embeddings based on Collobert and Weston (2008). We found that the LM-based clusters outperformed the other representations across POS and MORPH tagging, ID and OOD data sets and all languages. We also showed that our implementation MarLiN of Martin et al. (1998) is an order-of-magnitude more efficient and performs slightly better than the implementation by Liang (2005).

We also compared the learned representations to manually created Morphological Analyzers (MAs). We found that MarLiN outperforms MAs in POS tagging, but that it is substantially worse in morphological tagging. In our analysis of the results, we showed that both MarLiN and MAs decrease the error most for out-of-vocabulary words and for the features POS and gender.

Chapter 7

Conclusion

In this thesis we have presented a number of approaches to improve the handling of morphologically rich languages (MRLs).

In Chapter 3 we have investigated a novel **morphological language model**, an interpolation of a Kneser-Ney (KN) model with a class-based language model whose classes are defined by morphology and shape features. The model achieves consistent reductions in perplexity for all languages represented in the Europarl corpus, ranging from 3% to 11%, when compared to a KN model. We found perplexity reductions across all 21 languages for histories ending with four different types of word shapes: alphabetical words, special characters, and numbers. The model' 's hyperparamemters are θ, a threshold that determines for which frequencies words are given their own class and ϕ, the number of suffixes used to determine class membership; and morphological segmentation. Looking at their sensitivity we found that θ has a considerable influence on the performance of the model and that optimal values vary from language to language. This parameter should be tuned when the model is used in practice. In contrast, the number of suffixes and the morphological segmentation method only had a small effect on perplexity reductions. This is a surprising result since it means that simple identification of suffixes by frequency and choosing a fixed number of suffixes ϕ across languages is sufficient for getting most of the perplexity reduction that is possible.

We think that the best way to further improve the accuracy of our model would be to replace the simple linear interpolation by a more sophisticated approach. One possibility would be to interpolate the morphological model with the lower-order KN models. This would allow the model to put more emphasis on the KN model for frequent n-grams and more emphasis on the morphological model for rare or unseen n-grams.

In Chapter 4 we investigate the utility of **Hidden Markov models with latent annotations (HMM-LAs)** for dependency parsing. We have shown that HMM-LAs are not only a method to increase the performance of generative taggers, but also that the generated latent annotations are linguistically interpretable and can be used to improve dependency parsing. Our best systems improve an English parser from a LAS of 90.34 to 90.57 and a German parser without morphological features from 87.92 to 88.24 and with morphological features from 88.35 to 88.51. Our analysis of the parsing results shows that the major reasons for the improvements are: the separation of POS tags into more and less trustworthy subtags, the creation of POS subtags with higher

correlation to certain dependency labels and for German a correlation of tags and morphological features such as case.

While the procedure works in general, there are a couple of things that could be improved. One problem is that not every split made by the HMM-LA is actually useful for the parser. We pointed out lexicalization as a type of split that increases HMM accuracy, but does not help an already lexicalized parser. The question is whether one can identify such useless splits automatically and thereby create higher-quality POS tagsets. One way might be to use dependency tree information in the merge phase. It would also be interesting to use the hierarchy induced by the split-merge training to provide tags of different granularity. In a preliminary experiment we found that this turns out to be difficult as the hierarchy does not stay consistent over training. There is no guarantee that a tag NN_{00} is more similar to NN_{01} (both subtags of NN_0) than to, for example, NN_{10}. We think that smoothing that couples parents and children in the tag hierarchy (like the WB smoothing we proposed) might be one way to force the training into a consistent hierarchy. The challenge is to find a way to keep the hierarchy consistent without making the tags less specific or at least to find the right balance.

In Chapter 5 we presented a **fast and accurate** approach to **morphological tagging**. Our pruned CRF model is based on coarse-to-fine decoding and stochastic gradient descent training with early updating. We have shown that for moderate tagset sizes of ≈ 50, our implementation MarMoT gives significant speed-ups over a standard CRF with negligible losses in accuracy. Furthermore, we have shown that training and tagging for approximated trigram and fourgram models is still faster than standard 1^{st}-order tagging, but yields significant improvements in accuracy. In oracle experiments with morphological tagsets we demonstrated that the losses due to our approximation depend on the word level ambiguity of the respective language and are moderate (≤ 0.14) except for German where we observed a loss of 0.37. We also showed that higher order tagging – which is prohibitive for standard CRF implementations – yields significant improvements over unpruned 1^{st}-order models. Analogous to the oracle experiments we observed big improvements for languages with a high level of POS+MORPH ambiguity such as German and smaller improvements for languages with less ambiguity such as Hungarian and Spanish.

In parsing experiments on the SPMRL-ST 2013 data sets we showed that the model can be used to improve the results of a state-of-the art parser for all languages except French.

Possible future work would include to extend the model to even larger tagsets, for example by adding syntactic chunks as a third tagging level. Joint syntactic chunking and morphological tagging could lead to improved accuracy. However, in preliminary experiments we found our current training strategy with early updating to be insufficient for training models based on these complex lattices. Another line of research could try to improve the tagging accuracy by integrating the selectional preferences of certain verbs. Sentences such as *die Maus jagt die Katze* 'the cat chases the mouse' are ambiguous in German as the nominative and accusative case cannot be read off the form or the feminine article *die*. In many cases this could be resolved by knowing that cats are much more typical subjects of chase than mice.

In Chapter 6 we have presented a test suite for **robust morphological tagging** consisting of in-domain (ID) and out-of-domain (OOD) data sets for six languages: Czech, English, German, Hungarian, Latin and Spanish. We converted some of the data sets to obtain a reasonably consistent annotation and manually annotated the German part of the Smultron treebank.

We surveyed four different word representations: SVD-reduced count vectors, language model-based clusters, accumulated tag counts and distributed word embeddings based on Collobert and Weston (2008). We found that the LM-based clusters outperformed the other representations across POS and MORPH tagging, ID and OOD data sets and all languages. We also showed that our implementation MarLiN of Martin et al. (1998) is an order-of-magnitude more efficient and performs slightly better than the implementation by Liang (2005).

We also compared the learned representations to manually created Morphological Analyzers (MAs). We found that MarLiN outperforms MAs in POS tagging, but that it is substantially worse in morphological tagging. In our analysis of the results, we showed that both MarLiN and MAs decrease the error most for out-of-vocabulary words and for the features POS and gender.

In future work, one should try to combine the morphological resources and unlabeled data sets to obtain better word representations. This could for example be done by weighting the output of the finite-state morphologies. This could be helpful as many word forms have readings that are technically correct but very unlikely.

Appendix A

MarMoT Implementation and Usage

In this appendix we explain the important implementation details of our CRF tagger MarMoT. All the shown listings are simplified versions of the MarMoT source code. The latest version of the MarMoT and its documentation can be found at http://cistern.cis.lmu.de/marmot.

A.1 Feature Extraction

Many papers on structure prediction discuss how decoding can be improved, because it usually has a super-linear time complexity. However, in practice feature extraction seems to be the most time consuming part of decoding. In MarMoT we use an implementation of feature extraction that is similar to the implementation in the mate-tools parser by Bohnet (2010).[1] The implementation aims to reduce the cost of creating concatenated features by first mapping all atomic features to indexes, this is done by a simple hash-based symbol table. To illustrate the implementation we focus on the extraction of word form and suffix features. In MarMoT, words are represented by a class `Word`, that stores the string of the word form and the corresponding form and character indexes:

```
1  public class Word implements Token {
2    private String word_form_;
3    private int word_index_;
4    private short[] char_indexes_;
5
6    public String getWordForm();
7    public void setWordIndex(int word_index);
8    public int getWordFormIndex();
9    public void setCharIndexes(short[] char_indexes);
10   public short[] getCharIndexes();
11 }
```

[1] https://code.google.com/p/mate-tools/

The field `word_form` is set during the construction of a new Word object, while the index fields get set during the first step of the feature extraction. This first step is handled by the method `addIndexes` which uses two symbol tables to store the string and character values. We do not give a listing for the `SymbolTable` as it only has one important method `toIndex` that maps a symbol to it corresponding index. If the symbol is not present in the table it is either added to the table (if the parameter `insert` is true) or a default index (usually -1) is returned. The parameter `insert` is true during training and false during testing and tagging.

```
1  private SymbolTable<String> word_table_;
2
3  public void addIndexes(Word word, boolean insert) {
4    String word_form = word.getWordForm();
5    int word_index = word_table_.toIndex(word_form, -1, insert);
6    word.setWordIndex(word_index);
7    addCharIndexes(word, word_form, insert);
8  }
```

We see that `addIndexes` simply sets the index of the word form and calls a method that similarly creates an array with the index of every character. During the second part of the feature extraction the actual features are encoded into variable integer arrays of optimal size. In order to uniquely encode a features we need to encode the unique identifier of the feature template (e.g., 0 for the word form feature template), the order of the feature (state features such as the word form receive 0, first-order transition features 1 and so on) and the value of the feature. We assume that all these values are numbers from 0 to a known maximum and can thus calculate the number of bits needed to encode them. The method `append` of the `Encoder` class then shifts the necessary number of bits into the integer array. The integer array is then used to efficiently calculate a unique index for the feature value (method `getFeatureIndex`).

```
1  public FeatureVector extractStateFeatures(Sequence sequence,
2                                            int token_index) {
3    Word word = (Word) sequence.get(token_index);
4    int form_index = token.getWordFormIndex();
5    MorphFeatureVector features = new MorphFeatureVector();
6    int fc = 0;
7    encoder_.append(0, order_bits_);
8    encoder_.append(fc, state_feature_bits_);
9    encoder_.append(form_index, word_bits_);
10   Feature feature = encoder_.getFeature()
```

```
    features.add(getFeatureIndex(feature));
    encoder_.reset();
```

The advantage of the extraction method becomes clear during the extraction of suffix (or prefix) features where we otherwise would need to hash dozens of substrings. Note, that in the following listing we do not call `reset()` after each index calculation and thus simply keep adding feature values to the variable integer array:

```
encoder_.append(0, order_bits_);
encoder_.append(fc, state_feature_bits_);
for (int pos = 0; pos < chars.length; pos++) {
  short c = chars[chars.length - pos - 1];
  encoder_.append(c, char_bits_);
  Feature feature = encoder_.getFeature()
  features.add(getFeatureIndex());
}
encoder_.reset();
```

During decoding we need to concatenate feature and tag indexes in order to calculate the positions in the 1-dimensional weight vector. This is done by first mapping the two indexes to a unique 1-dimensional index and then hashing it in order to fit the capacity of the weights array:

```
  double[] weights_;
  int total_num_tags;

  private int getIndex(int feature, int tag_index) {
    int index = feature * total_num_tags_ + tag_index;
    int capacity = weights_.length;
    int h = hash(index, capacity);
    return h;
  }
```

A.2 Lattice Generation

In this section we describe the implementation of our pruned lattice generation algorithm. The core of the algorithm is a method called `getSumLattice`. A sum lattice is a data structure that allows for efficient calculation of marginal sums by using a form of the Forward-Backward

algorithm or in the case of a zero-order lattice by simply summing over the possible tag sums at every position. In our implementation we increase the order and the level of the lattice. The level is the complexity of the processed tags: In the morphological tagging case we just have two levels, the part-of-speech and the morphological level. We now give a simplified version of `getSumLattice`:

```
1  public SumLattice getSumLattice(Sequence sequence,
2                                  int max_order,
3                                  int max_level,
4                                  double threshold) {
5    List<List<State>> candidates = getStates(sequence);
6    SumLattice lattice = null;
7
8    for (int l = 0; l < max_level; l++) {
9      if (l > 0) {
10       candidates = lattice.getZeroOrderCandidates();
11       candidates = extendStates(candidates, sequence);
12     }
13     lattice = new ZeroOrderSumLattice(candidates, threshold);
14     for (int o = 0; o < order; o++) {
15       candidates = lattice.prune();
16       if (o > 0) {
17         candidates = increaseOrder(candidates, l);
18       }
19       addTransitions(candidates, l, o);
20       lattice = new SequenceSumLattice(candidates,
21                                        threshold,
22                                        o);
23     }
24   }
25   return lattice;
26 }
```

where we first use `getStates` to generate a complete set of tag candidates for every word in the sentence. If one would like to use morphological analyzers as a hard constraint to filter the readings of a token, this could be done by overloading this method. Then we iterate over the different levels. At every level but the first level we call `extendStates`, which increases the level of the candidates by adding all the possible combinations of the next level. In the case of MarMoT, if the candidates for a certain word are verb and noun then `extendStates` will create a list of all the possible morphological verb and noun readings. Then we create a simple zero-order lattice and iterate over the model order. At each order we prune the candidate space. For order > 1 we call `increaseOrder`, which merges the neighboring states in a lattice in

order to create a higher order lattice, this is not necessary when going from a zero-order to a first-order lattice, as both lattices work on simple tag states. After that we add the transitions between the new states in the lattice. This is done by a call to `addTransitions`, which adds all the possible transitions between neighboring states. For higher order models, it only creates transitions between consistently overlapping states. Lastly, we create a new sequence sum lattice. A crucial part of the implementation is the modeling of the lattice states: A first-order state is represented in the following way:

```
1 public class State {
2         private int index_;
3         private State sub_level_state_;
4         private FeatureVector vector_;
5         private double score_;
6         protected double estimated_count_;
7         private Transition[] transitions_;
```

where `index_` represents the tag index of the state (e.g. 0 for noun and 1 for verb). If we are dealing with a higher level state then `sub_level_state_` points to the representation of the next lower level state. Just as we discussed in Section A.1, `FeatureVector` stores the feature indexes of the state. This vector is shared between all the states representing the same word. Score denotes the potential of the state that is the dot product between the feature vector and the weight vector and `estimated_count_` stores the marginal sum calculated by the sum lattice. `transitions_` stores the possible transitions to the next state.

Transition and higher order states are both represented by the `Transition` class. This has the advantage that the transition can just be reused when increasing the lattice order.

```
1 public class Transition extends State {
2         private State previous_state_;
3         private State state_;
```

transitions have the same properties as states, but also point to the previous and next state. Tag n-gram feature indexes are stored in the `vector_` field while `score_` stores the sum of the sub states and the dot product of transition features and weight vector. This representation makes the calculation of the state scores rather complex:

```
1   public double dotProduct(State state, FeatureVector vector) {
2     State zero_order_state = state.getZeroOrderState();
```

```
3     int tag_index = getUniversalIndex(zero_order_state);
4     double score = 0.0;
5     for (int feature_index : vector) {
6       score += getWeight(getIndex(feature_index, tag_index));
7     }
8     return score;
9   }
```

In the dot product computation, we first need to calculate a universal index. This index is a 1-dimensional representation of the tag indexes at the different levels. If the state is a first level state this universal index corresponds to the tag index of the state. In case of a complex state we calculate the index from the state the transition is pointing to. For example, a state `(noun, verb)` corresponds to a transition from a noun to a verb state. As the tag indexes of the noun state are already represented in the feature vector, we need to calculate the combined index of feature and tags from the verb node. `getZeroOrderState()` thus returns the target of a complex transition state and the state itself for a simple state.

A.3 Java API

Loading a pretrained tagger and tagging a list of word forms using the MarMoT API looks like this:

```
1   List<List<String>> tag(List<String> forms,
2                          String taggerfile) {
3     Tagger tagger = FileUtils.loadFromFile(taggerfile);
4     List<Word> words = new LinkedList<Word>();
5     for (String form : forms) {
6       words.add(new Word(form));
7     }
8     Sentence sentence = new Sentence(words);
9     return tagger.tag(sentence);
10  }
```

Here the tagger is loaded using the utility function `loadFromFile`, but it could also be loaded using the standard Java serialization mechanism. This will give you an instance of the `Tagger` interface:

```
1 public interface Tagger extends Serializable {
2   List<List<String>> tag(Sequence sentence);
```

```
3   Model getModel();
4 }
```

This interface has two important methods. `tag` will tag a `Sequence` and return the tags as a list of lists. Each list corresponds to a token in the sequence and will contain one string per model level. That is, in the case of the morphological tagger it well contain the POS and MORPH tag for every token. The other method `getModel` will give you the underlying `Model` object. The model can be used to access the options and for example change the verbosity of the tagger:

```
1   Options options = tagger.getModel().getOptions();
2   options.setProperty(Options.VERBOSE, "true");
```

`Sequence` is also an interface, and as you can see we obtain one by creating a `Sentence` object:

```
1 public class Sentence extends AbstractList<Token>
2                       implements Sequence {
3
4   public Sentence(List<Word> tokens);
```

Sentence objects are created from lists of words which are the concrete implementation of `Token` interface above. We already discussed the `Word` class in Section A.1.

The MarMoT API can also be used to train a new model. If our training data is in a format where every token is a list containing form, POS and MORPH tag and every sentence is a list of such tokens then a complete example could look like this:

```
1   public void train(List<List<List<String>>> data,
2                     String tagger_file) {
3     // Create the training data.
4     List<Sequence> train_sequences = new LinkedList<>();
5     for (List<List<String>> sentence_data : data) {
6       List<Word> words = new LinkedList<Word>();
7       for (List<String> token_data : sentence_data) {
8   String form = token_data.get(0);
9   String pos = token_data.get(1);
10  String morph = token_data.get(2);
11  words.add(new Word(form, pos, morph));
```

```
Murmeltiere
sind
im
Hochgebirge
zu
Hause
.

Im
Hochgebirge
sind
Murmeltiere
zu
Hause
.
```

Figure A.1: Example of raw text input for the MarMoT command line utility.

```
12        }
13        train_sequences.add(new Sentence(words));
14      }
15
16      // Train the model.
17      MorphOptions options = new MorphOptions();
18      Tagger tagger = MorphModel.train(options, train_sequences);
19      FileUtils.saveToFile(tagger, tagger_file);
20    }
```

A.4 Command Line Usage

MarMoT can also be used from the command line. The input to MarMoT is a file in a one-token-per-line format. Sentence boundaries should be marked by a new line. A valid example can be found in Figure A.1

If the sentences in Figure A.1 were stored in a file called `text.txt` a pretrained MarMoT could be run by executing:

```
java -cp marmot.jar marmot.morph.cmd.Annotator\
     --model-file de.marmot\
     --test-file form-index=0,text.txt\
     --pred-file text.out.txt
```

where `form-index=0` specifies that the word form can be found in the first column. The MarMoT Annotator produces output in a truncated CoNLL 2009 format (the first 8 columns). The output for the first sentence is shown in the following Figure A.2.
A new model can be trained from this output file by running:

```
0     Murmeltiere   _     _     _     \NN      _     case=nom|number=pl|gender=masc
1     sind          _     _     _     VAFIN    _     number=pl|person=3|tense=pres|mood=ind
2     im            _     _     _     APPRART  _     case=dat|number=sg|gender=neut
3     Hochgebirge   _     _     _     \NN      _     case=dat|number=sg|gender=neut
4     zu            _     _     _     APPR     _     _
5     Hause         _     _     _     \NN      _     case=dat|number=sg|gender=neut
6     .             _     _     _     $.       _     _
```

Figure A.2: Example output of the MarMoT Annotator.

```
java -Xmx5G -cp marmot.jar marmot.morph.cmd.Trainer\
     -train-file form-index=1,tag-index=5,morph-index=7,text.out.txt\
     -tag-morph true\
     -model-file en.marmot
```

where `tag-index=5` specifies that the POS can be found in the sixth column and `morph-index=7` that the morphological features can be found in the eighth column. An important training parameter is the model order. The default is a second order model, but for some languages such as German a higher order might give better results. For completeness we give a list of all the available training options:

Parameter	Description	Default Value
prune	Whether to use pruning.	true
effective-order	Maximal order to reach before increasing the level.	1
seed	Random seed to use for shuffling. 0 for nondeterministic seed	42
prob-threshold	Initial pruning threshold. Changing this value should have almost no effect.	0.01
very-verbose	Whether to print a lot of status messages.	false
oracle	Whether to do oracle pruning. Probably not relevant.	false
trainer	Which trainer to use.	marmot.core.CrfTrainer
num-iterations	Number of training iterations.	10
candidates-per-state	Average number of states to obtain after pruning at each order. These are the μ values.	[4, 2, 1.5]
max-transition-feature-level	Something for testing the code.	-1
beam-size	Specify the beam size of the n-best decoder.	1
order	Set the model order.	2
initial-vector-size	Size of the weight vector.	10000000
averaging	Whether to use averaging. Perceptron only!	true
shuffle	Whether to shuffle between training iterations.	true
verbose	Whether to print status messages.	false
quadratic-penalty	L2 penalty parameter.	0.0
penalty	L1 penalty parameter.	0.0

Table A.1: General MarMoT options

Parameter	Description	Default Value
observed-feature	Whether to use the observed feature.	true
split-pos	Whether to split POS tags. See subtag-separator	false
form-normalization	Whether to normalize word forms before tagging	none
shape	Whether to use shape features	false
special-signature	Whether to mark if a word contains a special character in the word signature.	false
num-chunks	Number of chunks. CrossAnnotator only.	5
restrict-transitions	Whether to only allow POS $\rightarrow$ MORPH transitions that have been seen during training.	true
type-dict	Word type dictionary file (optional)	
split-morphs	Whether to split MORPH tags. See subtag-separator.	true
rare-word-max-freq	Maximal frequency of a rare word.	10
type-embeddings	Word type embeddings file (optional).	
tag-morph	Whether to train a morphological tagger or a POS tagger.	true
subtag-separator	Regular expression to use for splitting tags. (Has to work with Java's String.split)	\\\|
internal-analyzer	Use an internal morphological analyzer. Currently supported: 'ar' for AraMorph (Arabic)	none
model-file	Model file path.	none
train-file	Input training file	none
test-file	Input test file. (optional for training)	none
pred-file	Output prediction file in CoNLL09. (optional for training)	none
shape-trie-file	Path to the shape trie. Will be created if non-existent.	none

Table A.2: Morphological MarMoT options

Appendix B

MarLiN Implementation and Usage

In this appendix we explain the important implementation details of MarLiN (Martin et al., 1998). The latest version of the MarLiN source code and its documentation can be found at http://cistern.cis.lmu.de/marlin/.

B.1 Implementation

Our implementation follows the ideas explained in Martin et al. (1998). The most important part is the assignment of a word form to a specific class. This can be implemented efficiently, if we keep track of the left and right contexts of each word. The following C++ code shows how this is implemented in MarLiN:

```
1  void incrementBigrams(int word, int klass, int factor) {
2    forvec (_, Entry, entry, left_context_[word]) {
3      int cword = entry.item;
4      if (cword != word) {
5        int cclass = word_assignment_[cword];
6        addTagTagCount(cclass, klass, factor * entry.count);
7      } else {
8        addTagTagCount(klass, klass, factor * entry.count);
9      }
10   }
11   forvec (_, Entry, entry, right_context_[word]) {
12     int cword = entry.item;
13     if (cword != word) {
14       int cclass = word_assignment_[cword];
15       addTagTagCount(klass, cclass, factor * entry.count);
16     }
17   }
18 }
```

`left_context` and `right_context` map each form to the list of its left and right neighbors, respectively. `addTagTagCount` increments the transition count of class `klass` preceeding class `cclass`.

We also found that a huge speed up could be obtained if $n \cdot \log n$ was precomputed for all $n < 10.000$ and cached in an array:

```
size_t cache_size_ = 10000;
vector<double> nlogn_cache_;

void init_cache() {
  nlogn_cache_.resize(cache_size_);
  for (int i=0; i<cache_size_; i++) {
    nlogn_cache_[i] = (i + 1) * log(i + 1);
  }
}

double nlogn(int n) {
  assert (n >= 0);
  if (n == 0) {
    return 0;
  }
  if (n - 1 < cache_size_) {
    return nlogn_cache_[n - 1];
  }
  return n * log(n);
}
```

B.2 Usage

Our implementation consists of two programs: a PYTHON program called `marlin_count` and a C++ program called `marlin_cluster`.

`marlin_count` counts the unigrams and bigrams in a tokenized text corpus in a one sentence per line format and transforms these counts into an efficient index representation. The following command reads the tokenized text in `example.txt` and stores the extracted unigram and bigram statistics in the respective files.

```
marlin_count --text example.txt\
             --bigrams bigrams\
             --words unigrams
```

`marlin_cluster` then reads this representation and executes the actual cluster algorithm. The following command reads the unigram and bigram statistics in `unigrams` and `bigrams`

and clusters the respective word types into 10 clusters. The resulting clustering is then written to the file `clusters`.

```
marlin_cluster --words unigrams\
               --bigrams bigrams\
               --c 10\
               --output clusters
```

Appendix C

MULTEXT-East-PDT Tagset Conversion

For the Czech data sets we mapped the Prague Dependency Treebank (PDT) (Böhmová et al., 2003) annotation to the much older MULTEXT-East (MTE) (Erjavec, 2010) annotation. For PDT we use the data that was part of the CoNLL 2009 shared task data sets (Hajič et al., 2009). Our conversion is based on the documentation of the PDT and MTE tagset.[1,2] The first issue are SubPos both PDT and MTE have a SubPOS structure, but they are not compatible. While PDT for example differentiates verbs by tense and mood, MTE has categories such as full, auxiliary and model. Therefore we remove both SubPOS annotations. The three features that cannot be easily mapped are animacy, number and gender. The reason is that PDT merges the three features into a single tag. This gives rise to a couple of ambiguous cases. We therefore decided to remove the animacy annotation and wrote some code to disambiguate the number and gender annotation. The problematic PDT tags are *q*, *t*, *z* and *h*. *q* can denote feminine or neuter, but feminine only with singular and neuter only with plural. *t*, similarly can denote masculine or feminine, but feminine only with plural. We thus try to set the number feature first and set the gender feature accordingly. Many tokens however only specify number through the same tags which means that both are ambiguous. *z* and *h* cannot be disambiguated. *h* can be feminine or neuter (without any restrictions on number), and *z* can be masculine or neuter. We thus just ignore the two features. For number there is only one problematic tag *w*, which can denote singular (feminine only) or plural (neuter only). This forces us to run our code for gender and number specification in a loop. Our conversion code is available online.[3]

[1] https://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Doc/hmptagqr.html
[2] http://nl.ijs.si/ME/V4/msd/html/msd-cs.html
[3] http://cistern.cis.lmu.de/marmot/naacl2015/

Appendix D

Ancora-IULA Tagset Conversion

In the section we explain our conversion of the AnCora (Taulé et al., 2008) and IULA (Marimon et al., 2012) treebanks. Both annotations are based on the EAGLE Tags.[1] For AnCora we use the data that was part of the CoNLL 2009 shared task data sets (Hajič et al., 2009). Adjective degree is not annotated in the AnCora data so we remove these features from the IULA data as well. Numerals such as *dos* 'two' and *tres* 'three' are marked as *z* (numeral) in IULA and *p* (pronoun) or *d* (determiner) in AnCora. We fix this by marking numerals consistently as *z*. IULA marks particles with a specific function argument, while AnCora uses a mood attribute. As the EAGLE specification only specifies the mood variant, we normalize the annotations by converting IULA. IULA annotates all prepositions as `SPS00` (simple, no gender, no number). We normalize by mapping all prepositions in AnCora to the same tag. In some cases, such as for proper nouns and verbs, AnCora uses a gender value *common* and a number value *invariable*, while IULA leaves the corresponding features unspecified. We normalize by removing the two features from the AnCora annotation. Our conversion code is available online.[3]

[1] `http://nlp.lsi.upc.edu/freeling/doc/tagsets/tagset-es.html` (Spanish)

Curriculum Vitae

Education

Since April 2013	**Ph.D. Student, CIS, University of Munich, Germany** Research on Discriminative Tagging and Word Representations
April 2011 - Sept 2013	**Ph.D. Student, Institute for NLP, University of Stuttgart, Germany** Research on Language Modeling, Discriminative Tagging, and Parsing
Oct 2005 - March 2011	**University of Stuttgart, Germany** Diploma in Computer Science, Grade: 1.3 (very good) Specializations: Computer Vision, Intelligent Systems Minor: Physics
June 2004	**Königin-Luise-Gymnasium, Erfurt, Germany** General qualification for university entrance, Grade: 1.7 (good) Majors: Mathematics, Physics

Awards and Honors

July 2013	**Best System in the Shared Task on Parsing Morphologically Rich Languages** With Anders Björkelund, Özlem Çetinoğlu, Richárd Farkas and Wolfgang Seeker
May 2012	**Google European Doctorial Fellowship for Natural Language Processing**

Practical Experience

July 2012 - October 2012	**Research Intern** (Google GmbH, Zurich, Switzerland) Text summarization and headline generation. Development in C++ and MapReduce
Oct 2009 - March 2010	**ERASMUS Student**, Universitat Politècnica de València, Spain Research on offline Handwritten Text Recognition
Oct 2008 - Sept 2009	**Student Trainee** (Robert Bosch GmbH, Stuttgart) GUI development with C++, QT
June 2008 - Sept 2009	**Research Assistant** (Visualisation Institute, University of Stuttgart) GPU-based Visualisation, development C++, QT, OpenGL
April 2007 - Aug 2007	**Teaching Assistant** (Institute of Computer Architecture, University of Stuttgart) Education hardware laboratory (VHDL, CPU design)

Glossary

ABS absolute (discounting). 46

ACC accuracy. 17, 108, 112, 113

ADD additive smoothing. 45

ADJ adjective. 91, 92

ADV adverb. 91, 92

ART article. 97

ATC accumulated tag counts. 11, 63, 64, 127, 128

BFGS Broyden-Fletcher-Goldfarb-Shanno algorithm. 111

BO back-off. 85

CONJ conjunction. 91, 92

CPU central processing unit. 107, 157

CRF conditional random field. 10, 17, 22, 25, 29, 36, 59, 60, 64, 84, 100–103, 108, 109, 111, 114–116, 125, 134, 137

CW Collobert-Weston. 18, 63, 127

CYK Cocke-Younger-Kasami algorithm. 101

DA domain adaptation. 26, 119

DET determiner. 91

EM expectation maximization. 13, 28, 45, 52–54, 85–88, 93, 115

FB forward-backward algorithm. 53, 55, 59, 60

FLM factorized language model. 67

GT Good-Turing. 71

HMM hidden Markov model. 13, 16, 24, 28, 29, 31, 35, 36, 50–54, 56, 59, 60, 64, 70, 71, 83, 84, 87–90, 93, 94, 96, 97, 111, 133, 134

ID in-domain. 17, 18, 118, 120–128, 131, 134, 135

IND indicative. 33, 37

KN Kneser-Ney. 15, 47, 71–73, 75–78, 80, 81, 133

LAS labeled attachment score. 13, 16, 24, 28, 84, 93–97, 133

LDA latent Dirichlet allocation. 62

LL likelihood. 43, 52

LM language model. 18, 48, 62, 126, 127, 131, 135

MA morphological analyzer. 17, 18, 64, 100, 103, 105–107, 111, 118, 119, 124, 128–131, 135

MAP maximum aposteriori. 67, 70, 71

ME maximum entropy. 56–60

ML maximum likelihood. 43, 45, 46, 85

MORPH morphological tag. 18, 105, 108, 111, 119, 121, 123, 128, 131, 135, 143, 147

MRL morphologically rich language. 15, 23–28, 30, 33–35, 65, 66, 83, 115, 133

MTE Multext East. 121, 153

NER named entity recognition. 62, 63, 119

NLP natural language processing. 33–35, 42, 49, 51, 55, 61–63, 99–101, 117, 118, 157

NMOD noun modifier. 16, 96

NN noun. 134

NNP proper noun. 16, 96, 97

NOUN noun. 91, 92

NP proper noun. 84

NUM number. 91

OOD out-of-domain. 17, 18, 118–129, 131, 134, 135

OOV out-of-vocabulary. 15, 17, 66, 68, 72–74, 108, 122, 124, 125, 129

OWL orthant-wise limited-memory. 111

PCFG probabilistic context-free grammar. 63, 64, 83, 84, 114, 115

PDT Prague dependency treebank. 121, 153

POS part-of-speech. 10, 13, 15–18, 28, 62, 63, 68, 69, 83, 84, 86–91, 93, 96, 97, 99, 100, 105–109, 111, 112, 114, 115, 118, 119, 121, 123, 125, 128, 130, 131, 133–135, 143, 145, 147

SGD stochastic gradient descent. 25, 29, 60, 103, 108, 109, 111, 114

SPMRL statistical parsing of morphologically rich languages. 17, 22, 26, 30, 114–116, 120, 134

ST shared task. 114–116, 134

SVD singular value decomposition. 11, 61, 62, 119, 127, 128, 131, 135

TT training time. 17, 108, 112, 113

UAS unlabeled attachment score. 16, 93

UNK unknown. 72, 73

UPOS universal part-of-speech. 90

VERB verb. 91, 92

WB Witten-Bell. 13, 46, 85–90, 93, 134

Bibliography

Meni Adler and Michael Elhadad. An unsupervised morpheme-based HMM for Hebrew morphological disambiguation. In *Proceedings of ACL*, pages 665–672, 2006.

Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of CoNLL*, pages 183–192, 2013.

Rie Kubota Ando and Tong Zhang. A high-performance semi-supervised learning method for text chunking. In *Proceedings of ACL*, pages 1–9, 2005.

Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of ICML*, pages 33–40, 2007.

Giuseppe Attardi and Antonia Fuschetto. Wikipedia Extractor, 2013. `http://medialab.di.unipi.it/wiki/Wikipedia_Extractor`.

Lalit R. Bahl, Peter F. Brown, Peter V. de Souza, Robert L. Mercer, and David Nahamoo. A fast algorithm for deleted interpolation. In *Proceedings of Eurospeech*, pages 1209–1212, 1991.

Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

Jeff A. Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In *Proceedings of NAACL*, pages 4–6, 2003.

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

Anders Björkelund, Özlem Çetinoğlu, Richárd Farkas, Thomas Müller, and Wolfgang Seeker. (re)ranking meets morphosyntax: State-of-the-art results from the SPMRL 2013 shared task. In *Proceedings of SPMRL*, pages 135–145, 2013.

John Blitzer, Ryan T. McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of EMNLP*, pages 120–128, 2006.

Phil Blunsom and Trevor Cohn. A hierarchical Pitman-Yor process HMM for unsupervised part of speech induction. In *Proceedings of ACL*, pages 865–874, 2011.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. The Prague dependency treebank. In *Proceedings of Treebanks*. Springer, 2003.

Bernd Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of Coling*, pages 89–97, 2010.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In *Proceedings of TLT*, 2002.

Peter F. Brown, Vincent J. Della Pietra, Peter V. de Souza, Jennifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992a.

Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1):31–40, 1992b.

Tim Buckwalter. Buckwalter Arabic Morphological Analyzer Version 1.0. *Linguistic Data Consortium*, 2002.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*, pages 173–180, 2005.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

Grzegorz Chrupala. Efficient induction of probabilistic word classes with LDA. In *Proceedings of IJCNLP*, pages 363–372, 2011.

Grzegorz Chrupala, Georgiana Dinu, and Josef van Genabith. Learning morphology with morfette. In *Proceedings of LREC*, 2008.

Kenneth W Church and William A Gale. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech & Language*, 5(1):19–54, 1991.

Alexander Clark. Combining distributional and morphological information for part of speech induction. In *Proceedings of EACL*, pages 59–66, 2003.

Philip Clarkson and Ronald Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of Eurospeech*, 1997.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, 2002.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118, 2004.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of ICML*, pages 160–167, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537, 2011.

Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In *Proceedings of SIGMORPHON*, pages 21–30, 2002.

Mathias Creutz and Krista Lagus. Induction of a simple morphology for highly-inflecting languages. In *Proceedings of SIGMORPHON*, pages 43–51, 2004.

Mathias Creutz and Krista Lagus. Inducing the morphological lexicon of a natural langauge from unannotated text. In *Proceedings of AKRR*, 2005.

Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM TSLP*, 5 (1):3, 2007.

Dóra Csendes, János Csirik, Tibor Gyimóthy, and András Kocsor. The Szeged treebank. In *Proceedings TSD*, pages 123–131, 2005.

Carl de Marcken. *Unsupervised Language Acquisition*. PhD thesis, MIT, Cambridge, MA, 1995.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *JRSS*, 39:1–38, 1977.

Vladimir Eidelman, Zhongqiang Huang, and Mary P. Harper. Lessons learned in part-of-speech tagging of conversational speech. In *Proceedings of EMNLP*, pages 821–831, 2010.

Tomaz Erjavec. Multext-east version 4: Multilingual morphosyntactic specifications, lexicons and corpora. In *Proceedings of LREC*, 2010.

Richárd Farkas, Veronika Vincze, and Helmut Schmid. Dependency parsing of Hungarian: Baseline results and challenges. In *Proceedings of EACL*, pages 55–65, 2012.

Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. The infinite tree. In *Proceedings of ACL*, pages 272–279, 2007.

John R. Firth. A synopsis of linguistic theory, studies in linguistic analysis. *Studies in Linguistic Analysis*, pages 1–31, 1957.

Alexander M. Fraser, Helmut Schmid, Richárd Farkas, Renjing Wang, and Hinrich Schütze. Knowledge sources for constituent parsing of German, a morphologically rich and less-configurational language. *Computational Linguistics*, 39(1):57–85, 2013.

William Gale and Kenneth Church. What is wrong with adding one. *Corpus-based research into language*, 1:189–198, 1994.

Jesús Giménez and Lluís Màrquez. SVMTool: A general POS tagger generator based on support vector machines. In *Proceedings of LREC*, 2004.

Yoav Goldberg and Michael Elhadad. Word segmentation, unknown-word resolution, and morphological agreement in a Hebrew parsing system. *Computational Linguistics*, 39(1):121–160, 2013.

John Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198, 2001.

Joshua Goodman and Jianfeng Gao. Language model size reduction by pruning and clustering. In *Proceedings of Interspeech*, pages 110–113. ISCA, 2000.

Joshua T. Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15 (4):403–434, 2001.

Nizar Habash and Owen Rambow. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of ACL*, pages 573–580, 2005.

Jan Hajič. Morphological tagging: Data vs. dictionaries. In *Proceedings of NAACL*, pages 94–101, 2000.

Jan Hajič. Czech Free Morphology, 2001. URL http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging.

Jan Hajič and Barbora Hladká. Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *Proceedings of Coling*, pages 483–490, 1998.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Stepánek, Pavel Stranák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL*, pages 1–18, 2009.

Zellig S. Harris. Distributional structure. *Word*, 10:146–162, 1954.

Martin Haspelmath and Andrea Sims. *Understanding morphology*. Routledge, 2013.

Dag TT Haug and Marius Jøhndal. Creating a parallel treebank of the old indo-european bible translations. In *Proceedings of LaTeCH*, pages 27–34, 2008.

Fei Huang, Arun Ahuja, Doug Downey, Yi Yang, Yuhong Guo, and Alexander Yates. Learning representations for weakly supervised natural language processing tasks. *Computational Linguistics*, 40(1):85–120, 2014.

Zhongqiang Huang, Vladimir Eidelman, and Mary P. Harper. Improving A simple bigram HMM part-of-speech tagger by latent annotation and self-training. In *Proceedings of NAACL*, pages 213–216, 2009.

Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara I. Cabezas, and Okan Kolak. Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering*, 11(3): 311–325, 2005.

Frederick Jelinek. Interpolated estimation of markov source parameters from sparse data. *Pattern recognition in practice*, 1980.

Nobuhiro Kaji, Yasuhiro Fujiwara, Naoki Yoshinaga, and Masaru Kitsuregawa. Efficient staggered decoding for sequence labeling. In *Proceedings of ACL*, pages 485–494, 2010.

Samarth Keshava and Emily Pitler. A simpler, intuitive approach to morpheme induction. In *Proceedings of MorphoChallenge*, pages 31–35, 2006.

Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

David Klusácek. Maximum mutual information and word classes. In *Proceedings of WDS*, pages 185–190, 2006.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Proceedings of ICASSP*, pages 181–184, 1995.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings MT summit*, volume 5, pages 79–86, 2005.

Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL*, pages 595–603, 2008.

Pavel Kveton. Czech language tokenizer and segmenter, 2013. http://sourceforge.net/projects/czechtok/.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289, 2001.

Michael Lamar, Yariv Maron, Mark Johnson, and Elie Bienenstock. SVD and clustering for unsupervised POS tagging. In *Proceedings of ACL*, pages 215–219. ACL, 2010.

Thomas Lavergne, Olivier Cappé, and François Yvon. Practical very large scale CRFs. In *Proceedings of ACL*, pages 504–513, 2010.

Percy Liang. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology, 2005.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. The Penn Arabic treebank: Building a large-scale annotated Arabic corpus. In *Proceedings of NEMLAR*, pages 102–109, 2004.

Christopher D. Manning. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of CICLing*, pages 171–189, 2011.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of ACL: System Demonstrations*, pages 55–60, 2014.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Montserrat Marimon, Beatriz Fisas, Núria Bel, Marta Villegas, Jorge Vivaldi, Sergi Torner, Mercè Lorente, Silvia Vázquez, and Marta Villegas. The IULA treebank. In *Proceedings of LREC*. ELRA, 2012.

Sven C. Martin, Jörg Liermann, and Hermann Ney. Algorithms for bigram and trigram word clustering. *Speech Communication*, 24(1):19–37, 1998.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of Interspeech*, pages 1045–1048, 2010.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013.

Scott Miller, Jethran Guinness, and Alex Zamanian. Name tagging with word clusters and discriminative training. In *Proceedings of NAACL*, pages 337–342, 2004.

Andriy Mnih and Geoffrey E. Hinton. A scalable hierarchical distributed language model. In *Proceedings of NIPS*, pages 1081–1088, 2008.

Thomas Müller and Hinrich Schütze. Improved modeling of out-of-vocabulary words using morphological classes. In *Proceedings of ACL*, pages 524–528, 2011.

Thomas Müller and Hinrich Schütze. Robust morphological tagging with word embeddings. In *Proceedings of NAACL*, 2015.

Thomas Müller, Hinrich Schütze, and Helmut Schmid. A comparative investigation of morphological language modeling for the languages of the european union. In *Proceedings of NAACL*, pages 386–395, 2012.

Thomas Müller, Helmut Schmid, and Hinrich Schütze. Efficient higher-order CRFs for morphological tagging. In *Proceedings of EMNLP*, pages 322–332, 2013.

Thomas Müller, Richárd Farkas, Alex Judea, Helmut Schmid, and Hinrich Schütze. Dependency parsing with latent refinements of part-of-speech tags. In *Proceedings of EMNLP*, pages 963–967, 2014.

Hermann Ney and Ute Essen. On smoothing techniques for bigram-based natural language modelling. In *Proceedings of ICASSP*, pages 825–828, 1991.

Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

Franz J. Och. Maximum-likelihood-Schätzung von Wortkategorien mit Verfahren der kombinatorischen Optimierung. Studienarbeit, Friedrich-Alexander-Universität.

Franz J. Och. An efficient method for determining bilingual word classes. In *Proceedings of EACL*, pages 71–76, 1999.

Kemal Oflazer and Ilker Kuruöz. Tagging and morphological disambiguation of Turkish text. In *Proceedings of ANLP*, pages 144–149, 1994.

Naoaki Okazaki. CRFsuite: A fast implementation of conditional random fields (CRFs), 2007. URL http://www.chokkan.org/software/crfsuite.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL*, pages 380–390, 2013.

Lluís Padró and Evgeny Stanilovsky. Freeling 3.0: Towards wider multilinguality. In *Proceedings of LREC*, pages 2473–2479, 2012.

Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proceedings of NAACL*, pages 404–411, 2007.

Slav Petrov and Ryan McDonald. Overview of the 2012 shared task on parsing the web. In *Proceedings of SANCL*, 2012.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of ACL*, pages 433–440, 2006.

Slav Petrov, Dipanjan Das, and Ryan T. McDonald. A universal part-of-speech tagset. In *Proceedings of LREC*, pages 2089–2096, 2012.

Lev-Arie Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of CoNLL*, pages 147–155, 2009.

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP*, volume 1, pages 133–142, 1996.

Phillip Roelli. Corpus Corporum, 2014. http://www.mlat.uzh.ch/MLS/.

Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech & Language*, 10(3):187–228, 1996.

Ryan Roth, Owen Rambow, Nizar Habash, Mona T. Diab, and Cynthia Rudin. Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. In *Proceedings of ACL*, pages 117–120, 2008.

Alexander M. Rush and Slav Petrov. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of NAACL*, pages 498–507, 2012.

Anne Schiller. DMOR Benutzerhandbuch. Technical report, IMS, University of Stuttgart, 1995.

Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of NEMLP*, volume 12, pages 44–49, 1994.

Helmut Schmid. Unsupervised learning of period disambiguation for tokenisation. Technical report, IMS, University of Stuttgart, 2000.

Helmut Schmid and Florian Laws. Estimation of conditional probabilities with decision trees and an application to fine-grained pos tagging. In *Proceedings of Coling*, pages 777–784, 2008.

Helmut Schmid, Arne Fitschen, and Ulrich Heid. SMOR: A German computational morphology covering derivation, composition and inflection. In *Proceedings of LREC*, 2004.

Tobias Schnabel and Hinrich Schütze. FLORS: fast and simple domain adaptation for part-of-speech tagging. *TACL*, 2:15–26, 2014.

Hinrich Schütze. Dimensions of meaning. In *Proceedings of Supercomputing*, pages 787–796, 1992.

Hinrich Schütze. Distributional part-of-speech tagging. In *Proceedings of EACL*, pages 141–148, 1995.

Hinrich Schütze and Michael Walsh. Half-context language models. *Computational Linguistics*, 37(4):843–865, 2011.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of SPMRL*, pages 146–182, 2013.

Wolfgang Seeker and Jonas Kuhn. The effects of syntactic features in automatic prediction of morphology. In *Proceedings of EMNLP*, pages 333–344, 2013.

Libin Shen, Giorgio Satta, and Aravind K. Joshi. Guided learning for bidirectional sequence classification. In *Proceedings of ACL*, pages 760–767, 2007.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander J. Smola, and S. V. N. Vishwanathan. Hash kernels for structured data. *JMLR*, 10:2615–2637, 2009.

Noah A. Smith, David A. Smith, and Roy W. Tromble. Context-based morphological disambiguation with random fields. In *Proceedings of EMNLP*, pages 475–482, 2005.

Drahomíra Spoustová, Jan Hajič, Jan Raab, and Miroslav Spousta. Semi-supervised training for the averaged perceptron POS tagger. In *Proceedings of EACL*, pages 763–771, 2009.

Uwe Springmann, Dietmar Najock, Hermann Morgenroth, Helmut Schmid, Annette Gotscharek, and Florian Fink. OCR of historical printings of latin texts: problems, prospects, progress. In *Proceedings of DATeCH*, pages 71–75, 2014.

Andreas Stolcke. SRILM – An extensible language modeling toolkit. In *Proceedings of Interspeech*, 2002.

Jana Straková, Milan Straka, and Jan Hajič. Open-source tools for morphology, lemmatization, POS tagging and named entity recognition. In *Proceedings of ACL: System Demonstrations*, pages 13–18, 2014.

Erik B. Sudderth. *Graphical models for visual object recognition and tracking*. PhD thesis, Massachusetts Institute of Technology, 2006.

Xu Sun, Louis-Philippe Morency, Daisuke Okanohara, and Jun'ichi Tsujii. Modeling latent-dynamic in shallow parsing: A latent conditional model with improved inference. In *Proceedings of Coling*, pages 841–848, 2008.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of EMNLP*, pages 551–560, 2009.

Zsolt Szántó and Richárd Farkas. Special techniques for constituent parsing of morphologically rich languages. In *Proceedings of EACL*, pages 135–144, 2014.

Oscar Täckström, Ryan T. McDonald, and Jakob Uszkoreit. Cross-lingual word clusters for direct transfer of linguistic structure. In *Proceedings of NAACL*, pages 477–487, 2012.

Mariona Taulé, Maria Antònia Martí, and Marta Recasens. Ancora: Multilevel annotated corpora for Catalan and Spanish. In *Proceedings of LREC*, 2008.

Yee Whye Teh. A hierarchical bayesian language model based on Pitman-Yor processes. In *Proceedings of ACL*, pages 985–992, 2006.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of NAACL*, pages 173–180. ACL, 2003.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of ACL*, pages 477–485, 2009.

Joseph P. Turian, Lev-Arie Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394, 2010.

Peter D Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *JAIR*, 37(1):141–188, 2010.

Jakob Uszkoreit and Thorsten Brants. Distributed word clustering for large scale class-based language modeling in machine translation. In *Proceedings of ACL*, pages 755–762, 2008.

Dimitra Vergyri, Katrin Kirchhoff, Kevin Duh, and Andreas Stolcke. Morphology-based language modeling for Arabic speech recognition. In *Proceedings of ICSLP*, volume 4, pages 2245–2248, 2004.

Veronika Vincze, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin, and János Csirik. Hungarian dependency treebank. In *Proceedings of LREC*, 2010.

Martin Volk, Anne Göhring, Torsten Marek, and Yvonne Samuelsson. SMULTRON (version 3.0) — The Stockholm MULtilingual parallel TReebank, 2010. URL http://www.cl.uzh.ch/research/parallelcorpora/paralleltreebanks/smultron_en.html. An English-French-German-Spanish-Swedish parallel treebank with sub-sentential alignments.

David J. Weiss and Benjamin Taskar. Structured prediction cascades. In *Proceedings of AISTATS*, pages 916–923, 2010.

Edward WD Whittaker and Philip C Woodland. Particle-based language modelling. In *Proceedings of ICSLP*, pages 170–173, 2000.

Ian H. Witten and Tim C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4): 1085–1094, 1991.

Peng Xu and Frederick Jelinek. Random forests and the data sparseness problem in language modeling. *Computer Speech & Language*, 21(1):105–152, 2007.

Alexander Yeh. More accurate tests for the statistical significance of result differences. In *Proceedings of Coling*, pages 947–953, 2000.

Deniz Yuret and Ergun Biçici. Modeling morphologically rich languages using split words and unstructured dependencies. In *Proceedings of ACL*, pages 345–348, 2009.

Deniz Yuret and Ferhan Türe. Learning morphological disambiguation rules for Turkish. In *Proceedings of NAACL*, pages 328–334, 2006.

Yi Zhang and Rui Wang. Cross-domain dependency parsing using a deep linguistic grammar. In *Proceedings of ACL*, pages 378–386, 2009.

János Zsibrita, Veronika Vincze, and Richárd Farkas. magyarlanc: A toolkit for morphological and dependency parsing of Hungarian. In *Proceedings of RANLP*, pages 763–771, 2013.