Semi-Supervised Induction of POS-Tag Lexicons with Tree Models

Maciej Janicki
University of Leipzig
NLP Group, Department of Computer Science
Augustusplatz 10, 04109 Leipzig
macjan@o2.pl

Abstract

We approach the problem of POS tagging of morphologically rich languages in a setting where only a small amount of labeled training data is available. We show that a bigram HMM tagger benefits from re-training on a larger untagged text using Baum-Welch estimation. Most importantly, this estimation can be significantly improved by pre-guessing tags for OOV words based on morphological criteria. We consider two models for this task: a character-based recurrent neural network, which guesses the tag from the string form of the word, and a recently proposed graph-based model of morphological transformations. In the latter, the unknown POS tags can be modeled as latent variables in a way very similar to Hidden Markov Tree models and an analogue of the Forward-Backward algorithm can be formulated, which enables us to compute expected values over unknown taggings. We evaluate both the quality of the induced tag lexicon and its impact on the HMM's tagging accuracy. In both tasks, the graph-based morphology model performs significantly better than the RNN predictor. This confirms the intuition that morphologically related words provide useful information about an unknown word's POS tag.

1 Introduction

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

In this paper, we attempt to improve the POS tagging in a setting where only a small amount of labeled training data is available, as well as a significantly larger corpus of unlabeled text. We train a bigram Hidden Markov Model on the labeled part of the corpus and subsequently apply Baum-Welch estimation on the unlabeled part. Additionally, we use the labeled part to train a morphology model in an unsupervised setting. The key idea is to extend the tagger's lexicon with words occurring in the unlabeled part before the Baum-Welch estimation and to pre-guess their possible tags using the morphology model.

The choice of a bigram HMM for tagging is clearly suboptimal. However, our objective here is an initial proof-of-concept demonstrating that tagging in low-resource settings can benefit from unsupervised morphology. The choice of an HMM is dictated by the fact that it is easy to implement, well interpretable in its workings, possible to train on unlabeled data (using the Baum-Welch algorithm) and that it is possible to extend an exist-

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

Proceedings of Recent Advances in Natural Language Processing, pages 507–515,
Varna, Bulgaria, Sep 2–4, 2019.
https://doi.org/10.26615/978-954-452-056-4_060

ing model with new vocabulary without complete re-training. Furthermore, the closely related trigram HMMs used to be state-of-the-art for a long time and are still used in popular tools like HunPos (Halácsy et al., 2007; Megyesi, 2009). Transferring this result to state-of-the-art tagging methods remains a topic for further work.

2 Related Work

2.1 Morphology-Based Induction of POS Lexicons

The idea of guessing possible tags for out-of-vocabulary words based on automatically induced morphological rules was proposed already by (Mikheev, 1997). He introduces a probabilistic model of string transformations, followed by a statistical significance analysis, which learns the correspondence between string patterns and POS tags. A related problem is learning morphological paradigms from inflection tables (Durrett and DeNero, 2013; Ahlberg et al., 2014). Here, complete paradigm tables of annotated word forms are used to learn string transformations between different forms, which are subsequently applied to unknown words to derive their possible tags.

Recently, (Faruqui et al., 2016) approached a task very similar to ours using a graph-based model. Without explicitly modeling morphology, they model structural similarities between words, like a regular affix change or sharing a common affix, as graph edges. They use a discriminative model of label propagation through edges which is similar to the Ising model of intermolecular forces. The induced lexicons are evaluated directly, as well as on how they improve a morphosyntactic tagger and a dependency parser.

2.2 Hidden Markov Tree Models

Hidden Markov Tree (HMT) models are a generalization of Hidden Markov Models, in which every node can have multiple descendents. They were introduced in the context of signal processing by (Crouse et al., 1998; Durand et al., 2004), together with an analogue of the well-known forward-backward algorithm. They remain relatively unknown in the field of language processing – the only mentions that we are aware of are (Žabokrtský and Popel, 2009) and (Kondo et al., 2013).

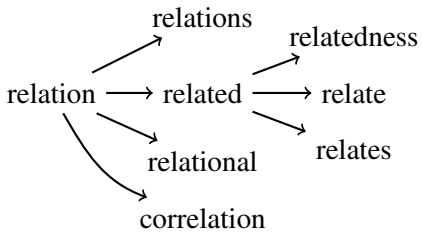

Figure 1: An example tree of word derivations. Edge labels are omitted for better readability. (reproduced from: (Sumalvico, 2017))

3 The Graph Model of Morphology

As a model of morphology suitable for unsupervised training, we use the graph-based model introduced by (Janicki, 2015), which is based on the Whole Word Morphology approach (Ford et al., 1997; Neuvel and Fulop, 2002). It expresses morphological relationships amongst words as a graph, in which words are vertices and morphologically related words are connected with an edge. A concrete morphological analysis of a set of related words is a tree, in which directed edges denote morphological derivation (Fig. 1). In general, the analysis of a vocabulary is a forest, i.e. a set of such trees.[2]

Every edge in the graph is an instance of a *morphological rule*, which describes a pattern applied to whole words. For example, the rule responsible for the pair (*relation*, *related*) might have the following form:

$$/X\text{ion}/ \rightarrow /X\text{ed}/ \qquad (1)$$

In this notation, X is a wildcard that can be instantiated with any string and a pattern between slashes refers to a whole word in its surface form. The rules may also include context: $/X\text{tion}/ \rightarrow /X\text{ted}/$ would be a possible alternative to (1). As there are multiple possible rules that can describe a relationship between a pair of words, the graph edges are defined as triplets of source word, target word and rule.

(Janicki, 2015) introduced a generative probabilistic model for trees such as the one depicted in Fig. 1. It assumes that the roots of the trees are

[2]It is important to point out that such trees are only an intermediary means in determining the strength of morphological relationship between string-similar words. The inference is always based on large samples of trees and finding the 'right' tree, i.e. the one that is consistent with linguistic analysis, is not the goal of the model.

generated independently from a distribution over arbitrary strings, called ρ. Furthermore, every rule r has a fixed probability θ_r of being applied to generate a new word. The probability of the whole forest is defined as follows:

$$Pr(V, E|R, \theta_R) \propto \prod_{v \in V_0} \rho(v) \times \prod_{v \in V} \prod_{r \in R} \prod_{v' \in r(v)} \begin{cases} \theta_r & \text{if } \langle v, v', r \rangle \in E, \\ 1 - \theta_r & \text{if } \langle v, v', r \rangle \notin E \end{cases} \quad (2)$$

V denotes the set of vertices (words) and E the set of (labeled) edges of the graph. R denotes the set of rules and θ_R the vector of probabilities θ_r for the rules from R. Furthermore, $V_0 \subseteq V$ denotes the set of root nodes of the graph and $r(v)$ the set of words that can be derived from word v by application of rule r. Note that the latter might be an empty set if the context on the left-hand side of the rule is not matched. Each possible derivation from $r(v)$ corresponds to a Bernoulli variable with probability θ_r.

(Sumalvico, 2017) describes an unsupervised fitting procedure for this model using Monte Carlo Expectation Maximization algorithm. The computation involves drawing large samples of graphs from the conditional distribution $Pr(E|V, R, \theta_R)$ using a Metropolis-Hastings sampler.

4 Computing POS-Tags

We now turn to applying the model introduced in the previous section to the task of semi-supervised POS tag lexicon induction. The idea is to train a model of morphology on a labeled vocabulary (coming from a tagged corpus) and apply this model to infer the tags for another vocabulary.

The underlying intuition is that morphological relationships between words can give hints about their POS tags and inflectional forms, which are not visible in the isolated forms. For example, consider the following German[3] words: *Fichten* 'spruce.N.PL', *richten* 'judge.V.INF', *rechten* 'right.ADJ.NOM.PL.DEF'.[4] Phonetically and orthographically they are very similar and all include an inflectional suffix *-en*, which is highly ambiguous in German. The knowledge that German nouns are always capitalized does not provide much of a clue, because words belonging to any other part-of-speech may also occur capitalized. Even worse, many further similar words are ambiguous in their meaning and part-of-speech, e.g. *Dichten* ('density.N.PL' or capitalized 'dense.ADJ.NOM.PL.DEF' or 'compose (e.g. a poem).V.INF'), *schlichten* ('simple.ADJ.NOM.PL.DEF' or 'mediate.V.INF').

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

4.1 Applying Tagged Rules to Untagged Words

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

To illustrate the approach with a minimal example, let us assume that our tagset consists of only three tags: NN, VVINF and VVFIN, and that the untagged vocabulary consists of the German words *machen*, *mache*, *macht*. We compute the edge probabilities for every edge that is possible according to the model, under every tagging. For example, the model might consist of the rules and parameters listed in Table 1.

Using those values, we can reason about the possible taggings based on an untagged graph.

[3] It is very difficult to find plausible examples of this phenomenon in English due to its small inflection and very productive zero-affix derivation.

[4] All cited words are ambiguous in their inflectional form (but not part-of-speech). The glosses shown here are picked as examples.

r	θ_r
$/X\text{en}/_{\text{NN}} \rightarrow /X\text{e}/_{\text{NN}}$	0.3
$/X\text{en}/_{\text{VVINF}} \rightarrow /X\text{e}/_{\text{VVFIN}}$	0.01
$/X\text{en}/_{\text{VVINF}} \rightarrow /X\text{t}/_{\text{VVFIN}}$	0.2

Table 1: An example model learnt from a tagged vocabulary.

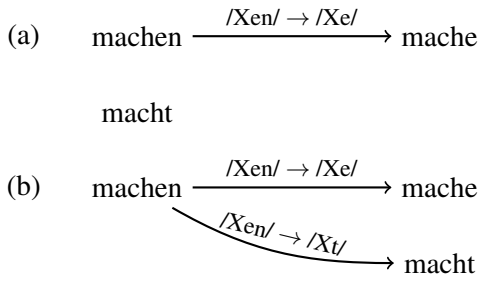

Figure 2: Two possible morphology graphs corresponding to the words *machen*, *mache*, *macht*. What does each of them tell us about the possible tags of those words according to Table 1?

Consider the two graphs shown in Fig. 2. What does each of them say about the possible taggings?

Graph 2a is consistent with either $\{machen_{\text{NN}}, mache_{\text{NN}}\}$ or $\{machen_{\text{VVINF}}, mache_{\text{VVFIN}}\}$, since the only edge in this graph is possible with both labelings. Note that the edge containing noun labels has much higher probability, so this graph suggests a strong preference for the noun hypothesis. It does not say anything about the possible tags of *macht*. On the other hand, the only tagging consistent with the graph 2b is $\{machen_{\text{VVINF}}, mache_{\text{VVFIN}}, macht_{\text{VVFIN}}\}$, since the edge between *machen* and *macht* is only possible if *machen* is a verb infinitive. It is important to notice that adding an edge between *machen* and *macht* diametrically changed the possible taggings for *mache*, although it is not touched by the edge in question. This illustrates how the graph model captures dependencies between the tags across a whole paradigm, although the edge probabilities are local.

Moving on to formalize the above introduction, let the tag of word v be denoted by a random variable T_v. We will be interested in the expected probability of a word v obtaining tag t, given an untagged vocabulary and a tagged morphology model. We call this value $\tau_{v,t}$ and define it as follows:

$$\tau_{v,t} = \mathbb{E}_{E|V,R,\theta}\mathbb{E}_{T|V,E,R,\theta}\delta_{T_v,t} \quad (3)$$

$\delta_{x,y}$ denotes the Kronecker delta. Thus, we take the expectation over all possible graphs for the given vocabulary, and then over all possible taggings for a fixed graph. The inner expectation can be computed exactly by a variant of Forward-Backward algorithm introduced in Sec. 4.2. In order to approximate the outer expectation, we use Markov Chain Monte Carlo sampling over untagged graphs as described by (Sumalvico, 2017). However, the sampling algorithm will need some modifications, as contrary to the original approach, the edge probabilities are not independent of the graph structure. We describe those modifications in Sec. 4.3. Finally, the computed values of $\tau_{v,t}$ will be fed to an already pre-trained HMM to provide it with guesses for the tags of unknown words, before it is reestimated on untagged text. This procedure is described in detail in Sec. 4.4.

4.2 The Forward-Backward Algorithm for Trees[5]

In order to compute $\tau_{v,t} = \mathbb{E}_{T|V,E,R,\theta}\delta_{T_v,t}$ for a fixed graph (V, E), let us recall the well-known Forward-Backward algorithm used for Hidden Markov Models.[6] The HMMs employed for POS tagging operate on sentences, which are linear sequences of words (Fig. 3). The summing over all possible tag sequences is tackled by introducing the so-called *forward probability* (usually written as α) and *backward probability* (β). The forward probability $\alpha_{v,t}$ of a node v with tag t is the joint probability of v and all its predecessors *and* v having tag t, while the backward probability $\beta_{v,t}$ is the probability of all successors of v onwards *provided that* v has tag t. The product of the two is the probability of the whole sequence *and* v having tag t.

The concepts of *successor* and *predecessor* can be applied to a tree model as well (Fig. 4). In this case, $\beta_{v,t}$ is the probability of the subtree rooted in v provided that v has tag t, and $\alpha_{v,t}$ is the probability of the rest of the tree *and* v having tag t. Note that $\alpha_{v,t}$ involves not only the path leading

[5]The algorithm presented in this section is similar, but not identical, to the one presented by (Crouse et al., 1998). The underlying models differ slightly.

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

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

Figure 3: The Forward-Backward computation for a linear sequence in an HMM. $\alpha_{v_6,t} = P(v_1, \ldots, v_6, T_6 = t)$, whereas $\beta_{v_6,t} = P(v_7, v_8, v_9 | T_6 = t)$.

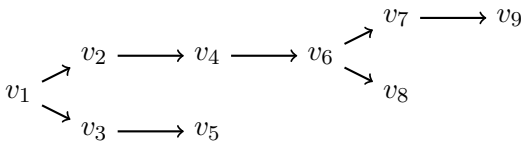

Figure 4: The Forward-Backward computation for a tree. Also here, $\alpha_{v_6,t} = P(v_1, \ldots, v_6, T_6 = t)$ and $\beta_{v_6,t} = P(v_7, v_8, v_9 | T_6 = t)$.

from root to v, but also all side branches sprouting from this path.

We will now derive recursive formulas for forward and backward probabilities for the tree case. For this purpose, we assign a *transition matrix* to each graph edge. For each possible tagging of the source and target node, the transition matrix contains a value $\frac{\theta_r}{1-\theta_r}$, where r is the corresponding tagged rule. Continuing the example from 4.1, the probabilities in Table 1 yield the following transition matrix:

$$T^{(\text{machen,mache,}/Xen/\rightarrow/Xe/)} = \begin{array}{c} \\ \text{NN} \\ \text{VVINF} \\ \text{VVFIN} \end{array} \begin{array}{c} \begin{array}{ccc} \text{NN} & \text{VVINF} & \text{VVFIN} \end{array} \\ \begin{pmatrix} \frac{0.3}{1-0.3} & 0 & 0 \\ 0 & 0 & \frac{0.01}{1-0.01} \\ 0 & 0 & 0 \end{pmatrix} \end{array} \quad (4)$$

Furthermore, let $\lambda_{v,t}$ denote the probability that the node v with tag t is a leaf, i.e. it contains no outgoing edges. Then:

$$\lambda_{v,t} = \prod_{r \in R} \prod_{(v',t') \in r(v,t)} (1 - \theta_r) \quad (5)$$

It is trivial to see that for leaf nodes, $\beta_v = \lambda_v$. For a non-leaf node v, we multiply λ_v by the terms $\frac{\theta_r}{1-\theta_r}$ for each outgoing edge, summed over every possible tagging. In the matrix and vector notation, this corresponds to the following formula:[7]

$$\beta_v = \lambda_v * \prod_{(v,v',r) \in out_{\mathcal{G}}(v)} T^{(v,v',r)} \beta_{v'} \quad (6)$$

[7] Asterisk denotes element-wise multiplication, while dot or no symbol denotes the dot product.

$out_{\mathcal{G}}(v)$ denotes the set of outgoing edges of node v. In order to compute the forward probability of a non-root node v, let us assume that it is derived by the edge (v', v, r). In addition to the forward probability of v' (the parent of v) and the edge deriving v, we also take into account the side branches, i.e. all subtrees rooted in children of v' other than v. The resulting formula is as follows:

$$\alpha_v = \alpha_{v'} * \prod_{\substack{(v',v'',r') \in out_{\mathcal{G}}(v') \\ v'' \neq v}} T^{(v',v'',r')} \beta_{v''} \cdot T^{(v',v,r)} \quad (7)$$

The last remaining issue is the forward probability of root nodes. The generative model defined by (2) contains a distribution $\rho(\cdot)$ over arbitrary strings, from which the string forms of the root nodes are chosen. In the tagged case, we augment this distribution to $\rho(v, t) = \rho_{\text{string}}(v)\rho_{\text{tag}}(t\ |v)$. As in (Sumalvico, 2017)'s experiments, we use a simple character-level unigram model for $\rho_{\text{string}}(\cdot)$. In order to model the distribution $\rho_{\text{tag}}(t|v)$, which predicts a word's tag from its string form, we use a character-level recurrent neural network. Note that the forward probability of root nodes is equal to their probability according to the root model, i.e. $\alpha_{v,t} = \rho(v, t)$.

4.3 Modifications to the Sampling Algorithm

In order to approximate the expected value over possible graphs given a vocabulary, we use the Metropolis-Hastings sampler proposed by (Sumalvico, 2017). The algorithm computes each new graph by proposing a small change in the previous graph. The possible moves are: adding or deleting a single edge, exchanging an edge for another one with the same target node and the so-called 'flip' move. The latter simultaneosly exchanges two edges for two others and is designed as a way to prevent the creation of a cycle while adding an edge. The algorithm subsequently computes an acceptance probability from the probabilities of the changed edges and decides whether to accept the proposed graph as a new sample point.

As we have seen in the analysis of Fig. 2, adding an edge typically has consequences for the whole subtree, in which the edge is added. The values τ_v for all nodes in the subtree may change, which in turn changes the probability of all edges in the subtree. This behavior constitutes a significant difference compared to the original sampling algorithm, in which the edge probabilities were independent of the graph structure and the cost of a

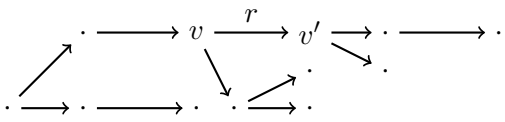

Figure 5: Adding or removing the edge (v, v', r).

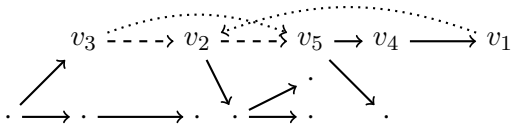

Figure 6: In case of a 'flip' move, the smallest subtree containing all changes is the one rooted in v_3. The deleted edges are dashed, while the newly added edges are dotted. In order to obtain the new β_{v_3}, we recompute the backward probabilities in the whole subtree. α_{v_3} is not affected by the changes. (The node labels are consistent with the definition in (Sumalvico, 2017).)

change could be easily computed from the cost of added and removed edges. However, we can use the forward and backward probabilities to score the changes.

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

Adding or removing a single edge. Consider the graph in Fig. 5, to which the edge (v, v', r) is supposed to be added. Without this edge, we have two separate trees with a total probability expressed by $(\sum_t \alpha_{v,t}\beta_{v,t})(\sum_t \alpha_{v',t}\beta_{v',t})$. After adding this edge, we obtain a single tree. As v obtains a new child node, β_v will change. Let β'_v denote the new value, which can be computed as follows:

$$\beta'_v = \beta_v * T^{(v,v',r)}\beta_{v'} \tag{8}$$

Note that neither α_v nor $\beta_{v'}$ is affected by adding this edge. The probability of the new tree is simply $\sum_t \alpha_{v,t}\beta'_{v,t}$. If the move is accepted, the β values of all nodes on the path from the root to v have to be updated, as well as the α values of all nodes except for this path.

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

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

First, we group all edges that are to be changed (added or deleted) according to the tree, to which they belong (more specifically, according to the root of the tree, to which the edge's source node currently belongs). In each tree, we look for a minimum subtree that contains all the changes (Fig. 6). We build a copy of this subtree with all changes applied and recompute the forward probability for its root and the backward probabilities for the whole subtree. Finally, we use the (newly computed) forward and backward probability of the subtree root to determine the probability of the whole tree after changes.

4.4 Extending an HMM with New Vocabulary

The vectors τ_v obtained from the sampling approach sketched in the previous subsections provide us with a morphologically motivated POS-tag distribution for words from the untagged corpus. We augment the HMM's emission probability matrix with those values for previously unseen words.[8]

5 Experiments

5.1 Experiment Setup

We conducted evaluation experiments for 9 languages: Ancient Greek (GRC), German (DEU), Finnish (FIN), Gothic (GOT), Latin (LAT), Latvian (LAV), Polish (POL), Romanian (RON) and Russian (RUS), using the Universal Dependencies[9] corpora. Each corpus is randomly split into

[8]The values τ_v are conditional probabilities of a tag given word, while the emission probabilities of an HMM are probabilities of a word given tag. We use the Bayes' formula to convert the former to the latter. We obtain the marginal probabilities of tags during the HMM pre-training and assume equal frequencies for unseen words.

[9]http://universaldependencies.org

1	Fitting on the training set.
2	**1** + estimation on the development set.
3	**2** + extension of the vocabulary with random initialization.
4	**2** + extension of the vocabulary with tag guesses provided by a character-based RNN.
5	**2** + extension of the vocabulary with tag guesses provided by the whole-word morphology model.
6	**2** + extension of the vocabulary with gold standard tag guesses.

Table 2: Different setups of the HMM tagger used in the tagging experiment.

training, development and testing dataset in proportions 10%:80%:10%. An HMM tagger is fitted to the (labeled) training data using ML estimation. The training dataset is also used to learn tagged morphological rules. Then, we remove the labels from the development corpus and re-estimate the HMM on this corpus using Baum-Welch algorithm. This step is performed in several configurations: either with or without vocabulary extension. In the latter case, all types occurring in the development corpus, but not known to the HMM (i.e. not occurring in the training corpus) are added to the vocabulary before the estimation. Finally, the tagging accuracy is assessed on the testing dataset. The details of the possible configurations are shown in Table 2.

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

The morphology-based tag guessing approach developed in the previous sections is compared to the guesses made only based on the word's string form. The latter are provided by the distribution $\rho_{\text{tag}}(\cdot)$, i.e. a character-based recurrent neural network, mentioned at the end of Sec. 4.2.

5.2 Evaluation Measures

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

$$\hat{\tau}_{v,t} = \frac{n_{v,t}}{\sum_{t'} n_{v,t'}} \tag{9}$$

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

$$accuracy = \frac{1}{|V|} \sum_{v \in V} \sum_{t} \min\{\tau_{v,t}, \hat{\tau}_{v,t}\} \tag{10}$$

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

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

5.3 Results

Table 3 shows the results of the lexicon evaluation. The figures show clearly that the morphology-based method outperforms the RNN in predicting possible tags for a given word type. Probably the most important reason for that is that the RNN always makes unsharp predictions – it never attributes the whole probability mass to a single tag. On the other hand, the graph-based morphology model often makes unambiguous predictions

Lang.	coarse-grained		fine-grained	
	RNN	WWM	RNN	WWM
GRC	.607	.660	.229	.300
FIN	.507	.643	.255	**.411**
DEU	.616	.676	.154	.210
GOT	.483	**.661**	.157	.308
LAT	.544	**.704**	.236	**.448**
LAV	.506	.646	.240	.368
POL	.587	.695	.196	.315
RON	.540	**.705**	.241	.301
RUS	.682	.769	.330	**.522**

Table 3: Lexicon evaluation.

because of the absence of rules allowing for alternatives (a phenomenon illustrated in Fig. 2). Thus, the latter achieves better scores especially on correctly tagged unambiguous words.

The comparison of tagging accuracies is shown in Tables 4 and 5. The columns correspond to the tagger configurations explained in Table 2. In general, a rise of the score from left to right is to be expected.

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

The most important comparison in this evaluation is between column 4 and 5. This illustrates the benefit of using extrinsic tag guessing (column 5), rather than intrinsic. This results in a consistent improvement, ranging from very slight to significant. The most significant improvements are shown in bold. Finally, column 6 displays what one might expect to be the upper bound on the accuracy: the one that would be achieved if tags were guessed perfectly (i.e. as $\hat{\tau}_v$). Surprisingly, it is not always the highest value in a row. It looks

Lang.	1	2	3	4	5	6
GRC	.669	.742	.732	.789	.791	.857
FIN	.606	.744	.728	.795	**.823**	.838
DEU	.771	.755	.831	.849	.851	.836
GOT	.768	.770	.803	.819	.828	.865
LAT	.743	.808	.816	.825	**.866**	.856
LAV	.670	.723	.731	.796	.803	.848
POL	.715	.787	.745	.781	**.842**	.829
RON	.785	.846	.853	.880	.884	.870
RUS	.809	.877	.877	.903	.905	.914

Table 4: Tagging accuracy with coarse-grained tags.

Lang.	1	2	3	4	5	6
GRC	.566	.464	.569	.628	.638	.699
FIN	.535	.377	.567	.651	**.675**	.717
DEU	.594	.498	.587	.618	.620	.631
GOT	.636	.550	.659	.677	.678	.739
LAT	.590	.493	.617	.660	**.687**	.734
LAV	.585	.471	.594	.657	.668	.713
POL	.554	.437	.575	.625	.627	.679
RON	.712	.713	.759	.764	.761	.824
RUS	.731	.654	.736	.780	.789	.813

Table 5: Tagging accuracy with fine-grained tags.

as if taking into account some wrong taggings during Baum-Welch estimation could accidentally improve the estimation, because the wrong tag might *also* have occurred in the given context. This seems especially plausible for cases like common and proper nouns, which are often confused.

5.4 Discussion

Although the results speak consistently in favor of using morphology-based tag guessing, as well as using tag guessing at all, the benefits are somewhat less clear than one could expect. Especially in the case of fine-grained tags, our expectation was that, due to the discrete nature of morphological rules, at least the tags of unambiguous words would be identified mostly correctly. This was supposed to greatly improve the Baum-Welch estimation, as instead of considering many hundred possible tags, the correct one is already known, which turns the estimation into almost supervised learning. However, we had underestimated the impact of the small size of training corpus on the morphology component. Most fine-grained tags are very rare, so many morphological rules related to such forms are not learnt.

References

Malin Ahlberg, Markus Forsberg, and Mans Hulden. 2014. Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the EACL*. pages 569–578.

Matthew S. Crouse, Robert D. Nowak, and Richard G. Baraniuk. 1998. Wavelet-based statistical signal processing using hidden markov models. *IEEE Transactions on Signal Processing* 46(4):886–902.

Jean-Baptiste Durand, Paulo Gonçalvès, and Yann Guédon. 2004. Computational methods for hidden markov tree models – an application to wavelet trees. *IEEE Transactions on Signal Processing* 52(9):2551–2560.

Greg Durrett and John DeNero. 2013. Supervised learning of complete morphological paradigms. In *Proceedings of NAACL-HLT*. pages 1185–1195.

Manaal Faruqui, Ryan T. McDonald, and Radu Soricut. 2016. Morpho-syntactic lexicon generation using graph-based semi-supervised learning. *TACL* 4:1–16.

Alan Ford, Rajendra Singh, and Gita Martohardjono. 1997. *Pace Pāṇini: Towards a word-based theory of morphology*. American University Studies. Series XIII, Linguistics, Vol. 34. Peter Lang Publishing, Incorporated.

Péter Halácsy, András Kornai, and Csaba Oravecz. 2007. Hunpos – an open source trigram tagger. In *Proceedings of the ACL 2007 Demo and Poster Sessions*. Prague, pages 209–212.

Maciej Janicki. 2015. A multi-purpose bayesian model for word-based morphology. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology – Fourth International Workshop, SFCM 2015*. Springer.

Frederick Jelinek. 1997. *Statistical Methods for Speech Recognition*. MIT Press.

Shuhei Kondo, Kevin Duh, and Yuji Matsumoto. 2013. Hidden markov tree model for word alignment. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*. pages 503–511.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Beáta B. Megyesi. 2009. The open source tagger hunpos for swedish. In *Proceedings of the 17th Nordic Conference of Computational Linguistics (NODALIDA)*.

Andrei Mikheev. 1997. Automatic rule induction for unknown-word guessing. *Computational Linguistics* 23(3):405–423.

Sylvain Neuvel and Sean A. Fulop. 2002. Unsupervised learning of morphology without morphemes. In *Proceedings of the 6th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON)*. pages 31–40.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. pages 88–99.

Maciej Sumalvico. 2017. Unsupervised learning of morphology with graph sampling. In *Proceedings to RANLP 2017*. Varna, Bulgaria.

Zdeněk Žabokrtský and Martin Popel. 2009. Hidden markov tree model in dependency-based machine translation. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics*. Singapore, pages 145–148.