A Poor Man's Morphology for German Transition-Based Dependency Parsing

Daniël de Kok

Seminar für Sprachwissenschaft
University of Tübingen
E-mail: daniel.de-kok@uni-tuebingen.de

Abstract

It has been shown that the performance of statistical dependency parsing of richly inflected languages can be improved by giving the parser access to fine-grained morphological analyses. In this paper we show that similar levels of accuracy can be obtained without explicit morphological analysis, by implicitly learning morphology while training the parser.

1 Motivation

Transition-based dependency parsers [9] have traditionally relied on tokens and part-of-speech tags (or abstractions thereof) to find the best transition given a parser state. Recent work has shown that higher levels of accuracy can be achieved on morphologically-rich languages by using morphological analyses of the input. For instance, Ambati et al. [1] show that the use of features related to case, tense, aspect, and modality improve accuracy substantially on Hindi. Similarly, Marton et al. [12] show that features such as definiteness, person, number, and gender are helpful in parsing Arabic.

Since there is a certain amount of interaction between morphology and syntax in morphologically-rich languages, Lee et al. [10] explore joint morphological disambiguation and dependency parsing using a graph-based dependency parser. Bohnet and Nivre [4] propose a joint model for part-of-speech tagging and transition-based dependency parsing. These works show that joint processing is possible and can avoid error propagation through a natural language processing pipeline.

Since such joint approaches start with the assumption that morphology should be part of the output, they rely on morphology annotations and explicit morphological features. We propose a model that from the user's perspective does not use morphology at all. Instead, we will build up morphological representations 'implicitly' to obtain the same performance as a parser trained with morphology.

In contrast to the recent work of Ballesteros et al. [2], our model uses morphological representations as an addition to word embeddings. Furthermore, rather than using an LSTM recurrent neural network, our model uses a simpler and faster feed-forward neural network.

We believe that learning such morphological features in an unsupervised manner has a couple of benefits: one does not need a treebank with detailed morphology annotations, no morphological analyzer is required, and the parser is not limited to information that is provided by an inventory of morphological features.

2 Parser architecture

The architecture of our parser is inspired by Chen and Manning [5]. Their parser uses the arc-standard transition system [15] and a feed-forward neural network to select the best transition given the current parsing state. The parser state, which consists of a buffer β of unprocessed tokens and a stack σ of tokens currently undergoing processing, is represented as a dense vector that is the concatenation of the embeddings of words, part-of-speech tags, and head relations in the relevant positions of β and σ. This vectorized representation of the parser state forms the input of the neural net. Their approach has several advantages over earlier proposals that used symbolic features and linear discriminative classifiers: using a non-linear activation function on the hidden layer allows the network to infer combinatory features; the use of word embeddings increases lexical coverage significantly; and it is generally faster because it avoids costly feature construction [3].

In our parser, any position on the stack (σ_0^{n-1}) or buffer (β_0^{n-1}) can be addressed. For each token on the stack or buffer, five embedding layers can be consulted: the actual token (TOKEN), its part-of-speech tag (TAG), its morphological analysis (MORPH), its morphological analysis where every morphological feature is encoded as a one-hot vector (MORPH-ONEHOT), and the relation of a token to its head (DEPREL) if available. Moreover, the indirections LDEP and RDEP can be used to address the *n*-th leftmost or rightmost dependent of a token.

Figure 1 shows the (simplified) topology of our network. The input of the network consists of the concatenation of embeddings of the form TOKEN$(\cdot)$, TAG$(\cdot)$, DEPREL$(\cdot)$, MORPH$(\cdot)$ and/or MORPH-ONEHOT$(\cdot)$. The input is fed to a hidden layer using the hyperbolic tangent (*tanh*) activation function.[1] Finally, the output of the hidden layer is fed to the output layer, which uses the *softmax* function to obtain a probability distribution over all possible transitions. A simplification made in Figure 1 is that in reality, there are more hidden layers to train the relation embeddings. We refer to De Kok [7] for more information about the method used to train these embeddings.

[1] We found that the cube activation function proposed by Chen and Manning [5] often results in non-convergence.

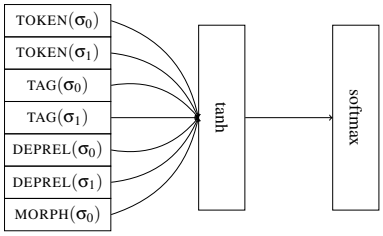

Figure 1: Simplified parser network topology, the number of inputs is reduced for illustrative purposes.

3 Use of a morphological analyzer

In a language processing pipeline where a morphological analyzer is used, analyses can be added as inputs to this parser in two different ways. For instance, suppose that the word at σ_1 is a singular and feminine, the morphological analyzer could assign a complex tag such as:[2]

```
number:singular|gender:feminine
```

The MORPH(σ_1) layer then simply returns the embedding of that morphological tag, which is derived from a large corpus (see Section 5.3). In contrast, the MORPH-ONEHOT(σ_1) layer constructs a feature vector. It decomposes the tag into individual features and represents each feature using a one-hot vector (or a zero vector if the feature is not relevant to a word class). The layer will then output the concatenation of the feature vectors as a representation of all the morphological features. Figure 2 shows the vector for the morphological tag above.

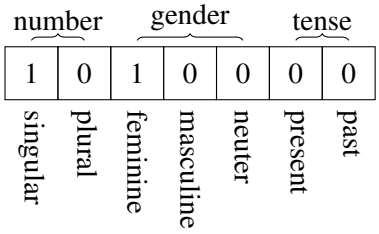

Figure 2: One-hot vector encodings of feature-value pairs for the tag `number:singular|gender:feminine`

[2]For clarity, this example will only use three features.

4 Integrating morphology

As discussed in Section 1 our goal is to integrate morphological analysis in the parser, rather than using information from a morphological analyzer earlier in the pipeline. Our approach has four key points that we will discuss in this section, before showing how morphological analysis fits into our parsing network: (1) characters are represented as (pre-trained) embeddings; (2) words are morphologically represented as the concatenation of prefix and suffix embeddings; (3) for each word that is represented morphologically, a hidden layer is used for feature formation; and (4) the weights of all such hidden layers are tied.

Word representations Although tokens are already represented directly in the parser input using $\text{TOKEN}(\cdot)$, we add a new orthographical input representation $\text{CHAR}(\cdot, p, s)$. This representation is the concatenation of the embeddings of the p prefix characters and s suffix characters of a particular token in the parser state. For instance, $\text{CHAR}(\sigma_0, 2, 2)$ is the concatenation of the character embeddings of the prefix and suffix, both of length 2, of the token that is on top of the stack in the current parser state.

Character embeddings Our motivation for using pre-trained character embeddings, as opposed to e.g. one-hot encoding, is robustness. In case a character was not seen in the training data because it is not part of the German orthography, it could still be mapped closely to similar characters in vector space. To give a motivating example, the capitalized slashed o (Ø) does not occur in the training data. However, Danish names such as *Øresund* occur occasionally in German text. Since the suffix *-und* occurs in multiple word classes, knowing that the word begins with a capital biases the distribution towards singular nouns or proper nouns. In our character embeddings (see Section 5) the letter Ø is indeed clustered with other capital letters (Figure 3).

Feature formation Obviously, an affix of a particular length is not necessarily a linguistically meaningful affix. Our goal is to let the network learn what prefixes or suffixes are meaningful in the context of dependency parsing. To achieve this goal, each input of the form $\text{CHAR}(\cdot, p, s)$ is fed through a hidden layer that uses the logistic function $g(x) = \frac{1}{1+e^{-x}}$ as the activation function.

Weight tying When $\text{CHAR}(\cdot, p, s)$ inputs are extracted for multiple positions in the parser state, as is typically the case, each position obtains its own hidden layer. However, the weights for all such hidden layers are tied. If one hidden layer was used or multiple hidden layers with untied weights, the morphological analyses will differ per parser state position, even if the prefixes and suffixes are exactly the same. The outputs of these hidden morphology layers are then added as additional inputs to the hidden layer discussed in Section 2. We show the topology of the

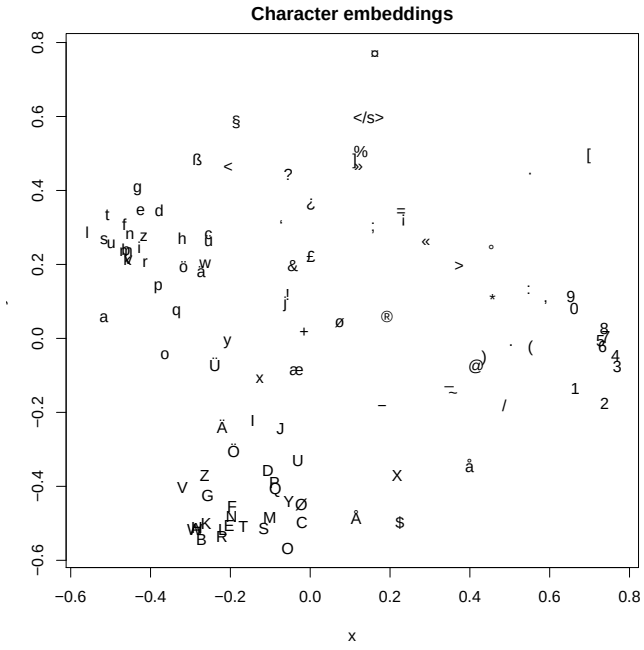

Figure 3: 2-dimensional MDS plot of the character embeddings trained on the TüPP-D/Z corpus. The capitalized slashed o (Ø) is clustered with other capitals.

network that integrates morphology in Figure 4.

The role of the hidden morphological layers can be interpreted in two related ways: (1) as extractors of morphological features that can be used in succeeding layers; or (2) as devices that can be used to create word embeddings such that morphosyntactically similar words are closer in vector space than dissimilar words.

5 Experimental setup

To evaluate the model that we propose, we compare a parser with this implicitly learned morphology (*morph-implicit*) to three parsers with and without access to analyses of a morphological analyzer. We will first describe the morphological analyzer used in our experiment, then we will give a description of the four parsers used in the evaluation. Since the parsers rely heavily on embeddings, we will then describe how the embeddings were trained. Finally, we will give a description of the training and evaluation procedure.

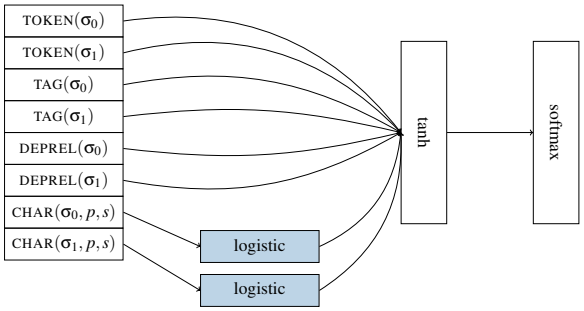

Figure 4: Feed-forward network that implicitly learns morphological features. The concatenated character embeddings of a word (CHAR$(\cdot, p, s)$) form the input of a hidden layer. The weights of such hidden morphology layers (here in blue) are tied.

5.1 Morphological analysis

The morphological analyses in our experiments are provided by RFTagger [18]. RFTagger is an HMM tagger that is tailored to tag sets with a large number of fine-grained tags. To avoid data sparsity, RFTagger splits complex morphological tags into attribute vectors and decomposes the context probabilities into products of attribute probabilities. This decomposition also allows for Markov models of a higher order, making it easier to capture long-distance dependencies. Finally, RFTagger can use a supplemental lexicon for improving coverage of words or word-tag combinations that are not seen in the training data.

For our experiments, we used the model and lexicon that was provided with RFTagger. The output contains the features *category*, *gender*, *case*, *number*, *grade*, *person*, *tense*, *mood*, and *finiteness*.

5.2 Parsers

no-morph The base parser does not have access to morphological information outside the relatively shallow analyses provided by part-of-speech tags. The parser uses pseudo-projective parsing [17] using the stack-projective transition system [16]. The neural network in this parser uses the concatenation of the following embeddings as its input: TOKEN(σ_0^3), TOKEN(β_0^2), TAG(σ_0^3), TAG(β_0^2), TOKEN(LDEP(σ_0^1)), TOKEN(RDEP(σ_0^1)), TAG(LDEP(σ_0^1)), TAG(RDEP(σ_0^1)), DEPREL(σ_0), DEPREL(LDEP(σ_0^1)), and DEPREL(RDEP(σ_0^1)).

morph This parser uses the same configuration as the *no-morph* parser, but adds the analyses of RFTagger to the input using the MORPH-ONEHOT layer, encoding morphological features using a sparse feature vector. The inputs MORPH-ONEHOT(σ_0^1) and MORPH-ONEHOT(β_0) are used in addition to the *no-morph* inputs.

morph-embed Like the *morph* parser, the *morph-embed* parser uses the output of a morphological analyzer. However, in the *morph-embed* parser, embeddings of complex tags are used via the MORPH layer. The inputs $\text{MORPH}(\sigma_0^1)$ and $\text{MORPH}(\beta_0)$ are used for morphology.

morph-implicit This parser also uses the transition system and inputs of the *no-morph* parser, but uses the method for learning morphological features that was described in Section 4. The following embeddings are added to the input vector: $\text{CHAR}(\sigma_0^1, 4, 4)$, $\text{CHAR}(\beta_0, 4, 4)$. We found empirically that using prefix and suffix sizes of 4 provides the best performance on our validation data.

5.3 Embeddings

The token and tag embeddings that are used in all four parsers and the morphology embeddings that are used in the *morph-embed* parser are trained on the TüBa-D/W [6] and TüPP-D/Z [14] corpora. The TüBa-D/W corpus contains 615 million tokens of German Wikipedia text and provides the necessary part-of-speech and morphological annotations. The TüPP-D/Z contains 204 million tokens from the German newspaper *taz*. From TüPP-D/Z we removed the articles that are overlapping with TüBa-D/Z [19]. Moreover, we reprocessed the corpus using the pipeline described by De Kok [6], so that the same annotation scheme is used as in the TüBa-D/W corpus.

The character embeddings for the *morph-implicit* parser are trained by treating each token as a sentence and each character as a token. The character embeddings were trained on only the TüPP-D/Z data, since it is cleaner than the TüBa-D/W.

The embeddings are trained using Wang2Vec [11], which is a modification of Word2Vec [13] that uses a structured skip n-gram model. In contrast to the unstructured model of Word2Vec, this model takes the proximity of words in the window to the focus word into account to create embeddings that are more tailored towards syntax-oriented tasks. For the word, tag, and morphology embeddings, we use the parameters suggested by Ling et al. [11]. For the character embeddings, we use smaller vectors of size 20.

5.4 Training and evaluation

All parsers are trained and evaluated with the dependency version [20] of the TüBa-D/Z release 9 [19]. The treebank, consisting of 85358 sentences and 1569916 tokens, is split into five interleaved parts. Two parts are used for training and validation. The remaining three parts are held out and used for evaluation.

6 Results

Table 1 shows labeled attachments scores for the three parsers. As we can see, all three parsers that use morphology perform better than the *no-morph* parser (signif-

icant at $p < 0.0001$). The difference is quite large if we take into account that the word embeddings already give the parsers very high lexical coverage — 97.13% of the tokens and 75.73% of the types in the evaluation data are known. The *morph-implicit* model that we propose in this paper is not only competitive the models that use a morphological analyzer, it even outperform them slightly (significant at $p < 0.05$).

Parser	LAS
no-morph	89.08
morph	89.35
morph-embed	89.40
morph-implicit	89.49

Table 1: Labeled attachment scores for the parsers with/without morphology.

As expected expected, the forward passes of the neural network of the *morph-implicit* parser are more expensive, since it applies extra hidden layers for morphological analysis. Since we did not implement the precompute trick [8] yet, for which Chen and Manning [5] report an order of magnitude speedup, the performance of the parser is highly dependent on having a performant BLAS library and a CPU that provides wide SIMD instructions. For this reason, we list the running time of each parser relative to the parser that does not use morphology in Table 2.[3]

Parser	Running time	Running time (+ RFTagger)
no-morph	1.00	
morph	1.58	2.73
morph-embed	1.03	2.18
morph-implicit	2.42	

Table 2: Running times for the parsers. The parsers using morphology have similar performance when including overhead of the morphological analyzer.

The use of morphology embeddings (*morph-embed*) has virtually no overhead compared to the *morph* parser. Encoding the morphology features using a sparse vector is, however, is over 1.5 times slower. The reason is that the *morph-embed* parser simply copies the embeddings into the input vector, while the *morph* parser has to split the complex tag first. The performance of the *morph* parser could be improved by pre-computing the vectors for complex tags. The parser proposed in this work (*morph-implicit*) is nearly 2.5 times slower as a result of the extra hidden layer(s). However, this comparison does not provide the complete picture, since the *morph* and *morph-embed* require the output of a morphological analyzer. The

[3]For reference, the *no-morph* parser processes 315 sentences per second using Intel Math Kernel Library using 4 threads on an Intel Xeon E5-2650 v3 @ 2.30GHz.

second column lists the running times of these parsers including morphological analysis. As we can see, the overall running times of the morphological parsers are roughly in the same ballpark.

7 Conclusion

We proposed a model for implicit morphological analysis, which can compete with the use of a morphological analyzer. This opens up the possibility to make competitive parsers using treebanks without morphological annotations, morphological analyzers, or hand-constructed morphology features.

An open question is if or how the model could be adjusted for languages that rely on infix morphology. Another interesting question is if the morphology features constructed in the network could also be used for morphological analysis where morphological tags are part of the output.

References

[1] Bharat Ram Ambati, Samar Husain, Joakim Nivre, and Rajeev Sangal. On the role of morphosyntactic features in hindi dependency parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 94–102. Association for Computational Linguistics, 2010.

[2] Miguel Ballesteros, Chris Dyer, and Noah A Smith. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 2015.

[3] Bernd Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97. Association for Computational Linguistics, 2010.

[4] Bernd Bohnet and Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1455–1465, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[5] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750, 2014.

[6] Daniël de Kok. TüBa-D/W: a large dependency treebank for german. In *Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT13)*, page 271, 2014.

[7] Daniël de Kok. Bootstrapping a neural net dependency parser for German using CLARIN resources. In *Proceedings of the CLARIN 2015 conference*, 2015.

[8] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1370–1380, 2014.

[9] Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

[10] John Lee, Jason Naradowsky, and David A Smith. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 885–894. Association for Computational Linguistics, 2011.

[11] Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL), Denver, CO*, 2015.

[12] Yuval Marton, Nizar Habash, and Owen Rambow. Improving arabic dependency parsing with lexical and inflectional morphological features. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 13–21. Association for Computational Linguistics, 2010.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

[14] Frank Henrik Müller. Stylebook for the tübingen partially parsed corpus of written German (TüPP-D/Z). In *Sonderforschungsbereich 441, Seminar für Sprachwissenschaft, Universität Tübingen*, volume 28, page 2006, 2004.

[15] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics, 2004.

[16] Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 351–359, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[17] Joakim Nivre and Jens Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics, 2005.

[18] Helmut Schmid and Florian Laws. Estimation of conditional probabilities with decision trees and an application to fine-grained POS tagging. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 777–784. Association for Computational Linguistics, 2008.

[19] Heike Telljohann, Erhard Hinrichs, and Sandra Kübler. The TüBa-D/Z treebank: Annotating German with a context-free backbone. In *In Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004*. Citeseer, 2004.

[20] Yannick Versley. Parser evaluation across text types. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT 2005)*, 2005.