A poor man's morphology for German transition-based dependency parsing

Daniël de Kok

Take-home message

- Using morphological information improves dependency parsing of German.
- You do not need a morphological analyzer to provide this information.

Neural net dependency parsing

Transition-based dependency parsing

Transition-based dependency parsers are (linear-time) shift-reduce parsers that typically rely on three operations:

- LEFT-ARC$_r$: Introduce a relation r pointing leftward.
- RIGHT-ARC$_r$: Introduce a relation r pointing rightward.
- SHIFT: Move the next token from the buffer to the processing stack.

Guided parsing

We choose the next operation using a classifier:

- Labels: $\{\text{LEFT-ARC}_r | r \in R\} \cup \{\text{RIGHT-ARC}_r | r \in R\} \cup \{\text{SHIFT}\}$
- Inputs: decompositions the parser state: dependencies found so far, tokens on the processing stack, and tokens on the buffer.

Guided parsing

We choose the next operation using a classifier:

- Labels: $\{\text{LEFT-ARC}_r | r \in R\} \cup \{\text{RIGHT-ARC}_r | r \in R\} \cup \{\text{SHIFT}\}$
- Inputs: decompositions the parser state: dependencies found so far, tokens on the processing stack, and tokens on the buffer.

Typical classifiers:

- Linear SVM
- Kernel SVM

Recent advancements

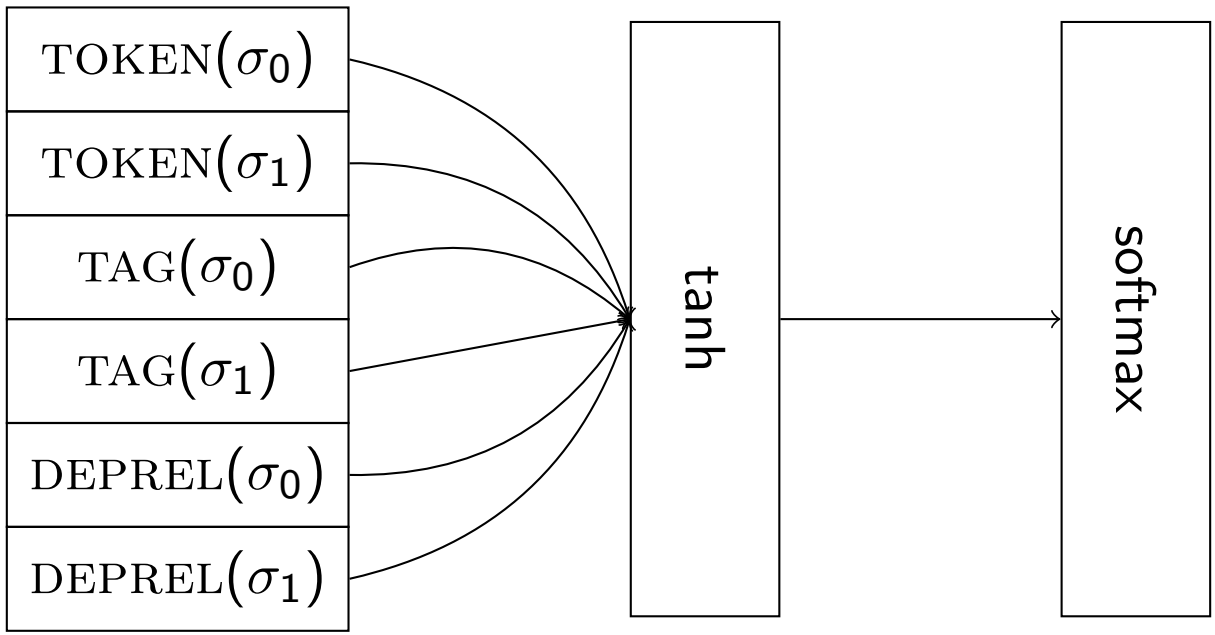
Stenetorp, 2013 and Chen and Manning, 2014 propose two important changes to the typical transition classifier:

- Represent the input using embeddings.
- Use a feed-forward neural net as the classifier.

Embedding

- A word embedding is a dense vector $\mathbb{R}^n$ that is a distributed representation of a word.
- The representation is typically learned as part of some task. E.g.:
  - Predicting a word given its context.
  - Predicting a context given a word.
- Goal: place words that are semantically or syntactically similar close in vector space.
- Generalizes to part-of-speech tags, dependency relations, etc.
- Unsupervised: relatively cheap to add more training data.

Neural Net Transition classifier



dparnn

Our parser (dparnn) follows the architecture of Chen and Manning, 2014:

- Layers: TOKEN, TAG, DEPREL
- Addresses: buffer and stack
- Indirections: LDEP and RDEP

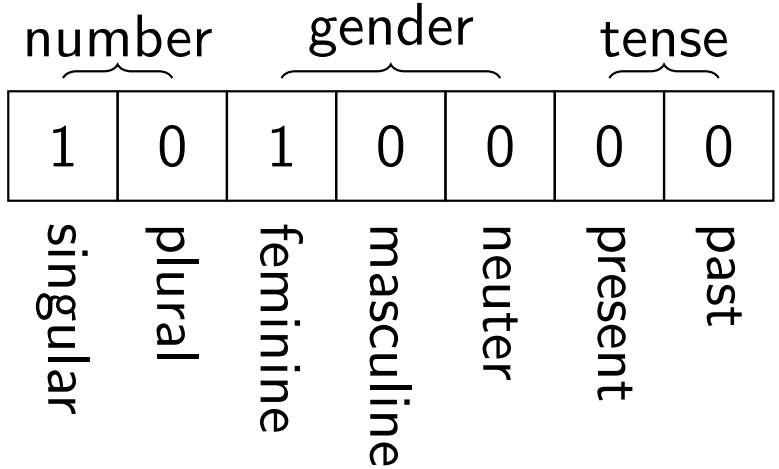
(With some modifications, see De Kok, 2015.)

Using morphological analyses

Introduction

- Prior work has shown that adding more fine-grained morphological information can improve parser accuracy (e.g. Marton et al., 2010 and Ambati et al., 2012)
- Two obvious ways to integrate the output of a morphological analyzer in a neural net dependency parser:
  1. Use a traditional feature vector to represent morphological features (one-hot encoding).
  2. Learn embeddings of morphological tags, use the embeddings as input.

One-hot encoding



Morphological tag embedding

$$\textit{number:singular|gender:feminine} \rightarrow \mathbb{R}^n$$

Experiment

- Training and evaluation on the dependency version of TüBa-D/Z release 9.
- Morphological annotations provided by RFTagger (Schmid & Laws, 2008)
- The competition:
  - **no-morph**: Typical inputs, see paper for more details.
  - **morph**: input as **no-morph** + morphological analyses encoded using one-hot vectors.
  - **morph-embed**: input as **no-morph** + morphological analyses as embeddings.

Results

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

Observations

- The difference between the **no-morph** and **morph** parsers is significant at $p < 0.0001$.
- Welcome improvement for a well-researched task.
- We should not expect miracles. Word embeddings hav a very high coverage of the test data:
  - 97.13% of tokens
  - 75.73% of types

Implicitly learned morphology

Motivation

Can we obtain the same improvements without a morphological analyzer? Benefits:

- Works for languages/treebanks for which no morphological analyzer is available.
- Less error propagation in the language processing pipeline.
- Exploit that syntax can inform morphology and vice versa.

New orthographical input representation

$$\textsc{char}(\cdot, p, s)$$

- Concatenation of:
  - embeddings of p prefix characters; and
  - embeddings of s suffix characters

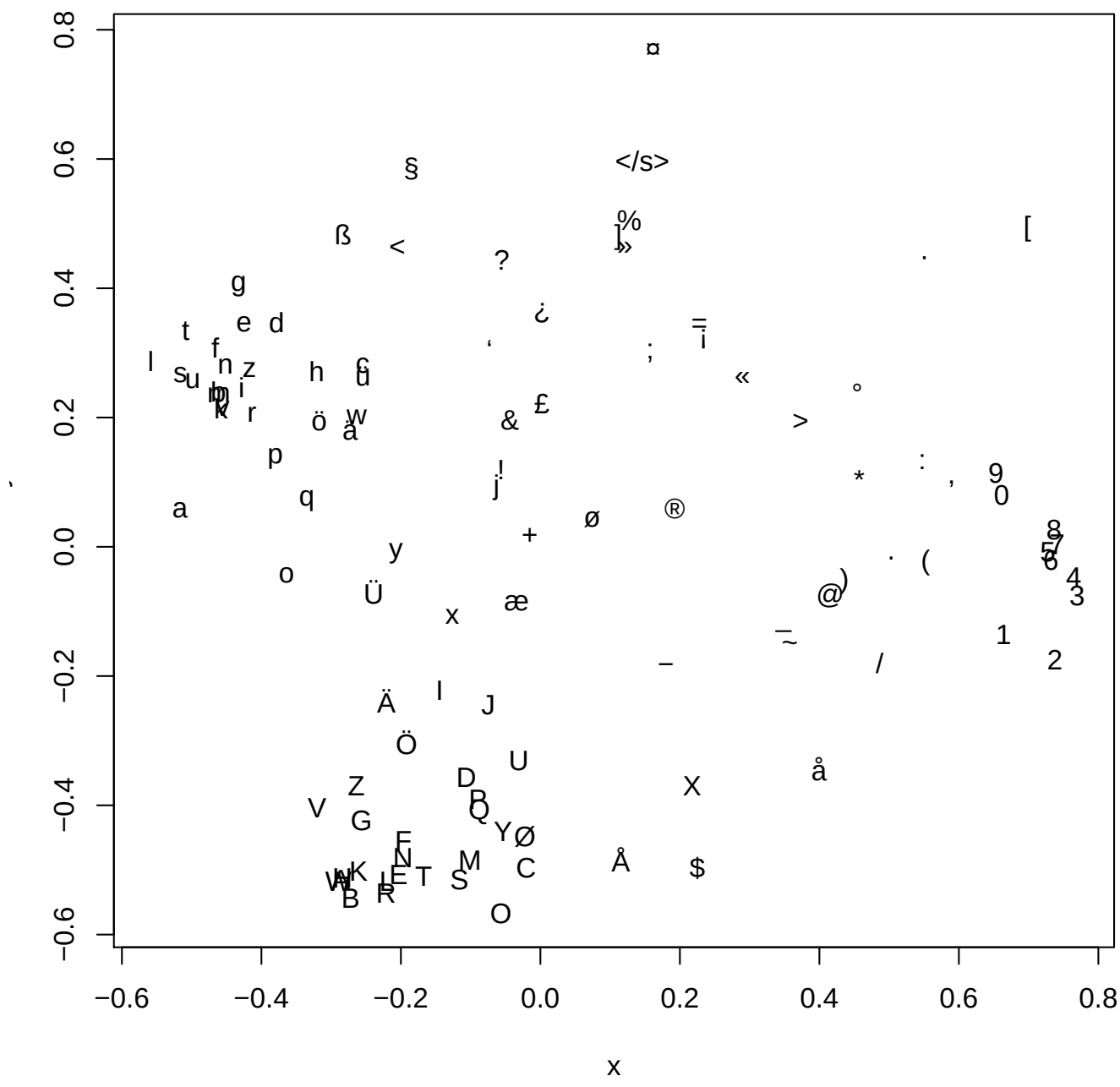
  For a particular token in the parser state.
- For example, $\textsc{char}(\sigma_0, 2, 2)$, is the concatenation of: the embeddings of the prefix and suffix of length 2 of the token on top of the parser stack.

Character embeddings

- $\textsc{char}(\cdot, p, s)$ uses character embedding for robustness.
- For instance:
  - The capitalized slashed o (Ø) does not occur in treebank used for training the parser.
  - Danish names such as *Øresund* occur occasionally in German text.
  - The suffix *-und* occurs in multiple word classes.
  - Knowing that the word begins with a capital letter biases the distribution towards singular nouns or proper nouns.
  - How do we know that Ø is a capital letter? It is clustered with other capital letters.

Learning character embeddings

- Take a large corpus.
- Learn to predict the characters in the context of the target character.
- Implementation:
  - Use software for learning word embeddings.
  - A word is a sentence.
  - A character is a word.


Character embeddings
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
x

Feature formation

- An affix of a particular length is not necessarily a linguistically meaningful affix.
- Goal: let the network learn what affixes are meaningful in the context of dependency parsing.
- Feed each input of the form $\textsc{char}(\cdot, p, s)$ through a hidden layer with a non-linear activation function.
- We had the most success with the logistic function ($g(x) = \frac{1}{1+e^{-x}}$)

Weight tying

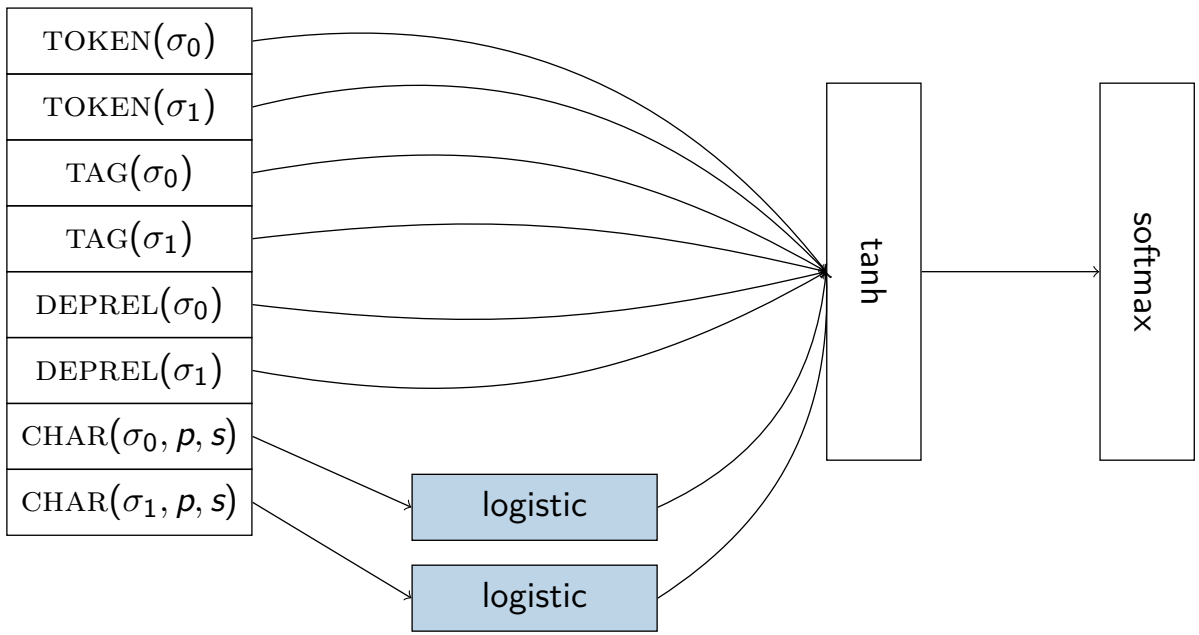
- Each $\textsc{char}(\cdot, p, s)$ input has its own hidden layer.
- We tie the weights among all such hidden layers.
- With untied weight, the morphological analyses will differ per parser state possition, even for equivalent pre-/suffixes.

Role of hidden morphological layers

The morphological layers can be interpreted in two related manners:

1. As extractors of morphological features that can be used in succeeding layers.
2. As devices that can be used to create word embeddings, such that morphosyntactically similar words are closed in vector space than dissimilar words.

Network topology with implicit morphology



Updated parser

The **morph-implicit** uses the feature set of the **no-morph** parser, with the following additions:

- $\textsc{char}(\sigma_0^1, 4, 4)$
- $\textsc{char}(\beta_0, 4, 4)$

Results

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

Results

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

Note: the difference between **morph-embed** and **morph-implicit** is significant at $p < 0.05$.

Parser performance

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	

Note: The precompute trick (Devlin et al., 2014) is not used yet.

Conclusion

- We proposed a model for implicit morphological analysis that can compete with the use of a morphological analyzer.
- Opens up the possibility to make competitive parsers using treebanks without morphological annotations or morphological analyzers.

Open questions

- Can the model be adjusted for languages that rely on infix morphology?
- Could this approach be extended for joint morphological analysis and dependency parsing?
- Arguably, models that exploit embeddings and feed-forward neural nets are more opaque. We need better techniques to understand e.g. what complex features are formed.

Thank you!

Results

Parser	LAS	UAS
no-morph	89.08	91.62
morph	89.35	91.76
morph-embed	89.40	91.77
morph-implicit	89.49	91.88