Modeling Input Uncertainty in Neural Network Dependency Parsing

Rob van der Goot  
University of Groningen  
r.van.der.goot@rug.nl

Gertjan van Noord  
University of Groningen  
g.j.m.van.noord@rug.nl

Abstract

Recently introduced neural network parsers allow for new approaches to circumvent data sparsity issues by modeling character level information and by exploiting raw data in a semi-supervised setting. Data sparsity is especially prevailing when transferring to non-standard domains. In this setting, lexical normalization has often been used in the past to circumvent data sparsity. In this paper, we investigate whether these new neural approaches provide similar functionality as lexical normalization, or whether they are complementary.

We provide experimental results which show that a separate normalization component improves performance of a neural network parser even if it has access to character level information as well as external word embeddings. Further improvements are obtained by a straightforward but novel approach in which the top-N best candidates provided by the normalization component are available to the parser.

1 Introduction

Recently, neural network dependency parsers (Chen and Manning, 2014; Dyer et al., 2015; Kiperwasser and Goldberg, 2016) obtained state-of-the-art performance for dependency parsing. These parsers incorporate character level information (de Lhoneux et al., 2017a; Ballesteros et al., 2015; Nguyen et al., 2017) and can more easily exploit raw text in a semi-supervised setup. These new methods are especially beneficial for words not occurring in the training data. In practice, such unseen words often are spelling mistakes, or alternative spellings of known words. In more classical parsing models, these unseen words were usually clustered using ad-hoc rules. For non-standard domains, the number of unseen words is much larger. To minimize the degradation in performance, lexical normalization is often used. Lexical normalization is the task of converting non-standard input to a more standard format. Previous work has shown that this is beneficial, in particular for parsing social media data (Foster, 2010; Zhang et al., 2013; van der Goot and van Noord, 2017b).

This leads to the question whether normalization is indeed no longer required for these modern character-based neural network parsers, or whether normalization is capable of solving problems beyond the scope of this type of neural network parsers.

Our main contributions are:

- We show that using normalization as preprocessing improves parser performance for non-standard language, even if pre-trained embeddings and character level information are used.
- We propose a novel technique to exploit the top-N candidates provided by the normalization component, and we show that this technique leads to a further increase in parser performance.
- A treebank containing non-standard language is created to evaluate the effect of normalization on parser performance. The treebank consists of 10,005 tokens annotated with lexical normalization and Universal Dependencies (Nivre et al., 2017). The treebank has been made publicly available.

2 Related Work

Early work on parser adaptation focused on relatively canonical domains, like biomedical data (McClosky and Charniak, 2008). More recently, there has been an increasing interest in parsing of the notoriously noisy domain of social media. A lot of previous work is orthogonal to our
Original word | new | pix | coming | tomorrow
---|---|---|---|---
Cand. 1 (p1) | new (0.95) | pix (0.79) | coming (0.57) | tomorrow (0.54)
Cand. 2 (p2) | news (0.03) | selfies (0.08) | coming (0.43) | tomorrow (0.39)
Cand. 3 (p3) | knew (0.01) | pictures (0.06) | combing (0.01) | tomorrow’s (0.02)

Table 1: Output of the normalization model for the example sentence “new pix coming tomorrow” including candidate probabilities. Only the top-3 candidates are shown here.

approach, as it focuses on adaptation of the training data (Foster et al., 2011; Khan et al., 2013; Kong et al., 2014; Blodgett et al., 2018). In the remainder of this section we will shortly review work which evaluated the effect of normalization on dependency parsing.

Zhang et al. (2013) tune a normalization model for the parsing task, and show performance improvement on a silver treebank obtained from manually normalized data. Daiber and van der Goot (2016) use an existing normalization model as pre-processing for a graph-based dependency parser, and show a small but significant performance improvement. In the shared task of parsing the web (Petrov and McDonald, 2012) held at SANCL 2012, some teams used a simple rule-based normalization, but the effect on final performance remained untested.

Baldwin and Li (2015) examined the theoretical impact of different normalization actions on parsing performance. To this end they use manual normalization. They show that edits beyond the word level can also be crucial for parsing (e.g. insertion of copulas and subjects). However, these are difficult to obtain automatically.

Note that all this previous work, except for Blodgett et al. (2018), is based on traditional feature-based dependency parsers, whereas we focus on neural network parsing.

3 Method

In this section we will first shortly review the two models we will combine: a lexical normalization model and a neural network parser. Then we describe how they can be combined.

3.1 Normalization

In this work we use an existing normalization model: MoNoise (van der Goot and van Noord, 2017a). This model is based on the observation that normalization requires a variety of different replacement actions. For these different actions, different modules are used to generate candidates, including: the Aspell spell checker, word embeddings and a lookup list generated from the training data. Features from these generation modules are complemented with N-gram features from canonical data and non-canonical data. A random forest classifier is used to score and rank the candidates. In this work, we use the top-N candidates and convert the confidence scores of the classifier to probabilities. An example of this output is shown in Table 1.

We train MoNoise on 2,577 tweets annotated with normalization by Li and Liu (2014), which only contains word-word replacements. In our initial experiments, we noted that the normalization model wrongfully normalized some words due to the different tokenization in the treebank (e.g. “can’t”), because these do not occur in the normalization data. We manually created a list of exceptions, which are not considered for normalization process.

3.2 Neural Network Parser

As a starting point, we use the shift-reduce UU-Parse 2.0 (de Lhoneux et al., 2017b; Kiperwasser and Goldberg, 2016). This parser uses the Arc-Hybrid Transition system (Kuhlmann et al., 2011). Words are first converted to continu-
ous vectors, which are then processed through a Bidirectional Long-Short Term Memory network (BiLSTM) (Graves and Schmidhuber, 2005) before they are passed on to the parsing algorithm. The decision whether to shift, reduce or swap is made by a multi-layer perceptron with one hidden layer. The BiLSTM is trained jointly with the parser. In our setup, this means that we use the best normalization sequence as input to the BiLSTM for word vectors of the top-N candidates are merged using linear interpolation:

The vectors of the original words of a sentence. The normalization model is optimized on the training data, \( \vec{c}_i \) is the result of a separate BiLSTM run over the characters of word \( i \) and \( \vec{c}_i^e \) is the external vector; it is obtained from external embeddings which are trained on huge amounts of raw texts. In this work we use the same word embeddings as used by the normalization model (van der Goot and van Noord, 2017a), which are trained on 760,744,676 tweets using word2vec (Mikolov et al., 2013).

### 3.3 Adaptation Strategy

**Notation** We use \( \vec{w}_0...\vec{w}_n \) to represent the vectors of the original words of a sentence. The vectors of the normalization candidates are represented by \( \vec{n}_{ij} \), where \( i \) is the index of the original word in the sentence, and \( j \) is the rank of the candidate. The corresponding probability as given by the normalization model is \( p_{ij} \). We use \( \vec{g}_i \) for the vector of the manual normalization of word \( i \).

Our baseline setup is to simply use the vector of the original word:

**Orig:** \( \vec{v}_i = \vec{w}_i \)

The most straightforward use of normalization is to use the best normalization sequence as input to the parser. In our setup, this means that we use the vector of the best normalization candidate for each position:

**Norm:** \( \vec{v}_i = \vec{n}_{i0} \)

To give more information to the parser, we will exploit the top-\( N \) candidates of the normalization model. The vectors of the top-\( N \) candidates are merged using linear interpolation:

**Integrated:** \( \vec{v}_i = \sum_{j=0}^{n} p_{ij} \cdot \vec{n}_{ij} \)

An interesting property of this integration approach is that it does not influence the size of the search space, so the effect on complexity of the parsing algorithm is negligible. The only extra runtime compared to \( \text{ORIG} \) originates from running the normalization model.

Finally, we include a theoretical upperbound of the effect of normalization, which uses manually annotated normalization:

**GOLD:** \( \vec{v}_i = \vec{g}_i \)

### 4 Data

To test the effect of normalization, we need a treebank containing non-standard language, preferably with a corresponding training treebank from a more standard domain. Since the existing treebanks are not noisy enough (Foster et al., 2011; Kaljahi et al., 2015)\(^3\) or do not have a corresponding training treebank in the same annotation format (Kong et al., 2014; Daiber and van der Goot, 2016) we annotate a small treebank for development and testing purposes\(^4\). We choose to use the Universal Dependencies 2.1 annotation format (Nivre et al., 2017), since the annotation efforts on the the English Web Treebank (Silveira et al., 2014) provide suitable training data. This treebank already contains web specific phenomena like URL’s, E-Mail addresses and emoticons, so we do not have to create special annotation guidelines and the parser can learn these phenomena from the training data.

Our treebank consists of tweets, taken from Li and Liu (2015). The tweets in this dataset originate from two sources: the LexNorm corpus (Han and Baldwin, 2011), which was originally annotated with normalization, and a corpus originally annotated with POS tags (Owoputi et al., 2013), which were originally normalized the development data from Foster et al. (2011), were even less words were in need of normalization. Li and Liu (2015) complemented this annotation for both datasets, so that they both have a normalization layer and a POS layer. To avoid overfitting on a specific filtering or time-frame we use the data collected by Owoputi et al. (2013) as development data and LexNorm as test data. We only keep the tweets which are still available on Twitter, resulting in a dataset of 305 development and

\(^3\)Kaljahi et al. (2015) only normalize 3.6%, and we manually normalized the development data from Foster et al. (2011), were even less words were in need of normalization.

\(^4\)It should be noted that two other suitable Twitter treebanks in the UD format where created in parallel to our treebank (Liu et al., 2018; Blodgett et al., 2018), which were released after submission of this paper.
327 test tweets (10,005 tokens in total). It should be noted that these corpora were filtered to contain domain-specific phenomena and non-standard language, and thus provide an ideal testbed for our experiments but are not representative of the whole Twitter domain.

Tokenization and normalization are first re-annotated, because the Universal Dependencies format requires treebank specific tokenization. To avoid parser bias, dependency relations are annotated from scratch. For more details on annotation decisions for domain-specific structures, we refer to the appendix.

MoNoise reaches 90% accuracy on the word level for the normalization task for our development data. In this dataset, 18% of all words are in need of normalization, so a baseline which simply copies the original words would reach an accuracy of 82%. The most common mistakes made by MoNoise are due to treebank specific normalizations, like ‘na’ → go. However, these also occur in the training treebank, so normalization is not crucial.

5 Evaluation

In this section, we first use the development data to compare the effect of the different normalization settings with the use of character level information and external embeddings. Secondly, we confirm our main results on the test set. Thirdly, we test if our model is sensitive to over-normalization on standard data. Finally, we perform some analysis to examine why normalization is beneficial. All scores reported in this section are obtained using the CoNLL 2017 evaluation script (Zeman et al., 2017). In Section 5.1 the results are the average over ten runs, using a different seed for the BiLSTM and the shuffling of the training data. In the remainder of this section, the best model is used to simplify interpretation. The parser is trained using default settings (de Lhoneux et al., 2017b).

In our initial experiments, it became apparent that the parser often considered a username mention or retweet in the beginning of the tweet as root, resulting in a propagation of errors. Because we want to exclude any influences from this simple construction, we added an heuristic to our parser which exclude usernames and retweets in the beginning of a tweet, and connects them to the root after parsing. We use this heuristic in all experiments.

![Figure 2: The effect of normalization on LAS for the different parsing models on the development data.](image)

5.1 Normalization Strategies

The results of the different parser and normalization settings on the development data are plotted in Figure 2. Using external embeddings (⃗e) results in a much bigger performance improvement compared to using character level information (⃗c). Adding character level embeddings on top of external embeddings only leads to a very minor improvement. This can partly be explained by the coverage of 98.4% of the embeddings on the development data.

In the settings without external embeddings, the direct use of normalization (NORM) results in a improvement of approximately 3 LAS points. However, when external embeddings are included the improvement becomes more than twice as small, indicating that the approaches target some common issues, but are also complementary to each other. When external embeddings and normalization are already used, the character level embeddings slightly harm performance. Integration of the normalization (INTEGRATED) consistently results in a slightly higher LAS compared to direct normalization. Interestingly, gold normalization still performs substantially better compared to automatic normalization.

5.2 Test Data

Table 2 shows the results of the parser with external embeddings and character embeddings (using the best seed from the development data), for the different normalization strategies on the test data. These results confirm the observations on the development data: normalization helps on top of ex-
**Table 2**: LAS scores for the Twitter test data. Statistically significant compared to the previous row at \( P < 0.05 \) using a paired t-test.

<table>
<thead>
<tr>
<th>Model</th>
<th>UAS</th>
<th>LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIG</td>
<td>69.63</td>
<td>59.64</td>
</tr>
<tr>
<td>NORM</td>
<td>70.51</td>
<td>61.76*</td>
</tr>
<tr>
<td>INTEGRATED</td>
<td>70.62</td>
<td>62.30*</td>
</tr>
<tr>
<td>GOLD</td>
<td>70.71</td>
<td>62.33</td>
</tr>
</tbody>
</table>

5.3 Robustness

As stated in Section 4, our development and test data is filtered to be very non-standard. However, it is undesirable to have a parser that performs bad on more standard texts. Hence, we also tested performance on the English Web Treebank development set. This dataset also consists of data from the web, however, it contains much less words in need of normalization; MoNoise normalizes less than 0.5% of all words. We compared the performance using no normalization (ORIG) versus our INTEGRATED approach, which showed a very minor performance improvement from 81.42 to 81.43 LAS. This is a direct effect of the normalization model giving high probabilities to the original words on this more canonical data.

5.4 Analysis

To gain insights into which constructions are parsed better when using normalization, we compared the predictions of the vanilla parser with our NORM and INTEGRATED methods on the development data. Starting with NORM, the first observation is that the incoming arcs of the words which are normalized are responsible for 44.1% of all improvements, whereas the outgoing arcs are responsible for 17.6% of all improvements. So, the direct context of the normalized words is responsible for only 61.7% of all improvements. Considering the type of syntactic constructions for which parsing improved, it is hard to identify trends, because the improvements are based on the output of the normalization model, which normalizes a wide variety of words. One clearly influential effect of using normalization, was that the parser improved upon finding the root. When multiple unknown words occurred in the beginning of a sentence, the vanilla parser often failed at identifying the root, which improved considerably after normalizing.

For the INTEGRATED method, almost all the improvements made by NORM remained. On top of these, some additional improvements were made. Manual inspection revealed that these improvements often originated from a non-standard word, for which the correct normalization was ranked high. This then leads to improvements for the non-standard word as well as its context. In some cases, even incorrect normalization candidates lead to performance improvements. For example for ‘Gma’, where the normalization model ranked the original word first, but ‘mom’ second. Even though ‘grandma’ is the correct normalization, ‘mom’ occurs in similar contexts, and is much easier for the parser to process.

6 Conclusion

We showed that normalization can improve performance of a neural network parser, even when making use of character level information and external word embeddings. Integrating multiple normalization candidates into the parser leads to an even larger performance increase. Normalization has shown to be complementary to external embeddings, in contrast to character embeddings, which add no additional information. Our experiments revealed that our approach is robust, and it does not harm performance on more canonical data. However, when comparing our approach to the theoretical upperbound of using gold normalization, we saw that on different datasets the performance gain is of a different magnitude. Furthermore, we release a dataset containing 636 tweets annotated with both normalization and Universal Dependencies. The data and all code to reproduce the results in this paper is available at: https://bitbucket.org/robvanderg/normpar
Acknowledgements

We would like to thank Gosse Bouma for help with the data annotation, and Antonio Toral, Barbara Plank and the anonymous reviewers for feedback on the paper. This work is part of the ‘Parsening Algorithms for Uncertain Input’ project sponsored by the Nuance Foundation.

References


Alex Graves and Jürgen Schmidhuber. 2005. Frame-


