Dependency Parsing as Head Selection

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
The European Chapter of the Association for Computational Linguistics (EACL 2017)

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Dependency Parsing as Head Selection

Xingxing Zhang, Jianpeng Cheng and Mirella Lapata
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB
{x.zhang,jianpeng.cheng}@ed.ac.uk, mlap@inf.ed.ac.uk

Abstract

Conventional graph-based dependency parsers guarantee a tree structure both during training and inference. Instead, we formalize dependency parsing as the problem of independently selecting the head of each word in a sentence. Our model which we call DENSE (as shorthand for Dependency Neural Selection) produces a distribution over possible heads for each word using features obtained from a bidirectional recurrent neural network. Without enforcing structural constraints during training, DENSE generates (at inference time) trees for the overwhelming majority of sentences, while non-tree outputs can be adjusted with a maximum spanning tree algorithm. We evaluate DENSE on four languages (English, Chinese, Czech, and German) with varying degrees of non-projectivity. Despite the simplicity of the approach, our parsers are on par with the state of the art.\footnote{Our code is available at http://github.com/XingxingZhang/dense_parser.}

1 Introduction

Dependency parsing plays an important role in many natural language applications, such as relation extraction (Fundel et al., 2007), machine translation (Carreras and Collins, 2009), language modeling (Chelba et al., 1997; Zhang et al., 2016) and ontology construction (Snow et al., 2005). Dependency parsers represent syntactic information as a set of head-dependent relational arcs, typically constrained to form a tree. Practically all models proposed for dependency parsing in recent years can be described as graph-based (McDonald et al., 2005a) or transition-based (Yamada and Matsumoto, 2003; Nivre et al., 2006b).

Graph-based dependency parsers are typically arc-factored, where the score of a tree is defined as the sum of the scores of all its arcs. An arc is scored with a set of local features and a linear model, the parameters of which can be effectively learned with online algorithms (Crammer and Singer, 2001; Crammer and Singer, 2003; Freund and Schapire, 1999; Collins, 2002). In order to efficiently find the best scoring tree during training and decoding, various maximization algorithms have been developed (Eisner, 1996; Eisner, 2000; McDonald et al., 2005b). In general, graph-based methods are optimized globally, using features of single arcs in order to make the learning and inference tractable. Transition-based algorithms factorize a tree into a set of parsing actions. At each transition state, the parser scores a candidate action conditioned on the state of the transition system and the parsing history, and greedily selects the highest-scoring action to execute. This score is typically obtained with a classifier based on non-local features defined over a rich history of parsing decisions (Yamada and Matsumoto, 2003; Zhang and Nivre, 2011).

Regardless of the algorithm used, most well-known dependency parsers, such as the MST-Parser (McDonald et al., 2005b) and the MaltParser (Nivre et al., 2006a), rely on extensive feature engineering. Feature templates are typically manually designed and aim at capturing head-dependent relationships which are notoriously sparse and difficult to estimate. More recently, a few approaches (Chen and Manning, 2014; Lei et al., 2014a; Kiperwasser and Goldberg, 2016) apply neural networks for learning dense feature representations. The learned features are subsequently used in a conventional graph- or transition-based parser, or better de-
In this work, we propose a simple neural network-based model which learns to select the head for each word in a sentence without enforcing tree structured output. Our model which we call DE\textsc{Ne} (as shorthand for Dependency Neural Selection) employs bidirectional recurrent neural networks to learn feature representations for words in a sentence. These features are subsequently used to predict the head of each word. Although there is nothing inherent in the model to enforce tree-structured output, when tested on an English dataset, it is able to generate trees for 95\% of the sentences, 87\% of which are projective. The remaining non-tree (or non-projective) outputs are post-processed with the Chu-Liu-Edmond (or Eisner) algorithm. DE\textsc{Ne} uses the head selection procedure to estimate arc weights during training. During testing, it essentially reduces to a standard graph-based parser when it fails to produce tree (or projective) output.

We evaluate our model on benchmark dependency parsing corpora, representing four languages (English, Chinese, Czech, and German) with varying degrees of non-projectivity. Despite the simplicity of our approach, experiments show that the resulting parsers are on par with the state of the art.

\section{Related Work}

\textbf{Graph-based Parsing} Graph-based dependency parsers employ a model for scoring possible dependency graphs for a given sentence. The graphs are typically factored into their component arcs and the score of a tree is defined as the sum of its arcs. This factorization enables tractable search for the highest scoring graph structure which is commonly formulated as the search for the maximum spanning tree (MST). The Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005b) is often used to extract the MST in the case of non-projective trees, and the Eisner algorithm (Eisner, 1996; Eisner, 2000) in the case of projective trees. During training, weight parameters of the scoring function can be learned with margin-based algorithms (Crammer and Singer, 2001; Crammer and Singer, 2003) or the structured perceptron (Freund and Schapire, 1999; Collins, 2002). Beyond basic first-order models, the literature offers a few examples of higher-order models involving sibling and grand parent relations (Carreras, 2007; Koo et al., 2010; Zhang and McDonald, 2012). Although more expressive, such models render both training and inference more challenging.

\textbf{Transition-based Parsing} As the term implies, transition-based parsers conceptualize the process of transforming a sentence into a dependency tree as a sequence of transitions. A transition system typically includes a stack for storing partially processed tokens, a buffer containing the remaining input, and a set of arcs containing all dependencies between tokens that have been added so far (Nivre, 2003; Nivre et al., 2006b). A dependency tree is constructed by manipulating the stack and buffer, and appending arcs with predetermined operations. Most popular parsers employ an \textit{arc-standard} (Yamada and Matsumoto, 2003; Nivre, 2004) or \textit{arc-eager} transition system (Nivre, 2008). Extensions of the latter include the use of non-local training methods to avoid greedy error propagation (Zhang and Clark, 2008; Huang and Sagae, 2010; Zhang and Nivre, 2011; Goldberg and Nivre, 2012).

\textbf{Neural Network-based Features} Neural network representations have a long history in syntactic parsing (Mayberry and Miikkulainen, 1999; Henderson, 2004; Titov and Henderson, 2007). Recent work uses neural networks in lieu of the linear classifiers typically employed in conventional transition- or graph-based dependency parsers. For example, Chen and Manning (2014) use a feed forward neural network to learn features for a transition-based parser, whereas Lei et al. (2014a) do the same for a graph-based parser. Lei et al. (2014b) apply tensor decomposition to obtain word embeddings in their syntactic roles, which they subsequently use in a graph-based parser. Dyer et al. (2015) redesign components of a transition-based system where the buffer, stack, and action sequences are modeled separately with stack long short-term memory networks. The hidden states of these LSTMs are concatenated and used as features to a final transition classifier. Kiperwasser and Goldberg (2016) use bidirectional LSTMs to extract features for a transition- and graph-based parser, whereas Cross and Huang (2016) build a greedy arc-standard parser using similar features.

In our work, we formalize dependency parsing
as the task of finding for each word in a sentence its most probable head. Both head selection and the features it is based on are learned using neural networks. The idea of modeling child-parent relations independently dates back to Hall (2007) who use an edge-factored model to generate \( k \)-best parse trees which are subsequently reranked using a model based on rich global features. Later Smith (2010) show that a head selection variant of their loopy belief propagation parser performs worse than a model which incorporates tree structure constraints. Our parser is conceptually simpler: we rely on head selection to do most of the work and decode the best tree directly without using a reranker. In common with recent neural network-based dependency parsers, we aim to alleviate the need for hand-crafting feature combinations. Beyond feature learning, we further show that it is possible to simplify the training of a graph-based dependency parser in the context of bidirectional recurrent neural networks.

3 Dependency Parsing as Head Selection

In this section we present our parsing model, DENSE, which tries to predict the head of each word in a sentence. Specifically, the model takes as input a sentence of length \( N \) and outputs \( N \) (head, dependent) arcs. We describe the model focusing on unlabeled dependencies and then discuss how it can be straightforwardly extended to the labeled setting. We begin by explaining how words are represented in our model and then give details on how DENSE makes predictions based on these learned representations. Since there is no guarantee that the outputs of DENSE are trees (although they mostly are), we also discuss how to extend DENSE in order to enforce projective and non-projective tree outputs. Throughout this paper, lowercase boldface letters denote vectors (e.g., \( \mathbf{v} \) or \( \mathbf{v}_i \)), uppercase boldface letters denote matrices (e.g., \( \mathbf{M} \) or \( \mathbf{M}_i \)), and lowercase letters denote scalars (e.g., \( w \) or \( w_i \)).

3.1 Word Representation

Let \( S = (w_0, w_1, \ldots, w_N) \) denote a sentence of length \( N \); following common practice in the dependency parsing literature (Kübeler et al., 2009), we add an artificial ROOT token represented by \( w_0 \). Analogously, let \( A = (a_0, a_1, \ldots, a_N) \) denote the representation of sentence \( S \), with \( a_i \) representing word \( w_i \) \((0 \leq i \leq N)\). Besides encoding information about each \( w_i \) in isolation (e.g., its lexical meaning or POS tag), \( a_i \) must also encode \( w_i \)’s positional information within the sentence. Such information has been shown to be important in dependency parsing (McDonald et al., 2005a). For example, in the following sentence:

\[
\text{ROOT a dog is chasing a cat}
\]

the head of the first \( a \) is \textit{dog}, whereas the head of the second \( a \) is \textit{cat}. Without considering positional information, a model cannot easily decide which \( a \) (nearer or farther) to assign to \textit{dog}.

Long short-term memory networks (Hochreiter and Schmidhuber, 1997; LSTMs), a type of recurrent neural network with a more complex computational unit, have proven effective at capturing long-term dependencies. In our case LSTMs allow to represent each word on its own and within a sequence leveraging long-range contextual information. As shown in Figure 1, we first use a forward LSTM (LSTM\(^F\)) to read the sentence from left to right and then a backward LSTM (LSTM\(^B\)) to read the sentence from right to left, so that the entire sentence serves as context for each word:

\[
\begin{align*}
\mathbf{h}_i^F, \mathbf{c}_i^F &= \text{LSTM}^F(x_i, \mathbf{h}_{i-1}^F, \mathbf{c}_{i-1}^F) \\
\mathbf{h}_i^B, \mathbf{c}_i^B &= \text{LSTM}^B(x_i, \mathbf{h}_{i+1}^B, \mathbf{c}_{i+1}^B)
\end{align*}
\]

where \( x_i \) is the feature vector of word \( w_i \), \( \mathbf{h}_i^F \in \mathbb{R}^d \) and \( \mathbf{c}_i^F \in \mathbb{R}^d \) are the hidden states and memory cells for the \( i \)th word \( w_i \) in LSTM\(^F\) and \( d \) is

---

2For more detail on LSTM networks, see e.g., Graves (2012) or Goldberg (2016).
the hidden unit size. $h^F_i$ is also the representation for $w_0$, $(w_1$ and its left neighboring words) and $c^F_i$ is an internal state maintained by LSTM$^F$. $h^B_i \in \mathbb{R}^d$ and $c^B_i \in \mathbb{R}^d$ are the hidden states and memory cells for the backward LSTM$^B$. Each token $w_i$ is represented by $x_i$, the concatenation of two vectors corresponding to $w_i$’s lexical and POS tag embeddings:

$$x_i = [W_e \cdot e(w_i); W_t \cdot e(t_i)] \quad (3)$$

where $e(w_i)$ and $e(t_i)$ are one-hot vector representations of token $w_i$ and its POS tag $t_i$; $W_e \in \mathbb{R}^{s \times |V|}$ and $W_t \in \mathbb{R}^{q \times |T|}$ are the word and POS tag embedding matrices, where $|V|$ is the vocabulary size, $s$ is the word embedding size, $|T|$ is the POS tag set size, and $q$ the tag embedding size. The hidden states of the forward and backward LSTMs are concatenated to obtain $a_i$, the final representation of $w_i$:

$$a_i = [h^F_i; h^B_i] \quad i \in [0, N] \quad (4)$$

Note that bidirectional LSTMs are one of many possible ways of representing word $w_i$. Alternative representations include embeddings obtained from feed-forward neural networks (Chen and Manning, 2014; Lei et al., 2014a), character-based embeddings (Ballesteros et al., 2015), and more conventional features such as those introduced in McDonald et al. (2005a).

### 3.2 Head Selection

We now move on to discuss our formalization of dependency parsing as head selection. We begin with unlabeled dependencies and then explain how the model can be extended to predict labeled ones.

In a dependency tree, a head can have multiple dependents, whereas a dependent can have only one head. Based on this fact, dependency parsing can be formalized as follows. Given a sentence $S = (w_0, w_1, \ldots, w_N)$, we aim to find for each word $w_i \in \{w_1, w_2, \ldots, w_N\}$ the most probable head $w_j \in \{w_0, w_1, \ldots, w_N\}$. For example, in Figure 1, to find the head for the token love, we calculate probabilities $P_{head}(\text{ROOT}|\text{love}, S)$, $P_{head}(\text{kids}|\text{love}, S)$, and $P_{head}(\text{candy}|\text{love}, S)$, and select the highest. More formally, we estimate the probability of token $w_j$ being the head of token $w_i$ in sentence $S$ as:

$$P_{head}(w_j|w_i, S) = \frac{\exp(g(a_j, a_i))}{\sum_{k=0}^{N} \exp(g(a_k, a_i))} \quad (5)$$

where $a_i$ and $a_j$ are vector-based representations of $w_i$ and $w_j$, respectively (described in Section 3.1); $g(a_j, a_i)$ is a neural network with a single hidden layer that computes the associative score between representations $a_i$ and $a_j$:

$$g(a_j, a_i) = v_a^T \cdot \tanh(U_a \cdot a_j + W_a \cdot a_i) \quad (6)$$

where $v_a \in \mathbb{R}^{2d}$, $U_a \in \mathbb{R}^{2d \times 2d}$, and $W_a \in \mathbb{R}^{2d \times 2d}$ are weight matrices of $g$. Note that the candidate head $w_j$ can be the ROOT, while the dependent $w_i$ cannot. Equations (5) and (6) compute the probability of adding an arc between two words, in a fashion similar to the neural attention mechanism in sequence-to-sequence models (Bahdanau et al., 2015).

We train our model by minimizing the negative log likelihood of the gold standard (head, dependent) arcs in all training sentences:

$$J(\theta) = -\frac{1}{|T|} \sum_{S \in T} \sum_{i=1}^{N_S} \log P_{head}(h(w_i)|w_i, S) \quad (7)$$

where $T$ is the training set, $h(w_i)$ is $w_i$’s gold standard head within sentence $S$, and $N_S$ the number of words in $S$ (excluding ROOT). During inference, for each word $w_i$ ($i \in [1, N_S]$) in $S$, we greedily choose the most likely head $w_j$ ($j \in [0, N_S]$):

$$w_j = \arg \max_{w_{j \in [0, N_S]}} P_{head}(w_j|w_i, S) \quad (8)$$

Note that the prediction for each word $w_i$ is made independently of the other words in the sentence.

Given our greedy inference method, there is no guarantee that predicted (head, dependent) arcs form a tree (maybe there are cycles). However, we empirically observed that most outputs during inference are indeed trees. For instance, on an English dataset, 95% of the arcs predicted on the development set are trees, and 87% of them are projective, whereas on a Chinese dataset, 87% of the arcs form trees, 73% of which are projective. This indicates that although the model does not explicitly model tree structure during training, it is able to figure out from the data (which consists of trees) that it should predict them.

So far we have focused on unlabeled dependencies, however it is relatively straightforward to extend DENSE to produce labeled dependencies. We basically train an additional classifier.
to predict labels for the arcs which have been already identified. The classifier takes as input features \([a_i; a_j; x_i; x_j]\) representing properties of the arc \(\langle w_j, w_i \rangle\). These consist of \(a_i\) and \(a_j\), the LSTM-based representations for \(w_i\) and \(w_j\) (see Equation (4)), and their word and part-of-speech embeddings, \(x_i\) and \(x_j\) (see Equation (3)). Specifically, we use a trained DeNSE model to go through the training corpus and generate features and corresponding dependency labels as training data. We employ a two-layer rectifier network (Glorot et al., 2011) for the classification task.

3.3 Maximum Spanning Tree Algorithms

As mentioned earlier, greedy inference may not produce well-formed trees. In this case, the output of DeNSE can be adjusted with a maximum spanning tree algorithm. We use the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) for building non-projective trees and the Eisner (1996) algorithm for projective ones.

Following McDonald et al. (2005b), we view a sentence \(S = (w_0 = \text{ROOT}, w_1, \ldots, w_N)\) as a graph \(G_S = (V_S, E_S)\) with the sentence words and the dummy root symbol as vertices and a directed edge between every pair of distinct words and from the root symbol to every word. The directed graph \(G_S\) is defined as:

\[
\begin{align*}
V_S &= \{w_0 = \text{ROOT}, w_1, \ldots, w_N\} \\
E_S &= \{(i, j) : i \neq j, (i, j) \in [0, N] \times [1, N]\} \\
s(i, j) &= P_{\text{head}}(w_i | w_j, S) \quad (i, j) \in E_S
\end{align*}
\]

where \(s(i, j)\) is the weight of edge \(\langle i, j \rangle\) and \(P_{\text{head}}(w_i | w_j, S)\) is known. The problem of dependency parsing now boils down to finding the tree with the highest score which is equivalent to finding a MST in \(G_S\) (McDonald et al., 2005b).

Non-projective Parsing

To build a non-projective parser, we solve the MST problem with the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). The algorithm selects for each vertex (excluding \(\text{ROOT}\)) the in-coming edge with the highest weight. If a tree results, it must be the maximum spanning tree and the algorithm terminates. Otherwise, there must be a cycle which the algorithm identifies, contracts into a single vertex and recalculates edge weights going into and out of the cycle. The greedy inference strategy described in Equation (8)) is essentially a sub-procedure in the Chu-Liu-Edmonds algorithm with the algorithm terminating after the first iteration. In implementation, we only run the Chu-Liu-Edmonds algorithm through graphs with cycles, i.e., non-tree outputs.

Projective Parsing

For projective parsing, we solve the MST problem with the Eisner (1996) algorithm. The time complexity of the Eisner algorithm is \(O(N^3)\), while checking if a tree is projective can be done reasonably faster, with a \(O(N \log N)\) algorithm. Therefore, we only apply the Eisner algorithm to the non-projective output of our greedy inference strategy. Finally, it should be noted that the training of our model does not rely on the Chu-Liu-Edmonds or Eisner algorithm, or any other graph-based algorithm. MST algorithms are only used at test time to correct non-tree outputs which are a minority; DeNSE acquires underlying tree structure constraints from the data without an explicit learning algorithm.

4 Experiments

We evaluated our parser in a projective and non-projective setting. In the following, we describe the datasets we used and provide training details for our models. We also present comparisons against multiple previous systems and analyze the parser’s output.

4.1 Datasets

In the projective setting, we assessed the performance of our parser on the English Penn Treebank (PTB) and the Chinese Treebank 5.1 (CTB). Our experimental setup closely follows Chen and Manning (2014) and Dyer et al. (2015).

For English, we adopted the Stanford basic dependencies (SD) representation (De Marneffe et al., 2006).\(^4\) We follow the standard splits of PTB, sections 2–21 were used for training, section 22 for development, and section 23 for testing. POS tags were assigned using the Stanford tagger (Toutanova et al., 2003) with an accuracy of 97.3%. For Chinese, we follow the same split of CTB5 introduced in Zhang and Clark (2008). In particular, we used sections 001–815, 1001–1136 for training, sections 886–931, 1148–1151 for development, and sections 816–885, 1137–1147 for testing. The original constituency trees in CTB were converted to dependency trees with the

\(^4\)We obtained SD representations using the Stanford parser v3.3.0.
<table>
<thead>
<tr>
<th>Dataset</th>
<th># Sentences</th>
<th>(%) Projective</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>39,832</td>
<td>99.9</td>
</tr>
<tr>
<td>Chinese</td>
<td>16,091</td>
<td>100.0</td>
</tr>
<tr>
<td>Czech</td>
<td>72,319</td>
<td>76.9</td>
</tr>
<tr>
<td>German</td>
<td>38,845</td>
<td>72.2</td>
</tr>
</tbody>
</table>

Table 1: Projective statistics on four datasets. Number of sentences and percentage of projective trees are calculated on the training set.

5http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html

We used gold segmentation and gold POS tags as in Chen and Manning (2014) and Dyer et al. (2015).

In the non-projective setting, we assessed the performance of our parser on Czech and German, the largest non-projective datasets released as part of the CoNLL 2006 multilingual dependency parsing shared task. Since there is no official development set in either dataset, we used the last 374/367 sentences in the Czech/German training set as development data. Projective statistics of the four datasets are summarized in Table 1.

4.2 Training Details

We trained our models on an Nvidia GPU card; training takes one to two hours. Model parameters were uniformly initialized to $[-0.1, 0.1]$. We used Adam (Kingma and Ba, 2014) to optimize our models with hyper-parameters recommended by the authors (i.e., learning rate 0.001, first momentum coefficient 0.9, and second momentum coefficient 0.999). To alleviate the gradient exploding problem, we rescaled the gradient when its norm exceeded 5 (Pascanu et al., 2013). Dropout (Srivastava et al., 2014) was applied to our model with the strategy recommended in the literature (Zaremba et al., 2014; Semeniuta et al., 2016). On all datasets, we used two-layer LSTMs and set $d = s = 300$, where $d$ is the hidden unit size and $s$ is the word embedding size.

As in previous neural dependency parsing work (Chen and Manning, 2014; Dyer et al., 2015), we used pre-trained word vectors to initialize our word embedding matrix $W_e$. For the PTB experiments, we used 300 dimensional pre-trained GloVe vectors (Pennington et al., 2014). For the CTB experiments, we trained 300 dimensional GloVe vectors on the Chinese Gigaword corpus which we segmented with the Stanford Chinese Segmenter (Tseng et al., 2005). For Czech and German, we did not use pre-trained word vectors. The POS tag embedding size was set to $q = 30$ in the English experiments, $q = 50$ in the Chinese experiments and $q = 40$ in both Czech and German experiments.

4.3 Results

For both English and Chinese experiments, we report unlabeled (UAS) and labeled attachment scores (LAS) on the development and test sets; following Chen and Manning (2014) punctuation is excluded from the evaluation.

Experimental results on PTB are shown in Table 2. We compared our model with several recent papers following the same evaluation protocol and experimental settings. The first block in the table contains mostly graph-based parsers which do not use neural networks: Bohnet10 (Bohnet, 2010), Martins13 (Martins et al., 2013), and Z&M14 (Zhang and McDonald, 2014). Z&N11 (Zhang and Nivre, 2011) is a transition-based parser with non-local features. Accuracy results for all four parsers are reported in Weiss et al. (2015).

The second block in Table 2 presents results obtained from neural network-based parsers. C&M14 (Chen and Manning, 2014) is a transition-based parser using features learned with a feed
## Table 3: Results on Chinese dataset (CTB).

<table>
<thead>
<tr>
<th>Parser</th>
<th>Dev UAS</th>
<th>Test UAS</th>
<th>Dev LAS</th>
<th>Test LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z&amp;N11</td>
<td>—</td>
<td>—</td>
<td>86.00</td>
<td>84.40</td>
</tr>
<tr>
<td>Z&amp;M14</td>
<td>—</td>
<td>—</td>
<td>87.96</td>
<td>86.34</td>
</tr>
<tr>
<td>C&amp;M14</td>
<td>84.00</td>
<td>82.40</td>
<td>83.90</td>
<td>82.40</td>
</tr>
<tr>
<td>Dyer15</td>
<td>87.20</td>
<td>85.30</td>
<td>87.20</td>
<td>85.70</td>
</tr>
<tr>
<td>K&amp;G16 graph</td>
<td>—</td>
<td>—</td>
<td>86.60</td>
<td>85.10</td>
</tr>
<tr>
<td>K&amp;G16 trans</td>
<td>—</td>
<td>—</td>
<td>87.60</td>
<td>86.10</td>
</tr>
<tr>
<td>DeNSE-Pei</td>
<td>82.50</td>
<td>80.74</td>
<td>82.38</td>
<td>80.55</td>
</tr>
<tr>
<td>DeNSE-Pei+E</td>
<td>83.40</td>
<td>81.63</td>
<td>83.30</td>
<td>81.65</td>
</tr>
<tr>
<td>DeNSE</td>
<td>87.27</td>
<td>85.73</td>
<td>87.63</td>
<td>85.94</td>
</tr>
<tr>
<td>DeNSE+E</td>
<td><strong>87.35</strong></td>
<td>85.85</td>
<td>87.84</td>
<td>86.15</td>
</tr>
</tbody>
</table>

+E: we post-process non-projective outputs with the Eisner algorithm.

Table 3: Results on Chinese dataset (CTB).

## Table 4: UEM results on PTB and CTB.

<table>
<thead>
<tr>
<th>Parser</th>
<th>PTB Dev</th>
<th>PTB Test</th>
<th>CTB Dev</th>
<th>CTB Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>C&amp;M14</td>
<td>43.35</td>
<td>40.93</td>
<td>32.75</td>
<td>32.20</td>
</tr>
<tr>
<td>Dyer15</td>
<td>51.94</td>
<td>50.70</td>
<td><strong>39.72</strong></td>
<td><strong>37.23</strong></td>
</tr>
<tr>
<td>DeNSE</td>
<td>51.24</td>
<td>49.34</td>
<td>34.74</td>
<td>33.66</td>
</tr>
<tr>
<td>DeNSE+E</td>
<td><strong>52.47</strong></td>
<td><strong>50.79</strong></td>
<td>36.49</td>
<td>35.13</td>
</tr>
</tbody>
</table>

Table 4: UEM results on PTB and CTB.

## Table 5: Non-projective results on the CoNLL 2006 dataset. +CLE: we post-process non-tree outputs with the Chu-Liu-Edmonds algorithm.

<table>
<thead>
<tr>
<th>Parser</th>
<th>Czech UAS</th>
<th>Czech LAS</th>
<th>German UAS</th>
<th>German LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MST-1st</td>
<td>86.18</td>
<td>—</td>
<td>89.54</td>
<td>—</td>
</tr>
<tr>
<td>MST-2nd</td>
<td>87.30</td>
<td>—</td>
<td>90.14</td>
<td>—</td>
</tr>
<tr>
<td>Turbo-1st</td>
<td>87.66</td>
<td>—</td>
<td>90.52</td>
<td>—</td>
</tr>
<tr>
<td>Turbo-3rd</td>
<td>90.32</td>
<td>—</td>
<td><strong>92.41</strong></td>
<td>—</td>
</tr>
<tr>
<td>RBG-1st</td>
<td>87.90</td>
<td>—</td>
<td>90.24</td>
<td>—</td>
</tr>
<tr>
<td>RBG-3rd</td>
<td><strong>90.50</strong></td>
<td>—</td>
<td>91.97</td>
<td>—</td>
</tr>
<tr>
<td>DeNSE-Pei</td>
<td>86.00</td>
<td>77.92</td>
<td>89.42</td>
<td>86.48</td>
</tr>
<tr>
<td>DeNSE-Pei+CLE</td>
<td>86.52</td>
<td>78.42</td>
<td>89.52</td>
<td>86.58</td>
</tr>
<tr>
<td>DeNSE</td>
<td>89.60</td>
<td>81.70</td>
<td>92.15</td>
<td>89.58</td>
</tr>
<tr>
<td>DeNSE+CLE</td>
<td>89.68</td>
<td>81.72</td>
<td>92.19</td>
<td>89.60</td>
</tr>
</tbody>
</table>

Table 5: Non-projective results on the CoNLL 2006 dataset. +CLE: we post-process non-tree outputs with the Chu-Liu-Edmonds algorithm.

forward neural network. Although very fast, its performance is inferior compared to graph-based parsers or strong non-neural transition based parsers (e.g., Z&N11). Dyer15 (Dyer et al., 2015) uses (stack) LSTMs to model the states of the buffer, the stack, and the action sequence of a transition system. Weiss15 (Weiss et al., 2015) is another transition-based parser, with a more elaborate training procedure. Features are learned with a neural network model similar to C&M14, but much larger with two layers. The hidden states of the neural network are then used to train a structured perceptron for better beam search decoding. Andor16 (Andor et al., 2016) is similar to Weiss15, but uses a globally normalized training algorithm instead.

Unlike all models above, DeNSE does not use any kind of transition- or graph-based algorithm during training and inference. Nonetheless, it obtains a UAS of 94.02%. Around 95% of the model’s outputs after inference are trees, 87% of which are projective. When we post-process the remaining 13% of non-projective outputs with the Eisner algorithm (DeNSE+E), we obtain a slight improvement on UAS (94.10%).

Kiperwasser and Goldberg (2016) extract features from bidirectional LSTMs and feed them to a graph- (K&G16 graph) and transition-based parser (K&G16 trans). Their LSTMs are jointly trained with the parser objective. DeNSE yields very similar performance to their transition-based parser while it outperforms K&G16 graph. A key difference between DeNSE and K&G16 lies in the training objective. The objective of DeNSE is log-likelihood based without tree structure constraints (the model is trained to produce a distribution over possible heads for each word, where each head selection is independent), while K&G16 employ a max-margin objective with tree structure constraints. Although our probabilistic objective is non-structured, it is perhaps easier to train compared to a margin-based one.

We also assessed the importance of the bidirectional LSTM on its own by replacing our LSTM-based features with those obtained from a feed-forward network. Specifically, we used the 1-order-atomic features introduced in Lei et al. (2014a) which represent POS-tags, modifiers, heads, and their relative positions. As can be seen in Table 2 (DeNSE-Pei), these features are less effective compared to LSTM-based ones and the contribution of the MST algorithm (Eisner) during decoding is more pronounced (DeNSE-Pei+E). We observe similar trends in the Chinese, German, and Czech datasets (see Tables 3 and 5).

Results on CTB follow a similar pattern. As shown in Table 3, DeNSE outperforms all previous neural models (see the test set columns) on UAS and LAS. DeNSE performs competitively with Z&M14, a non-neural model with a com-
plex high order decoding algorithm involving cube pruning and strategies for encouraging diversity. Post-processing the output of the parser with the Eisner algorithm generally improves performance (by 0.21%; see last row in Table 3). Again we observe that 1-order-atomic features (Lei et al., 2014a) are inferior compared to the LSTM. Table 4 reports unlabeled sentence level exact match (UEM) in Table 4 for English and Chinese. Interestingly, even when using the greedy inference strategy, DeNSE yields a UEM comparable to Dyer15 on PTB. Finally, in Figure 2 we analyze the performance of our parser on sentences of different length. On both PTB and CTB, DeNSE has an advantage on long sentences compared to C&M14 and Dyer15.

For Czech and German, we closely follow the evaluation setup of CoNLL 2006. We report both UAS and LAS, although most previous work has focused on UAS. Our results are summarized in Table 5. We compare DeNSE against three non-projective graph-based dependency parsers: the MST parser (McDonald et al., 2005b), the Turbo parser (Martins et al., 2013), and the RBG parser (Lei et al., 2014b). We show the performance of these parsers in the first order setting (e.g., MST-1st) and in higher order settings (e.g., Turbo-3rd). The results of MST-1st, MST-2nd, RBG-1st and RBG-3rd are reported in Lei et al. (2014b) and the results of Turbo-1st and Turbo-3rd are reported in Martins et al. (2013). We show results for our parser with greedy inference (see DeNSE in the table) and when we use the Chu-Liu-Edmonds algorithm to post-process non-tree outputs (DeNSE+CLE).

As can been seen, DeNSE outperforms all other first (and second) order parsers on both German and Czech. As in the projective experiments, we observe slight a improvement (on both UAS and LAS) when using a MST algorithm. On German, DeNSE is comparable with the best third-order parser (Turbo-3rd), while on Czech it lags behind Turbo-3rd and RBG-3rd. This is not surprising considering that DeNSE is a first-order parser and only uses words and POS tags as features. Comparison systems use a plethora of hand-crafted features and more sophisticated high-order decoding algorithms. Finally, note that a version of DeNSE with features in (Lei et al., 2014a) is consistently worse (see the second block in Table 5).

Our experimental results demonstrate that using a MST algorithm during inference can slightly improve the model’s performance. We further ex-
amined the extent to which the MST algorithm is necessary for producing dependency trees. Table 6 shows the percentage of trees before and after the application of the MST algorithm across the four languages. In the majority of cases DEENSE outputs trees (ranging from 87.0% to 96.7%) and a significant proportion of them are projective (ranging from 65.5% to 86.6%). Therefore, only a small proportion of outputs (14.0% on average) need to be post-processed with the Eisner or Chu-Liu-Edmonds algorithm.

5 Conclusions

In this work we presented DEENSE, a neural dependency parser which we train without a transition system or graph-based algorithm. Experimental results show that DEENSE achieves competitive performance across four different languages and can seamlessly transfer from a projective to a non-projective parser simply by changing the post-processing MST algorithm during inference. In the future, we plan to increase the coverage of our parser by using tri-training techniques (Li et al., 2014) and multi-task learning (Luong et al., 2015).

Acknowledgments We would like to thank Adam Lopez and Frank Keller for their valuable feedback. We acknowledge the financial support of the European Research Council (ERC; award number 681760).

References


Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 334–343, Beijing, China.


