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 **De**pendency **N**eural **Se**lection) 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.[1]

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 MaltPaser (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-

[1] Our code is available at http://github.com/XingxingZhang/dense_parser.

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 665–676,
Valencia, Spain, April 3-7, 2017.

signed variants (Dyer et al., 2015).

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 DENSE (as shorthand for **De**pendency **N**eural **Se**lection) 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. DENSE 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.

2 Related Work

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.

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 *arc-standard* (Yamada and Matsumoto, 2003; Nivre, 2004) or *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).

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

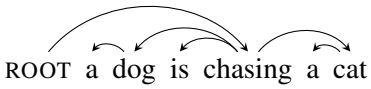
as the task of finding for each word in a sentence its most probable head. Both head selection and the features it is based on are learned using neural networks. The idea of modeling child-parent relations independently dates back to Hall (2007) who use an edge-factored model to generate k-best parse 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 $\mathbf{M}_b$), 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 (Kübler et al., 2009), we add an artificial ROOT token represented by w_0. Analogously, let $A = (\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_N)$ denote the representation of sentence S, with $\mathbf{a}_i$ 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), $\mathbf{a}_i$ 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:



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

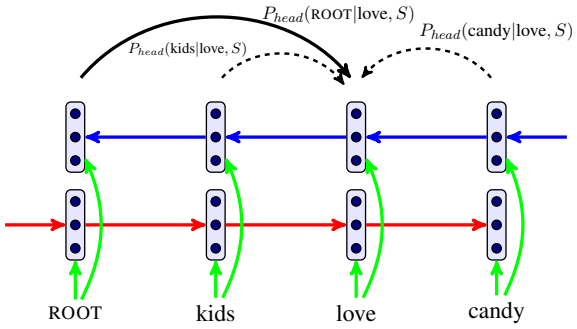


Figure 1: DENSE estimates the probability a word being the head of another word based on bidirectional LSTM representations for the two words. P_{head}(ROOT|love, S) is the probability of ROOT being the head of *love* (dotted arcs denote candidate heads; the solid arc is the goldstandard).

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:[2]

$$\mathbf{h}_i^F, \mathbf{c}_i^F = \text{LSTM}^F(\mathbf{x}_i, \mathbf{h}_{i-1}^F, \mathbf{c}_{i-1}^F) \quad (1)$$

$$\mathbf{h}_i^B, \mathbf{c}_i^B = \text{LSTM}^B(\mathbf{x}_i, \mathbf{h}_{i+1}^B, \mathbf{c}_{i+1}^B) \quad (2)$$

where $\mathbf{x}_i$ 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 ith word w_i in LSTM^F and d is

[2]For more detail on LSTM networks, see e.g., Graves (2012) or Goldberg (2016).

the hidden unit size. $\mathbf{h}_i^F$ is also the representation for $w_{0:i}$ (w_i and its left neighboring words) and $\mathbf{c}_i^F$ is an internal state maintained by LSTM^F. $\mathbf{h}_i^B \in \mathbb{R}^d$ and $\mathbf{c}_i^B \in \mathbb{R}^d$ are the hidden states and memory cells for the backward LSTM^B. Each token w_i is represented by $\mathbf{x}_i$, the concatenation of two vectors corresponding to w_i's lexical and POS tag embeddings:

$$\mathbf{x}_i = [\mathbf{W}_e \cdot e(w_i); \mathbf{W}_t \cdot e(t_i)] \quad (3)$$

where $e(w_i)$ and $e(t_i)$ are one-hot vector representations of token w_i and its POS tag t_i; $\mathbf{W}_e \in \mathbb{R}^{s\times|V|}$ and $\mathbf{W}_t \in \mathbb{R}^{q\times|T|}$ 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 $\mathbf{a}_i$, the final representation of w_i:

$$\mathbf{a}_i = [\mathbf{h}_i^F; \mathbf{h}_i^B] \quad i \in [0, N] \quad (4)$$

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:

$$P_{head}(w_j|w_i, S) = \frac{\exp(g(\mathbf{a}_j, \mathbf{a}_i))}{\sum_{k=0}^{N} \exp(g(\mathbf{a}_k, \mathbf{a}_i))} \quad (5)$$

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

$$g(\mathbf{a}_j, \mathbf{a}_i) = \mathbf{v}_a^\top \cdot \tanh(\mathbf{U}_a \cdot \mathbf{a}_j + \mathbf{W}_a \cdot \mathbf{a}_i) \quad (6)$$

where $\mathbf{v}_a \in \mathbb{R}^{2d}$, $\mathbf{U}_a \in \mathbb{R}^{2d\times 2d}$, and $\mathbf{W}_a \in \mathbb{R}^{2d\times 2d}$ 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:

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

where $\mathcal{T}$ is the training set, $h(w_i)$ is w_i's gold standard head[3] 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 ($j \in [0, N_S]$):

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

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

[3] Note that $h(w_i)$ can be ROOT.

to predict labels for the arcs which have been already identified. The classifier takes as input features $[\mathbf{a}_i; \mathbf{a}_j; \mathbf{x}_i; \mathbf{x}_j]$ representing properties of the arc $\langle w_j, w_i \rangle$. These consist of $\mathbf{a}_i$ and $\mathbf{a}_j$, the LSTM-based representations for w_i and w_j (see Equation (4)), and their word and part-of-speech embeddings, $\mathbf{x}_i$ and $\mathbf{x}_j$ (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 $G_S = \langle V_S, E_S \rangle$ 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:

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

where $s(i, j)$ is the weight of edge $\langle i, j \rangle$ and $P_{head}(w_i | w_j, S)$ 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 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 v.3.3.0.

Dataset	# Sentences	(%) Projective
English	39,832	99.9
Chinese	16,091	100.0
Czech	72,319	76.9
German	38,845	72.2

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

Penn2Malt tool.[5] 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.[6] 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 $\mathbf{W}_e$. For the PTB experiments, we used 300 dimensional pre-trained GloVe[7] 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.

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

[6] We make the number of sentences in the development and test sets comparable.

[7] http://nlp.stanford.edu/projects/glove/

Parser	Dev UAS	Dev LAS	Test UAS	Test LAS
Bohnet10	—	—	92.88	90.71
Martins13	—	—	92.89	90.55
Z&M14	—	—	93.22	91.02
Z&N11	—	—	93.00	90.95
C&M14	92.00	89.70	91.80	89.60
Dyer15	93.20	90.90	93.10	90.90
Weiss15	—	—	93.99	92.05
Andor16	—	—	**94.61**	**92.79**
K&G16 *graph*	—	—	93.10	91.00
K&G16 *trans*	—	—	93.90	91.90
DENSE-Pei	90.77	88.35	90.39	88.05
DENSE-Pei+E	91.39	88.94	91.00	88.61
DENSE	94.17	91.82	94.02	91.84
DENSE+E	**94.30**	**91.95**	94.10	91.90

Table 2: Results on English dataset (PTB with Stanford Dependencies). +E: we post-process non-projective output with the Eisner algorithm.

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

Parser	Dev UAS	Dev LAS	Test UAS	Test LAS
Z&N11	—	—	86.00	84.40
Z&M14	—	—	**87.96**	**86.34**
C&M14	84.00	82.40	83.90	82.40
Dyer15	87.20	**85.90**	87.20	85.70
K&G16 *graph*	—	—	86.60	85.10
K&G16 *trans*	—	—	87.60	86.10
DENSE-Pei	82.50	80.74	82.38	80.55
DENSE-Pei+E	83.40	81.63	83.46	81.65
DENSE	87.27	85.73	87.63	85.94
DENSE+E	**87.35**	85.85	87.84	86.15

Table 3: Results on Chinese dataset (CTB). +E: we post-process non-projective outputs with the Eisner algorithm.

Parser	PTB Dev	PTB Test	CTB Dev	CTB Test
C&M14	43.35	40.93	32.75	32.20
Dyer15	51.94	50.70	**39.72**	**37.23**
DENSE	51.24	49.34	34.74	33.66
DENSE+E	**52.47**	**50.79**	36.49	35.13

Table 4: UEM results on PTB and CTB.

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.

Parser	Czech UAS	Czech LAS	German UAS	German LAS
MST-1st	86.18	—	89.54	—
MST-2nd	87.30	—	90.14	—
Turbo-1st	87.66	—	90.52	—
Turbo-3rd	90.32	—	**92.41**	—
RBG-1st	87.90	—	90.24	—
RBG-3rd	**90.50**	—	91.97	—
DENSE-Pei	86.00	77.92	89.42	86.48
DENSE-Pei+CLE	86.52	78.42	89.52	86.58
DENSE	89.60	81.70	92.15	89.58
DENSE+CLE	89.68	81.72	92.19	89.60

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

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-

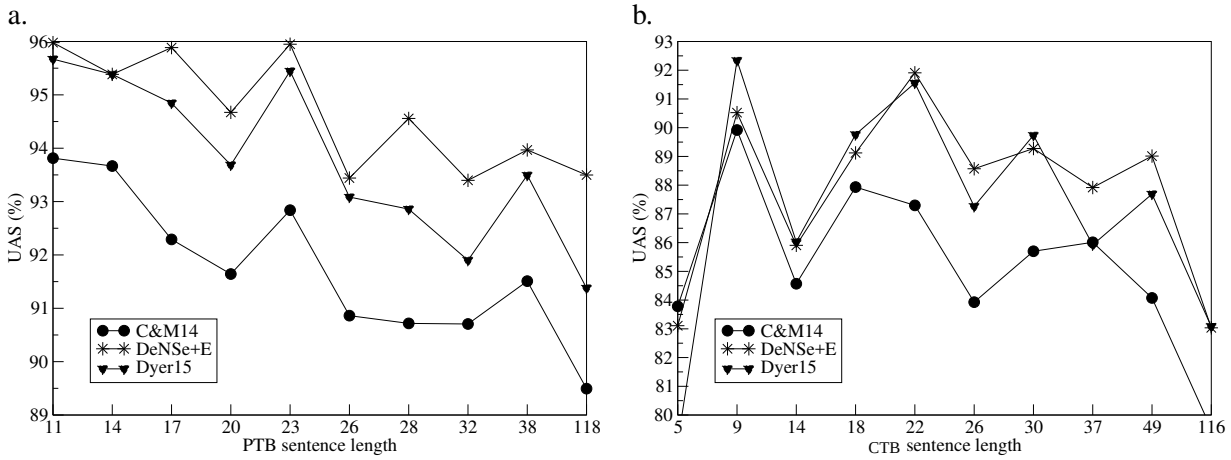


Figure 2: UAS against sentence length on PTB and CTB (development set). Sentences are sorted by length in ascending order and divided equally into 10 bins. The horizontal axis is the length of the last sentence in each bin.

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).

Dataset	#Sent	Before MST Tree	Before MST Proj	After MST Tree	After MST Proj
PTB	1,700	95.1	86.6	100.0	100.0
CTB	803	87.0	73.1	100.0	100.0
Czech	374	87.7	65.5	100.0	72.7
German	367	96.7	67.3	100.0	68.1

Table 6: Percentage of trees and projective trees on the development set before and after DENSE uses a MST algorithm. On PTB and CTB, we use the Eisner algorithm and on Czech and German, we use the Chu-Liu-Edmonds algorithm.

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 DENSE 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 DENSE, a neural dependency parser which we train without a transition system or graph-based algorithm. Experimental results show that DENSE 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

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learing Representations*, San Diego, California.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. 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*, pages 349–359, Lisbon, Portugal.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China.

Xavier Carreras and Michael Collins. 2009. Non-projective parsing for statistical machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 200–209, Singapore.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic.

Ciprian Chelba, David Engle, Frederick Jelinek, Victor Jimenez, Sanjeev Khudanpur, Lidia Mangu, Harry Printz, Eric Ristad, Ronald Rosenfeld, Andreas Stolcke, and Dekai Wu. 1997. Structure and performance of a dependency language model. In *Fifth European Conference on Speech Communication and Technology, EUROSPEECH 1997*, Rhodes, Greece.

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

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8.

Koby Crammer and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

James Cross and Liang Huang. 2016. Incremental parsing with minimal features using bi-directional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, Genoa, Italy.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual*

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.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

Jason Eisner. 1996. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86, Santa Cruz, California, USA.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer.

Yoav Freund and Robert E Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Katrin Fundel, Robert Küffner, and Ralf Zimmer. 2007. Relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, Cadiz, Spain.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India.

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Alex Graves. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer.

Keith Hall. 2007. K-best spanning tree parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 392–399, Prague, Czech Republic.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 95–102, Barcelona, Spain.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA.

Sandra Kübler, Ryan McDonald, Joakim Nivre, and Graeme Hirst. 2009. *Dependency Parsing*. Morgan and Claypool Publishers.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014a. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1381–1391, Baltimore, Maryland.

Tao Lei, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014b. Low-rank tensors for scoring dependency structures. In *Proceedings of the ACL*. Baltimore, Maryland.

Zhenghua Li, Min Zhang, and Wenliang Chen. 2014. Ambiguity-aware ensemble training for semi-supervised dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 457–467, Baltimore, Maryland.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. In *Proceedings of the 4th International Conference on Learning Representations*, San Juan, Puerto Rico.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria.

Marshall R. Mayberry and Risto Miikkulainen. 1999. SardSrn: A neural network shift-reduce parser. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 820–825, Stockholm, Sweden.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 91–98, Ann Arbor, Michigan.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural*

Language Processing, pages 523–530, Vancouver, British Columbia, Canada.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006a. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, Genoa, Italy.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006b. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1310–1318, Atlanta, Georgia.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent dropout without memory loss. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1757–1766, Osaka, Japan.

David Arthur Smith. 2010. *Efficient inference for trees and alignments: modeling monolingual and bilingual syntax with hard and soft constraints and latent variables*. Johns Hopkins University.

Rion Snow, Daniel Jurafsky, and Andrew Y Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Advances in Neural Information Processing Systems 17*, pages 1297–1304, Vancouver, British Columbia.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ivan Titov and James Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 173–180, Edmonton, Canada.

Huihsin Tseng, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. 2005. A conditional random field word segmenter for Sighan bakeoff 2005. In *Proceedings of the 4th SIGHAN workshop on Chinese language Processing*, pages 168–171, Jeju Island, Korea.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th Workshop on Parsing Technologies*, pages 195–206, Nancy, France.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 656–661, Baltimore, Maryland.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2016. Top-down tree long short-term memory networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 310–320, San Diego, California.