Integrating Graph-Based and Transition-Based Dependency Parsers in the Deep Contextualized Era

Agnieszka Falenska[1] and **Anders Björkelund**[2] and **Jonas Kuhn**[1]
[1] University of Stuttgart, Institute for Natural Language Processing
[2] Lund University, Department of Astronomy and Theoretical Physics
{falenska,jonas}@ims.uni-stuttgart.de
anders.bjorkelund@thep.lu.se

Abstract

Graph-based and transition-based dependency parsers used to have different strengths and weaknesses. Therefore, combining the outputs of parsers from both paradigms used to be the standard approach to improve or analyze their performance. However, with the recent adoption of deep contextualized word representations, the chief weakness of graph-based models, i.e., their limited scope of features, has been mitigated. Through two popular combination techniques – blending and stacking – we demonstrate that the remaining diversity in the parsing models is reduced below the level of models trained with different random seeds. Thus, an integration no longer leads to increased accuracy. When both parsers depend on BiLSTMs, the graph-based architecture has a consistent advantage. This advantage stems from globally-trained BiLSTM representations, which capture more distant look-ahead syntactic relations. Such representations can be exploited through multi-task learning, which improves the transition-based parser, especially on treebanks with a high ratio of right-headed dependencies.

1 Introduction

Dependency parsers can roughly be divided into two classes: graph-based (Eisner, 1996; McDonald et al., 2005) and transition-based (Yamada and Matsumoto, 2003; Nivre, 2003). The two paradigms differ in their approach to the trade-off between access to contextual features in the output dependency tree and exactness of search (McDonald and Nivre, 2007). The complementary strengths of those paradigms have given grounds to numerous *diversity-based* methods for integrating parsing models (Nivre and McDonald, 2008; Sagae and Lavie, 2006, among others). To date, the methods are commonly used for improving the accuracy of single parsers[1], achieving robust predictions for the silver-standard resource preparation (Schweitzer et al., 2018), or as analysis tools (de Lhoneux et al., 2019).

One of the most significant recent developments in dependency parsing is based on encoding rich sentential context into word representations, such as BiLSTM vectors (Hochreiter and Schmidhuber, 1997; Graves and Schmidhuber, 2005) and deep contextualized word embeddings (Peters et al., 2018; Devlin et al., 2019). Including these representations as features has set a new state of the art for both graph-based and transition-based parsers (Kiperwasser and Goldberg, 2016; Che et al., 2018). However, it also brought the two architectures closer. Kulmizev et al. (2019) showed that after including deep contextualized word embeddings, the average error profiles of graph- and transition-based parsers converge, potentially reducing gains from combining them. On the other hand, the authors also noticed that the underlying trade-off between the parsing paradigms is still visible in their results. Thus, it is an open question to what extent the differences between the parsing paradigms could still be leveraged.

In this paper, we fill the gaps left in understanding the behavior of transition- and graph-based dependency parsers that employ today's state-of-the-art deep contextualized representations. We start from the setting of Kulmizev et al. (2019), i.e., Kiperwasser and Goldberg's (2016) seminal transition-based and graph-based parsers extended with deep contextualized word embeddings. We show that, on average, the differences between BiLSTM-based graph-based and transition-based models are reduced below the level of different random seeds. Interestingly, the diversity needed for a successful integration vanishes al-

[1] See results from the CoNLL 2018 shared task on dependency parsing (Zeman et al., 2018)

Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task, pages 25–39

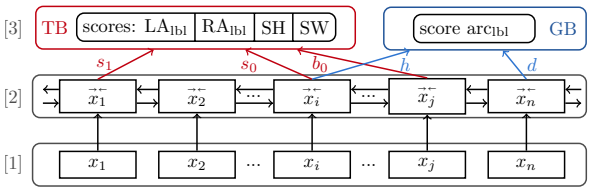


Figure 1: Architectures of BiLSTM-based dependency parsers employed in this work. Layers: [1] word representations, [2] BiLSTMs, [3] multi-layer perceptron.

ready with BiLSTM feature representations and does not change when deep contextualized embeddings are added.

We further consider treebank-specific differences between graph- and transition-based models. Through a set of carefully designed experiments, we show that our graph-based parser has an advantage when parsing treebanks with a high ratio of right-headed dependencies. This advantage comes from globally-trained BiLSTMs and can be exploited by the locally-trained transition-based parser through multi-task learning (Caruana, 1993). This combination improves the performance of the two parsing architectures and narrows the gap between them without requiring additional computational effort at parsing time.

2 Experimental Setup

2.1 Parsing Architecture

We re-implement the basic transition- and graph-based architectures proposed by Kiperwasser and Goldberg (2016) (denoted **K&G**) with a few changes outlined below. We follow Kulmizev et al. (2019) and intentionally abstain from extensions such as Dozat and Manning's (2016) attention layer to keep our experimental setup simple. This enables us to control for all relevant methodological aspects of the architectures. Our hypothesis is that adding more advanced mechanisms would resemble adding contextualized word embeddings, i.e., improve the overall performance but not change the picture regarding parser combination. However, testing this hypothesis is orthogonal to this work.

All the described parsers are implemented with the DyNet library (Neubig et al., 2017).[2] We provide details on used hyperparameters in Appendix A.

[2]The code is available for download on the first author's website.

Deep contextualized word representations. The two most popular models of deep contextualized representations are ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019). Both models have been used with dependency parsers, either for multi-lingual applications (Kondratyuk and Straka, 2019; Schuster et al., 2019) or to improve parsing accuracy (Che et al., 2018; Jawahar et al., 2018; Lim et al., 2018). Recently, Kulmizev et al. (2019) analyzed the influence of both of the models on the K&G architecture and showed that they give similar results, BERT being slightly ahead. Since the scope of our experiments is to analyze the influence of contextualized embeddings on parser integration, and not to analyze differences between different embedding models, we use ELMo, which is more accessible.

ELMo representations encode words within the context of the entire sentence. The representations are built from a linear combination of several layers of BiLSTMs pre-trained on a task of language modeling:

$$\text{ELMo}(x_{1:n}, i) = \gamma \sum_{j=1}^{L} s_j \text{BiLSTM}^j_{LM}(x_{1:n}, i)$$

We use pre-trained ELMo models provided by Che et al. (2018) and train task-specific parameters s_j and γ together with the parser. The final representations are combinations of $L = 3$ layers and have dimensionality 1024.

Word representations. In both transition- and graph-based architectures input tokens are represented in the same way (see level [1] in Figure 1). For a given sentence with words $[w_1, \ldots, w_n]$ and part-of-speech (POS) tags $[t_1, \ldots, t_n]$ each word representation x_i is built from concatenating: embedding of the word, its POS tag, BiLSTM character-based embedding, and word's ELMo representation:

$$x_i = e(w_i) \circ e(t_i) \circ \text{BiLSTM}_{ch}(w_i) \circ \text{ELMo}(x_{1:n}, i)$$

Word embeddings are initialized with the pre-trained fastText vectors (Grave et al., 2018) and trained together with the model. The representations x_i are passed to the BiLSTM feature extractors (level [2]) and represented by a vector $\vec{x}_i = \text{BiLSTM}(x_{1:n}, i)$.

Transition-based parser. The part of the architecture that is specific to the transition-based

K&G parser is colored red in Figure 1. For every configuration consisting of a stack, buffer, and the current set of arcs, the parser builds a feature set of three items: the two top-most items of the stack and the first item on the buffer (denoted s_0, s_1, and b_0). Next, it concatenates their BiLSTM vectors and passes on to a multi-layer perceptron (MLP, level [3] in Figure 1). The MLP scores all possible transitions, and the highest-scoring one is applied to proceed to the next configuration.

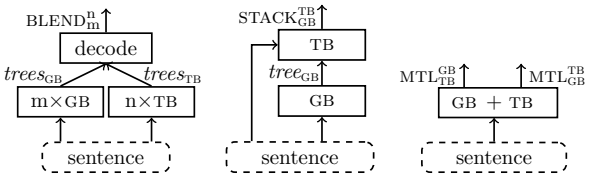
Our implementation (denoted **TB**) uses the arc-standard transition system extended with the SWAP transition (Nivre, 2009) and can thus handle non-projective trees.[3] We use Nivre et al.'s (2009) lazy SWAP oracle for training. Labels are predicted together with the transitions.

For analysis, we also use variants of TB trained without BiLSTMs. In these cases, vectors x_i are passed directly to the MLP layer (similarly to Chen and Manning (2014)), and the implicit context encoded by the BiLSTMs is lost. We compensate for it by using Kiperwasser and Goldberg's (2016) *extended* feature set, which adds the embedding information of eight additional tokens in the structural context of the parser state.

Graph-based parser. The specific parts of the graph-based K&G parser are highlighted in blue in Figure 1. At parsing time, every pair of tokens $\langle x_i, x_j \rangle$ yields a feature set $\{\overrightarrow{x_i}, \overleftarrow{x_j}\}$. The BiLSTM representations are concatenated and passed to an MLP to compute the score for an arc $x_i \xrightarrow{lbl} x_j$ for every possible dependency label lbl (unlike the original K&G implementation, we predict labels together with the arcs). To find the highest-scoring tree, we apply the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). We denote this architecture **GB**.

In experiments where GB is trained without BiLSTMs, we extend the feature set with surface features known from classic graph-based parsers, such as distance between head and dependent, and words at the distance of 1 and 2 from heads and dependents (McDonald et al., 2005).

[3] We performed multiple experiments within the K&G architecture by differentiating transition-systems (e.g., removing SWAP, using the arc-hybrid system (Kuhlmann et al., 2011), and adding the dynamic oracle (Goldberg and Nivre, 2012, 2013)), graph-decoders (testing Eisner's (1996) algorithm), feature sets (also extended feature set from Kiperwasser and Goldberg (2016)), and word representations. In all the tested scenarios, the general picture was essentially the same. Therefore, we present results only for the best-performing configurations.



(a) Blending (b) Stacking (c) Multi-task learning

Figure 2: Schematic illustration of the integration methods used in this work.

2.2 Integration Methods

Parser combination approaches can be divided into two categories: methods that integrate base parsers at prediction time and training time. We use one well-established representative from each of the categories, i.e., blending and feature-based stacking, respectively. Additionally, for analysis purposes, we combine the two parsers through multi-task learning.

Blending (see Figure 2a), also known as reparsing (Sagae and Lavie, 2006), is a parsing-time integration method. It consists of running basic models in separation and combining their outputs into one graph. Weights in this graph depend on how many basic models predicted a particular arc. Finally, a graph-based decoder is used to find the maximum spanning tree in the combined graph.

In our implementation, we use the Chu-Liu-Edmonds algorithm to find the final tree. For every resulting arc, we select the most frequent label across all the labels previously assigned to it. Blending needs at least three basic models to apply the voting scheme. Therefore, we follow Kuncoro et al. (2016) and train multiple instances of models with random seeds and denote **BLEND$^{n}_{m}$** a combination of $m\times$GB and $n\times$TB parsers. For analysis, we vary the ratio of TB and GB models while leaving the total number of models constant at 6 for a fair comparison.

Feature-based stacking (see Figure 2b) was introduced by Nivre and McDonald (2008) and Martins et al. (2008). It involves running two parsers in sequence so that the second (level-1) parser can use the output of the first (level-0) parser as features (denoted **STACK$^{level\text{-}1}_{level\text{-}0}$**).

To generate training data for the level-1 parser, we apply 10-fold cross-validation on the training sets with the level-0 parser. Then, we follow Ouchi et al. (2014) and extract stacking features from the level-0 parser's predictions in the form

of supertags. More precisely, for every word w_i, we build its supertag by filling the template `label/hdir+hasLdep_hasRdep`, where `label` is the dependency relation, `hdir` denotes relative head direction, and `hasLdep/hasRdep` mark presence of left/right dependents. Such supertags are then, similarly to POS tags, represented as embeddings and concatenated with other representations to build x_i. The dimensionality and type of information encoded in the stacking representations were determined in exploratory experiments on the English development data and left unchanged for other languages.

Multi-task learning (see Figure 2c) allows combining the transition- and graph-based K&G parsers by sharing their BiLSTM representations (level [2] in Figure 1). We keep feature extraction and MLP layers separate, and do not enforce any agreement between the two decoders. Effectively this means that training yields two parsers that can be applied independently: one transition-based (denoted $\text{MTL}^{\text{TB}}_{\text{GB}}$) and one graph-based ($\text{MTL}^{\text{GB}}_{\text{TB}}$).

We use a straightforward MTL training protocol: for every sentence, we calculate the BiLSTM representations $\vec{x_i}$ and collect all local losses from both tasks (TB and GB). Then, the losses are summed and the model parameters are updated through backpropagation. We note in passing that this training protocol leaves many options for improvements, such as adding weights to losses from different tasks (Shi et al., 2017b), sharing representations on different levels of BiLSTMs (Søgaard and Goldberg, 2016), or employing stack-propagation (Zhang and Weiss, 2016). We abstain from such extensions as they are orthogonal to the central points of our analysis.

2.3 Data Sets and Preprocessing

We conduct experiments on a selection of thirteen treebanks from Universal Dependencies v2.4 (Nivre et al., 2019): Arabic (ar_padt), Basque (eu_bdt), Chinese (zh_gsd), English (en_ewt), Finnish (fi_tdt), Hebrew (he_htb), Hindi (hi_hdtb), Italian (it_isdt), Japanese (ja_gsd), Korean (ko_gsd), Russian (ru_syntagrus), Swedish (sv_talbanken), and Turkish (tr_imst). This selection was proposed by Kulmizev et al. (2019) and varies in terms of language family, domain, and amount of non-projective arcs.

We use automatically predicted universal POS tags in all the experiments. The tags are assigned using a CRF tagger (Mueller et al., 2013). We annotate the training sets via 5-fold jackknifing.

2.4 Evaluation and Analysis

We evaluate the experiments using Labeled Attachment Score (LAS).[4] We train models for 30 epochs and select the best model based on development LAS. For the results on the test sets, we follow Reimers and Gurevych's (2018) recommendation and report averages and standard deviations from six models trained with different random seeds. We test for significance using the Wilcoxon rank-sum test with p-value < 0.05.

An analysis is carried out on the development sets in order not to compromise the test sets. We follow Kulmizev et al. (2019) and sample the same number of sentences from every development set (484 sentences since this is the size of the smallest one). We then aggregate results from three models trained with different random seeds and present the combined results.

3 Diversity-Based Integration

We start by evaluating the two integration methods (STACK and BLEND) and applying them to our transition- and graph-based parsers (TB and GB).

Average results. The first column in Table 1 gives the average results. In the case of stacking, the performance of combined models is almost the same as that of the baseline models. Small improvements are noticeable for $\text{STACK}^{\text{TB}}_{\text{GB}}$ vs. TB, but they are statistically significant only for one treebank. Comparing $\text{STACK}^{\text{GB}}_{\text{TB}}$ vs. GB we even notice a small average drop of 0.08 LAS. In the case of blending, the method does provide big improvements over single baselines (BLEND^{6}_{0} vs. TB and BLEND^{0}_{6} vs. GB). However, those improvements are not coming from integrating different paradigms since BLEND^{3}_{3} achieves the same average performance as BLEND^{0}_{6}, which uses only GB.

There are two possible explanations for lack of gains from integrating parsing paradigms: either (1) in general, the neural models are simply not capable of benefiting from such combination, or (2) feature representations based on the BiLSTMs and the deep contextualized representations bring the architectures too close to each other for the integration to be beneficial. In Section 4, we investigate which of those two situations takes place.

[4] The percentage of tokens that received the correct head and label.

	avg.	ar	en	eu	fi	he	hi	it	ja	ko	ru	sv	tr	zh
TB	84.60	82.59	86.61	79.96	86.75	85.57	91.00	90.61	93.47	82.17	90.30	86.52	63.94	80.27
STACK$^{TB}_{GB}$	84.72	82.59	86.72	80.35	86.38†	85.94†	91.19	90.83	93.32	81.95	90.60	86.65	64.18	80.71
GB	85.40	82.95	86.97	82.21	87.16	86.55	91.58	91.18	93.34	82.99	90.90	87.09	66.19	81.11
STACK$^{GB}_{TB}$	85.32	83.02	87.15	81.71	87.47	86.62	91.32†	91.22	93.39	82.62	90.83	86.97	66.03	80.80
BLEND$^{6}_{0}$	86.06	83.66	87.85	82.19	88.25	86.87	91.79	91.56	93.92	84.02	91.34	88.10	66.90	82.31
BLEND$^{0}_{6}$	86.63	84.09	87.95	83.98	88.35	87.68	92.16	91.93	93.97	84.41	91.80	88.49	68.52	82.83
BLEND$^{3}_{3}$	86.63	84.08	88.11	83.70	88.61	87.41	92.05	91.95	94.05	84.31	91.85	88.67	68.34	83.01

Table 1: Average (from six runs) parsing results (LAS) on test sets. † marks statistical significance compared to single model baselines (p-value < 0.05). Corresponding standard deviations are provided in Table 4 in Appendix A. Since blending already involves multiple models, we run it only once and do not test the results for significance.

Treebank-specific results. Next, we take a closer look at the treebank-specific accuracy. Comparing single baselines (TB vs. GB), we note that GB has a clear advantage over TB. It surpasses TB on twelve out of thirteen treebanks (all improvements are significant). We reproduce analysis from Kulmizev et al. (2019) and confirm that this advantage is consistent across arcs of different lengths, distances to root, and sentences with different sizes (we provide corresponding plots in Appendix A). Interestingly, the dominance of GB over TB significantly differs across treebanks and is especially prominent for more challenging ones, e.g., with small amounts of training data or a high level of non-projectivity. For instance, the largest difference of 2.25 LAS is visible for Basque, which is the treebank with the largest number of non-projective arcs, and Turkish, which has the smallest training dataset. Moreover, those are the treebanks where STACK$^{TB}_{GB}$ offers small improvements (0.39 LAS and 0.24 LAS, respectively), but both STACK$^{GB}_{TB}$ and BLEND$^{3}_{3}$ cannot make use of the diversity in predictions of the two models and cause the accuracy to drop (comparing STACK$^{GB}_{TB}$ vs. GB and BLEND$^{3}_{3}$ vs. BLEND$^{0}_{6}$). In the case of non-neural parsers, a big gap between the performance of a strong graph-based model and a greedy transition-based model does not prevent the former to learn from the latter (Faleńska et al., 2015). Therefore, the questions arise where those treebank-specific differences come from and why integration methods cannot benefit from them. We address these questions in Section 5.

4 Parsing Architectures and Diversity

In this section, we investigate which aspects of the K&G architecture are responsible for no gains from the integration. For this purpose, we run ablation experiments and apply blending and stacking on models trained with and without BiLSTMs and with and without ELMo representations.

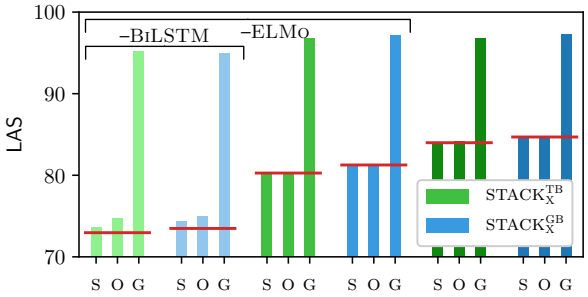
4.1 Feature-based Stacking

We perform stacking with different types of level-0 information. Apart from the standard way, in which TB is stacked on top of GB or vice versa (denoted **O**; for other) we carry out two types of control experiments: **S** (for self), where we stack a model on itself, and **G** (for gold), where gold-standard trees are used as level-0 predictions.

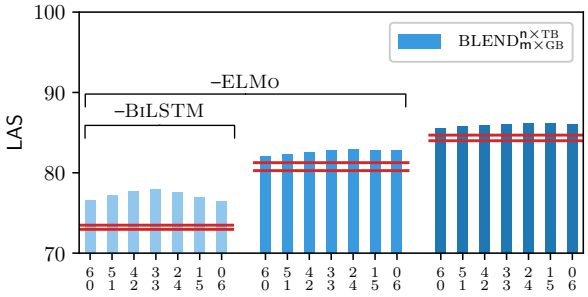
Oracle experiments. Figure 3a displays results for stacking with different level-0 information. We immediately see that scenario G, in which models are stacked on gold-standard trees, exhibits almost perfect performance. Regardless of the level-1 parser and employment of BiLSTMs and ELMo, all models achieve accuracy higher than 95 LAS, proving that they are capable of learning from the stacking representations.

Influence of representations. Next, we consider the models which were trained without BiLSTMs and ELMo (left, lightest bars). Surprisingly, for both TB (green) and GB (blue), small improvements can be noticed in the self-application scenario S, which was not the case for non-neural models (Martins et al., 2008; Faleńska et al., 2015). One explanation for this is the diversity of the models coming with random seeds, which was less prominent in their non-neural versions (Reimers and Gurevych, 2017). However, clearer improvements are visible in scenario O, which combines models of different types. Both STACK$^{TB}_{GB}$ and STACK$^{GB}_{TB}$ surpass both of the single baselines, proving that integration is beneficial when BiLSTMs and ELMo are not used.

Considering the case where BiLSTMs are included (middle) changes the picture. Self-



(a) Stacking with different types of level-0 information: S – self, O – other, G – gold.



(b) Blending; top line – GB single model, bottom line – TB single model.

Figure 3: Parsing accuracy (average LAS over thirteen treebanks on dev sets) for diversity-based integration methods when models are trained with or without BiLSTMs and with or without ELMo. Red lines mark the average LAS of the single baseline models.

application behaves almost on par with stacking the parsers on each other. The only modest improvement (amounting to 0.18 LAS on average) occurs for STACK$_{GB}^{TB}$, but it is not enough to surpass a single GB baseline.

As expected, adding ELMo (right, darkest bars) results in big improvements comparing to the models without these representations. However, those improvements do not impact stacking results, and the picture regarding the integration of the architectures stays the same.

4.2 Blending

Figure 3b presents results for blending with different ratios of TB and GB. We start by analyzing models trained without BiLSTMs and ELMo (left). We can observe a pattern we would expect from diverse models: (1) blending always improves over the baselines (signified by red lines); (2) combining models only of one sort (BLEND$_6^0$ or BLEND$_0^6$) yields lower scores than when we introduce more diversity into the combination; (3) the best result is obtained by BLEND$_3^3$, where the same number of TB and GB models is used.

For the models that use BiLSTMs (middle), the gains coming from blending are smaller. For example, BLEND$_0^6$ improves TB by 1.84 LAS, whereas the corresponding improvement when no BiLSTMs are used is 3.67 LAS. Interestingly, the models show a different pattern when it comes to diversity within the combination. The accuracy of the blend increases with the number of GB models. Although BLEND$_4^2$ achieves the highest accuracy, it surpasses BLEND$_6^0$ by only 0.05 LAS. This suggests that TB models do not bring enough diversity into the combination, and the accuracy of BLEND is mostly influenced by GB models.

Finally, for models that use ELMo (right), improvements over baselines are slightly smaller – BLEND$_6^0$ improves GB by 1.34 LAS comparing to 1.59 LAS when no ELMo is used. However, the picture regarding diversity is the same, and overall performance depends on the number of GB models and not the diversity among combined paradigms.

To conclude, we showed that the performance of TB and GB models can be improved through the traditional diversity-based approaches as long as no BiLSTMs are used. Otherwise, the gains from combination methods decrease considerably. Adding ELMo representations improves the performance of both of the models but has almost no impact on the outcome of the integration.

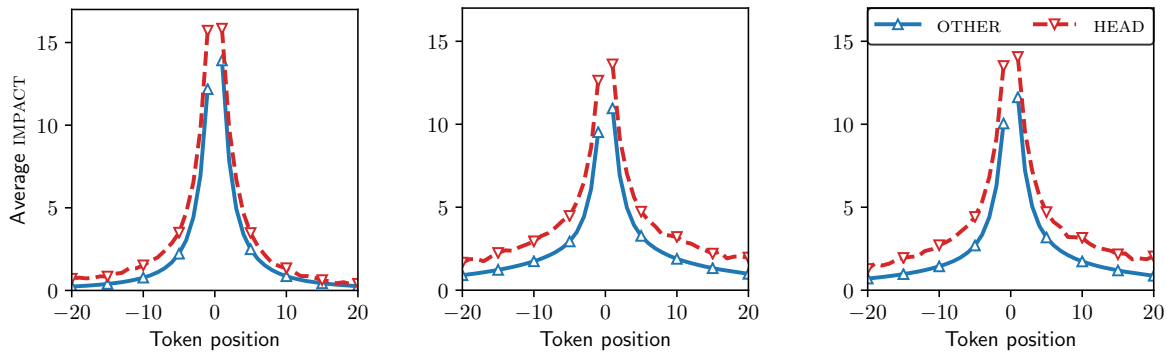
5 Representations and Treebank-Specific Diversity

In the previous section, we saw that BiLSTMs mitigate average benefits from integration methods. One explanation might be that when both TB and GB use the same feature representations, the diversity between them is much smaller, thus reducing the gains the models could draw from each other. However, when comparing treebank-specific results in Table 1, we noticed that in specific cases the two baselines differ considerably. We now investigate where do those differences come from and if they could be beneficial.

5.1 Representation Analysis

First, we take a closer look at information encoded in representations learned by the transition- and graph-based models.

IMPACT metric. We follow Gaddy et al. (2018) and use derivatives to estimate how sensitive a par-



(a) TB (b) GB (c) MTL

Figure 4: The average IMPACT of tokens on BiLSTM vectors with respect to the token position and the structural (gold-standard) relation between them (heads vs. non-heads of the analyzed vector).

ticular part of the architecture is with respect to changes in input. Specifically, we use our metric IMPACT from Falenska and Kuhn (2019) that measures how every BiLSTM representation $\vec{x_i}$ is influenced by every word representation x_j from the sentence. Intuitively, IMPACT can be thought of as a percentage distributed over all words of the sentence – the higher the percentage of x_j the more it influenced the representation of $\vec{x_i}$.

For every sentence from the development set and every vector $\vec{x_i}$ we calculate IMPACT values of all words x_j on $\vec{x_i}$ and bucket those values according to the distance between j and i. Figure 4 shows the average impact of tokens at particular positions. We see the same two general patterns as Gaddy et al. (2018): (1) closer words have larger effects on the representations, and (2) even words 15 or more positions away influence the vectors.

Transition-based parser. For representations trained with TB (Figure 4a) the difference in signals coming from heads and other tokens is bigger on the left side than on the right side (see, e.g., positions -15 and 15). de Lhoneux et al. (2019) provided an explanation for this and showed that for greedy *locally-trained* models, the forward LSTMs could be interpreted as rich history-based features while the backward LSTMs could be thought of as look-ahead features. Since the information to the right mostly (i.e., except for the buffer front) comes from backward LSTMs, it contains, as in the case of standard look-ahead features, less structural relations.

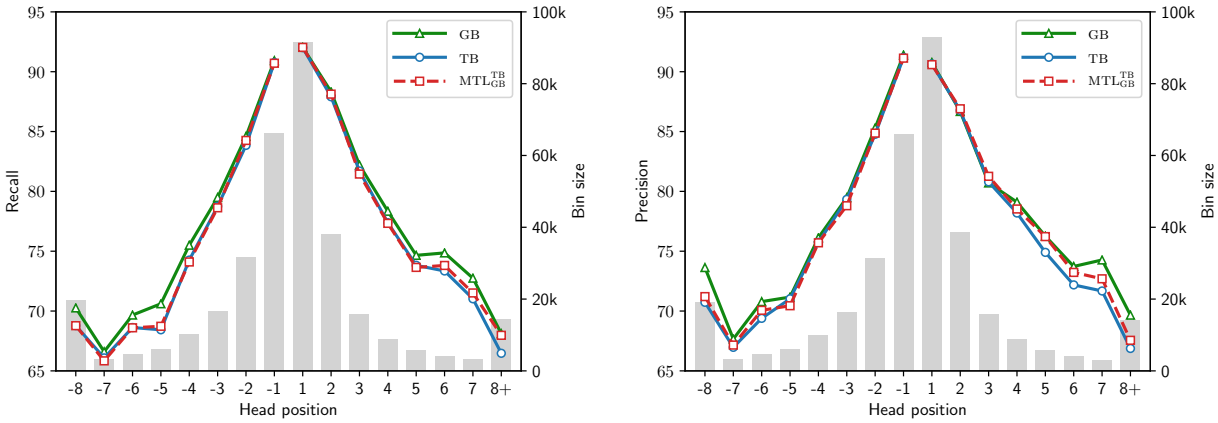
Graph-based parser. Representations trained together with GB (Figure 4b) show a slightly different pattern. Compared to TB, the impact of heads is smaller for tokens closeby, but it deteriorates slower. Since this model is *globally*-trained, the influence of heads does not depend on the side – the plot is almost symmetrical, suggesting that representations encode as much information about syntactic relations on the left as on the right.

5.2 BiLSTMs Integration

Next, we investigate whether the observed differences in the information encoded in BiLSTM representations can explain the advantage of GB over TB. We train new models where we share those intermediate representations between the two parsers through multi-task learning (MTL). We hypothesize that if the advantage of GB stems from global training and the influence it has on the representations, then MTL will re-balance the representations and, as a result, narrow the gap between the two models. We note in passing that MTL is typically carried out on different tasks, often with different training sets. However, it is perfectly possible to consider graph-based and transition-based dependency parsing as two separate tasks trained on the same training set.

IMPACT analysis. Figure 4c displays the IMPACT statistics for MTL models. The plot shows that the BiLSTM representations draw on the advantages from both locally trained TB and globally-trained GB – the distribution has a slightly stronger peak for closer words as in TB, but flattens out more slowly as in GB. This effect is particularly pronounced when comparing the far right (look-ahead) of TB with the MTL distribution, especially as heads become more influential.

Error analysis. To understand how the changes in representations influence the parsing performance, we break down the LAS by dependency



(a) Recall (b) Precision

Figure 5: Dependency recall and precision relative to the head position on development sets.

length and head direction. Figure 5 shows the dependency recall and precision of models with respect to the positions of the heads.[5]

First, we compare TB (blue) with GB (green) and observe that GB has a consistent advantage. However, when comparing recall and precision, we note an interesting difference. In terms of recall (Figure 5a), the plot is symmetrical and the advantage of GB is roughly the same for heads on the left and on the right side of the token. In terms of precision (Figure 5b), both TB and GB behave identically for heads on the left, but the performance of TB drops faster for heads on positions 3 and more to the right.

Second, we analyze $\text{MTL}^{\text{TB}}_{\text{GB}}$ (red). We notice that sharing representations with GB does not influence TB's recall or precision on heads on the left, while for right-headed dependencies precision improves. The model catches up with GB's performance and starts deteriorating much later, for heads on positions 6 and more to the right.

Treebank-specific improvements. Finally, we look at the treebank-specific accuracy of the MTL models. The two bottom rows in Table 2 show the effects of MTL on GB. Sharing representations between the two architectures has a small influence on GB, and on average, improves its performance by 0.18 LAS. Although for few treebanks bigger improvements can be seen, e.g., Chinese (0.39 LAS) or Swedish (0.35 LAS), none of them is statistically significant. Therefore, it is not clear if those improvements come from the actual combination of different parsing paradigms, or MTL in this case acts as additional regularization, ultimately reducing overfitting during training.

In the case of TB, the average performance is improved through MTL by 0.42 LAS, with statistically significant differences for four treebanks. The biggest gains are visible for the treebanks, where the difference between TB and GB is greatest, such as Basque (0.94 LAS). Interestingly, among the treebanks with the biggest improvements, we can notice Turkish (1.28 LAS) and Chinese (0.52 LAS), which are the two treebanks with the greatest ratio of right-headed arcs (62.58% and 71.86%, respectively). This result is in line with the results of de Lhoneux et al. (2019), who demonstrated that backward LSTMs are especially important for head-final languages.

To conclude, we saw that the advantage of GB over TB stems from global training. The training increases the impact of tokens (far) to the right as compared to a locally trained TB model and translates into an improved prediction of right-headed dependencies. Thus, the distance between the two models is treebank-related and can be reduced through integration methods such as MTL, especially when parsing more challenging treebanks.

6 Related Work

Traditional integration of dependency parsers. Classical integration methods were initially introduced to take advantage of differences in the strengths of the component parsers. Such differences were usually the result of different parsing paradigms, as in the case of feature-based stack-

[5] Dependency recall is defined as the percentage of correct predictions among gold standard arcs with head position p. Precision is the percentage of correct predictions among all predicted arcs. The definitions slightly differ from McDonald and Nivre (2007), who looked at absolute arc lengths.

	avg.	ar	en	eu	fi	he	hi	it	ja	ko	ru	sv	tr	zh
TB	84.60	82.59	86.61	79.96	86.75	85.57	91.00	90.61	93.47	82.17	90.30	86.52	63.94	80.27
$\text{MTL}^{\text{TB}}_{\text{GB}}$	85.02	82.55	86.97	80.90†	87.09	85.90	91.29†	90.97	93.45	82.36	90.58	87.16†	65.22†	80.79
GB	85.40	82.95	86.97	82.21	87.16	86.55	91.58	91.18	93.34	82.99	90.90	87.09	66.19	81.11
$\text{MTL}^{\text{GB}}_{\text{TB}}$	85.58	82.99	87.24†	82.55	87.52	86.68	91.71	91.31	93.47	83.12	91.11	87.44	65.88	81.50

Table 2: Average (from six runs) parsing results (LAS) on test sets. † marks statistical significance compared to single model baselines (p-value < 0.05). Corresponding standard deviations are provided in Table 5 in Appendix A.

ing, blending, or beam search-based transition-based parsers with features strongly inspired by graph-based models (Zhang and Clark, 2008; Bohnet and Kuhn, 2012). However, combining parsers that process input left-to-right and right-to-left (Hall et al., 2007; Attardi and Dell'Orletta, 2009), or even parsers and sequence labelers (Faleńska et al., 2015), was also proposed. Blending was usually applied to a mixture of graph-based and transition-based left-to-right and right-to-left parsers (Sagae and Lavie, 2006; Surdeanu and Manning, 2010; Björkelund et al., 2017, among others). Moreover, in the case of stacking, integrating two parsers of the same type gives at most minor improvements (Martins et al., 2008).

Neural-specific ensemble dependency parsers. Since neural network training can be sensitive to initialization (Reimers and Gurevych, 2017), recent ensemble dependency parsers are rather combining models trained with different random seeds than different paradigms. For example, out of 24 teams participating in the CoNLL 2018 Shared Task on dependency parsing (Zeman et al., 2018), five employed ensemble techniques. However, all of them took advantage of either diversity coming from random seeds or different languages.

Neural parsers of the same type can be combined by taking the sum of their MLP scores (Che et al., 2017), averaging softmax scores (Che et al., 2018), or through re-parsing (Kuncoro et al., 2016). The last authors also showed that such an ensemble could be distilled into a single graph-based parser. Finally, Shi et al. (2017b) used MTL in a similar way to ours. They shared BiLSTMs between three parsers to speed up their training time. However, all the models where globally-trained and the authors did not evaluate if the combination improved their performance.

7 Discussion and Conclusion

In this paper, we investigated the recent advances in dependency parsing from the perspective of the traditional integration methods. These methods are known for exploiting diversity in the strengths and weaknesses of transition- and graph-based parsing paradigms. We found out that when models use BiLSTMs, such diversity is on the level of different random seeds. Adding deep contextualized representations on top of BiLSTMs improves the performance of both parsers but does not change the picture regarding the integration.

Rich-feature sets used to be the advantage of the transition-based parsers. Now that the parsers do not need structural features (Falenska and Kuhn, 2019), the graph-based parsers have an advantage that the locally-trained transition-based parsers cannot make up for. Therefore, improving parsers through combination methods is not as straightforward as it used to be. Such a combination has to take into consideration the specificity of the treebank and depend on whether accuracy or parsing time is the priority. The greatest gains in accuracy can be obtained by blending multiple graph-based models. However, the method comes with the cumbersome overhead of running multiple predictors at application time. When speed is essential and the accuracy can be sacrificed (Gómez-Rodríguez et al., 2017) greedy transition-based parsers or even sequence labelers are the preferable choices (Strzyz et al., 2019). In such cases, alternative integration approaches such as multi-task learning can boost the performance of locally-trained models without requiring additional computational effort at parsing time.

Introduction of BiLSTMs into dependency parsers had another consequence, i.e., it enabled the use of exact search algorithms for transition-based parsers (Shi et al., 2017a; Gómez-Rodríguez et al., 2018). Therefore, it is an interesting question if the error profiles of such parsers are even less distinguishable from the graph-based outputs. We leave this question for future work.

Acknowledgments

This work was in part supported by the Deutsche Forschungsgemeinschaft (DFG) via the SFB 732, project D8. Anders Björkelund was funded by AIR Lund Chest pain (VR; grant no 2019-00198). We would like to thank the anonymous reviewers for their comments. We also thank our colleagues Özlem Çetinoğlu and Xiang Yu for many conversations and comments on this work.

References

Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse Revision and Linear Tree Combination for Dependency Parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264, Boulder, Colorado. Association for Computational Linguistics.

Anders Björkelund, Agnieszka Falenska, Xiang Yu, and Jonas Kuhn. 2017. IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51, Vancouver, Canada. Association for Computational Linguistics.

Bernd Bohnet and Jonas Kuhn. 2012. The Best of Both Worlds – A Graph-based Completion Model for Transition-based Parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87, Avignon, France. Association for Computational Linguistics.

Richard Caruana. 1993. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann.

Wanxiang Che, Jiang Guo, Yuxuan Wang, Bo Zheng, Huaipeng Zhao, Yang Liu, Dechuan Teng, and Ting Liu. 2017. The HIT-SCIR System for End-to-End Parsing of Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 52–62, Vancouver, Canada. Association for Computational Linguistics.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2016. Deep Biaffine Attention for Neural Dependency Parsing. *CoRR*, abs/1611.01734.

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

Jason M. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

Agnieszka Faleńska, Anders Björkelund, Özlem Çetinoğlu, and Wolfgang Seeker. 2015. Stacking or Supertagging for Dependency Parsing – What's the Difference? In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 118–129, Bilbao, Spain. Association for Computational Linguistics.

Agnieszka Falenska and Jonas Kuhn. 2019. The (Non-)Utility of Structural Features in BiLSTM-based Dependency Parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's Going On in Neural Constituency Parsers? An Analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

Yoav Goldberg and Joakim Nivre. 2013. Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Carlos Gómez-Rodríguez, Iago Alonso-Alonso, and David Vilares. 2017. How Important is Syntactic Parsing Accuracy? An Empirical Evaluation on Rule-Based Sentiment Analysis. *Artificial Intelligence Review*, pages 1–17.

Carlos Gómez-Rodríguez, Tianze Shi, and Lillian Lee. 2018. Global Transition-based Non-projective Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2664–2675, Melbourne, Australia. Association for Computational Linguistics.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning Word Vectors for 157 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Alex Graves and Jürgen Schmidhuber. 2005. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5):602–610.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryigit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-C oNLL 2007*, pages 933–939, Prague, Czech Republic. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ganesh Jawahar, Benjamin Muller, Amal Fethi, Louis Martin, Éric Villemonte de la Clergerie, Benoît Sagot, and Djamé Seddah. 2018. ELMoLex: Connecting ELMo and Lexicon Features for Dependency Parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 223–237, Brussels, Belgium. Association for Computational Linguistics.

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.

Dan Kondratyuk and Milan Straka. 2019. 75 Languages, 1 Model: Parsing Universal Dependencies Universally. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682, Portland, Oregon, USA. Association for Computational Linguistics.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing - A Tale of Two Parsers Revisited. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2755–2768, Hong Kong, China. Association for Computational Linguistics.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an Ensemble of Greedy Dependency Parsers into One MST Parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas. Association for Computational Linguistics.

Miryam de Lhoneux, Miguel Ballesteros, and Joakim Nivre. 2019. "Recursive Subtree Composition in LSTM-Based Dependency Parsing". In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1566–1576, Minneapolis, Minnesota. Association for Computational Linguistics.

KyungTae Lim, Cheoneum Park, Changki Lee, and Thierry Poibeau. 2018. SEx BiST: A Multi-Source Trainable Parser with Deep Contextualized Lexical Representations. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 143–152, Brussels, Belgium. Association for Computational Linguistics.

André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking Dependency Parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166, Honolulu, Hawaii. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, Prague, Czech Republic. Association for Computational Linguistics.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient Higher-Order CRFs for Morphological Tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332. Association for Computational Linguistics.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The Dynamic Neural Network Toolkit. *CoRR*, abs/1701.03980.

Joakim Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT*, pages 149–160, Nancy, France.

Joakim Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359. Association for Computational Linguistics.

Joakim Nivre, Mitchell Abrams, Željko Agić, et al. 2019. Universal Dependencies 2.4. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76. Association for Computational Linguistics.

Joakim Nivre and Ryan McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio. Association for Computational Linguistics.

Hiroki Ouchi, Kevin Duh, and Yuji Matsumoto. 2014. Improving Dependency Parsers with Supertags. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 154–158, Gothenburg, Sweden. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2017. Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, Copenhagen, Denmark. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2018. Why Comparing Single Performance Scores Does Not Allow to Draw Conclusions About Machine Learning Approaches. *CoRR*, abs/1803.09578.

Kenji Sagae and Alon Lavie. 2006. Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA. Association for Computational Linguistics.

Tal Schuster, Ori Ram, Regina Barzilay, and Amir Globerson. 2019. Cross-Lingual Alignment of Contextual Word Embeddings, with Applications to Zero-shot Dependency Parsing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1599–1613, Minneapolis, Minnesota. Association for Computational Linguistics.

Katrin Schweitzer, Kerstin Eckart, Markus Gärtner, Agnieszka Falenska, Arndt Riester, Ina Rösiger, Antje Schweitzer, Sabrina Stehwien, and Jonas Kuhn. 2018. German Radio Interviews: The GRAIN Release of the SFB732 Silver Standard Collection. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Tianze Shi, Liang Huang, and Lillian Lee. 2017a. Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23. Association for Computational Linguistics.

Tianze Shi, Felix G. Wu, Xilun Chen, and Yao Cheng. 2017b. Combining Global Models for Parsing Universal Dependencies. In *Proceedings of the CoNLL*

2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 31–39, Vancouver, Canada. Association for Computational Linguistics.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235, Berlin, Germany. Association for Computational Linguistics.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Viable Dependency Parsing as Sequence Labeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 717–723, Minneapolis, Minnesota. Association for Computational Linguistics.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, California. Association for Computational Linguistics.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206, Nancy, France.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved Representation Learning for Syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566, Berlin, Germany. Association for Computational Linguistics.

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. Association for Computational Linguistics.

A Appendix

Word embedding dimension	300
POS tag embedding dimension	20
Character embedding dimension	24
Supertag embedding dimension	30
ELMo representation dimension	1024
Hidden units in MLP	100
BiLSTM layers	2
BiLSTM dimensions	125
BiLSTM dropout	0.33
Character-based BiLSTM dimensions	100
α for word dropout	0.25
Trainer	Adam
Non-lin function	tanh

Table 3: Hyperparameters for the parsers.

	ar	en	eu	fi	he	hi	it	ja	ko	ru	sv	tr	zh
TB	0.089	0.184	0.297	0.162	0.250	0.169	0.201	0.213	0.432	0.116	0.153	0.571	0.495
STACK$^{\text{TB}}_{\text{GB}}$	0.145	0.180	0.212	0.176	0.136	0.093	0.136	0.158	0.288	0.042	0.209	0.406	0.200
GB	0.159	0.166	0.314	0.216	0.226	0.130	0.159	0.108	0.414	0.064	0.174	0.280	0.261
STACK$^{\text{GB}}_{\text{TB}}$	0.131	0.143	0.234	0.105	0.339	0.116	0.067	0.105	0.234	0.056	0.171	0.459	0.351

Table 4: Standard deviation for results in Table 1.

	ar	en	eu	fi	he	hi	it	ja	ko	ru	sv	tr	zh
TB	0.089	0.184	0.297	0.162	0.250	0.169	0.201	0.213	0.432	0.116	0.153	0.571	0.495
MTL$^{\text{TB}}_{\text{GB}}$	0.249	0.144	0.325	0.385	0.368	0.134	0.409	0.167	0.239	0.078	0.167	0.336	0.604
GB	0.159	0.166	0.314	0.216	0.226	0.130	0.159	0.108	0.414	0.064	0.174	0.280	0.261
MTL$^{\text{GB}}_{\text{TB}}$	0.225	0.058	0.170	0.237	0.356	0.081	0.298	0.157	0.319	0.087	0.152	0.290	0.425

Table 5: Standard deviation for results in Table 2.

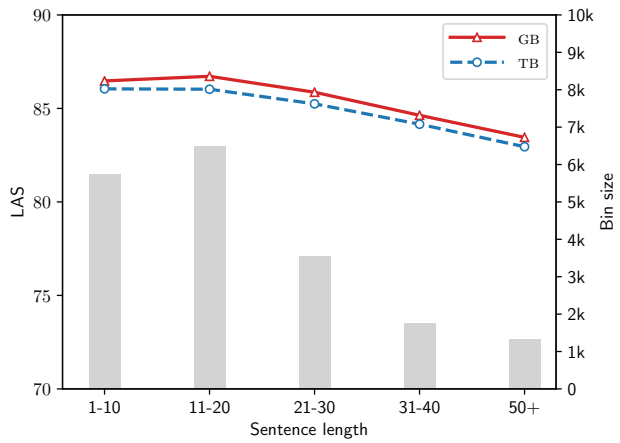
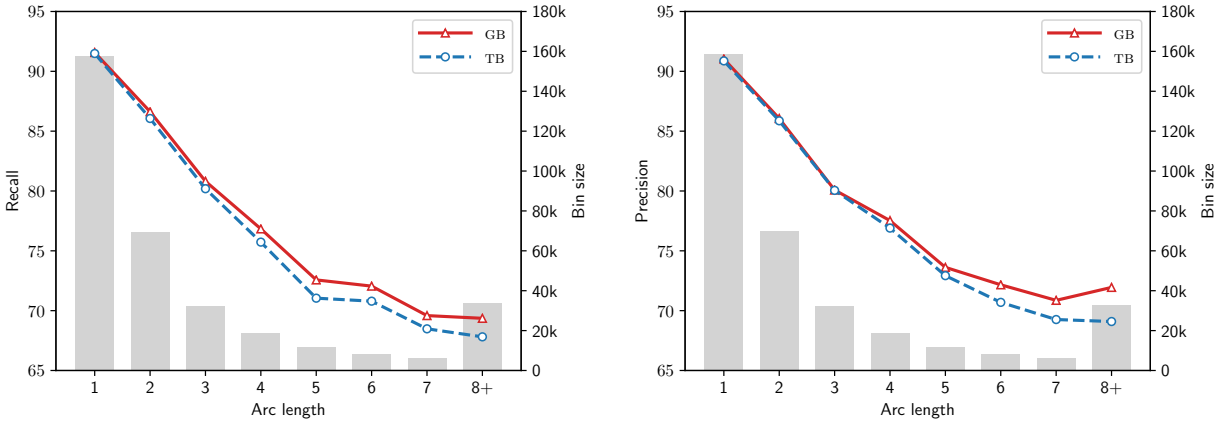


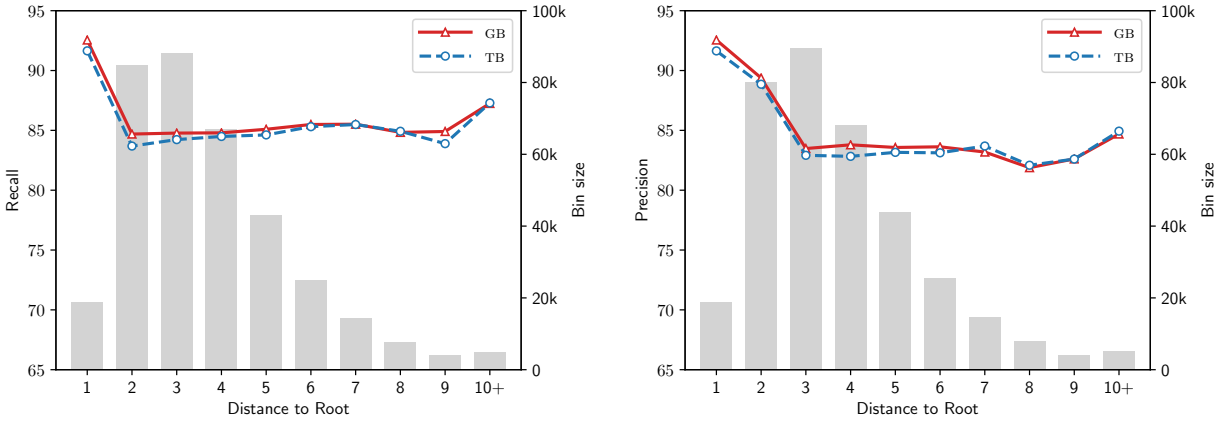
Figure 6: Average LAS relative to sentence length on development sets.



(a) Recall

(b) Precision

Figure 7: The dependency recall and precision relative to arc length on development sets.



(a) Recall

(b) Precision

Figure 8: The dependency recall and precision relative to the distance to root on development sets.