Recursive Subtree Composition in LSTM-Based Dependency Parsing

Miryam de Lhoneux[♠] **Miguel Ballesteros**[♢] **Joakim Nivre**[♠]
[♠] Department of Linguistics and Philology, Uppsala University
[♢] IBM Research AI, Yorktown Heights, NY
{miryam.de_lhoneux,joakim.nivre}@lingfil.uu.se
miguel.ballesteros@ibm.com

Abstract

The need for tree structure modelling on top of sequence modelling is an open issue in neural dependency parsing. We investigate the impact of adding a tree layer on top of a sequential model by recursively composing subtree representations (composition) in a transition-based parser that uses features extracted by a BiLSTM. Composition seems superfluous with such a model, suggesting that BiLSTMs capture information about subtrees. We perform model ablations to tease out the conditions under which composition helps. When ablating the backward LSTM, performance drops and composition does not recover much of the gap. When ablating the forward LSTM, performance drops less dramatically and composition recovers a substantial part of the gap, indicating that a forward LSTM and composition capture similar information. We take the backward LSTM to be related to lookahead features and the forward LSTM to the rich history-based features both crucial for transition-based parsers. To capture history-based information, composition is better than a forward LSTM on its own, but it is even better to have a forward LSTM as part of a BiLSTM. We correlate results with language properties, showing that the improved lookahead of a backward LSTM is especially important for head-final languages.

1 Introduction

Recursive neural networks allow us to construct vector representations of trees or subtrees. They have been used for constituency parsing by Socher et al. (2013) and Dyer et al. (2016) and for dependency parsing by Stenetorp (2013) and Dyer et al. (2015), among others. In particular, Dyer et al. (2015) showed that composing representations of subtrees using recursive neural networks can be beneficial for transition-based dependency parsing. These results were further strengthened in Kuncoro et al. (2017) who showed, using ablation experiments, that composition is key in the Recurrent Neural Network Grammar (RNNG) generative parser by Dyer et al. (2016).

In a parallel development, Kiperwasser and Goldberg (2016b) showed that using BiLSTMs for feature extraction can lead to high parsing accuracy even with fairly simple parsing architectures, and using BiLSTMs for feature extraction has therefore become very popular in dependency parsing. It is used in the state-of-the-art parser of Dozat and Manning (2017), was used in 8 of the 10 highest performing systems of the 2017 CoNLL shared task (Zeman et al., 2017) and 10 out of the 10 highest performing systems of the 2018 CoNLL shared task (Zeman et al., 2018).

This raises the question of whether features extracted with BiLSTMs in themselves capture information about subtrees, thus making recursive composition superfluous. Some support for this hypothesis comes from the results of Linzen et al. (2016) which indicate that LSTMs can capture hierarchical information: they can be trained to predict long-distance number agreement in English. Those results were extended to more constructions and three additional languages by Gulordava et al. (2018). However, Kuncoro et al. (2018) have also shown that although sequential LSTMs can learn syntactic information, a recursive neural network which explicitly models hierarchy (the RNNG model from Dyer et al. (2015)) is better at this: it performs better on the number agreement task from Linzen et al. (2016).

To further explore this question in the context of dependency parsing, we investigate the use of recursive composition (henceforth referred to as *composition*) in a parser with an architecture like the one in Kiperwasser and Goldberg (2016b). This allows us to explore variations of features and isolate the conditions under which composi-

Proceedings of NAACL-HLT 2019, pages 1566–1576
Minneapolis, Minnesota, June 2 - June 7, 2019.

tion is helpful. We hypothesise that the use of a BiLSTM for feature extraction makes it possible to capture information about subtrees and therefore makes the use of subtree composition superfluous. We hypothesise that composition becomes useful when part of the BiLSTM is ablated, the forward or the backward LSTM. We further hypothesise that composition is most useful when the parser has no access to information about the function of words in the context of the sentence given by POS tags. When using POS tags, the tagger has indeed had access to the full sentence. We additionally look at what happens when we ablate character vectors which have been shown to capture information which is partially overlapping with information from POS tags. We experiment with a wider variety of languages than Dyer et al. (2015) in order to explore whether the usefulness of different model variants vary depending on language type.

2 K&G Transition-Based Parsing

We define the parsing architecture introduced by Kiperwasser and Goldberg (2016b) at a high level of abstraction and henceforth refer to it as K&G. A K&G parser is a greedy transition-based parser.[1] For an input sentence of length n with words $w_1, \ldots, w_n$, a sequence of vectors $x_{1:n}$ is created, where the vector x_i is a vector representation of the word w_i. We refer to these as *type* vectors, as they are the same for all occurrences of a word type. Type vectors are then passed through a feature function which learns representations of words in the context of the sentence.

$$x_i = e(w_i)$$
$$v_i = f(x_{1:n}, i)$$

We refer to the vector v_i as a *token* vector, as it is different for different tokens of the same word type. In Kiperwasser and Goldberg (2016b), the feature function used is a BiLSTM.

As is usual in transition-based parsing, parsing involves taking transitions from an initial configuration to a terminal one. Parser configurations are represented by a stack, a buffer and set of dependency arcs (Nivre, 2008). For each configuration c, the feature extractor concatenates the token representations of core elements from the stack and buffer. These token vectors are passed to a classifier, typically a Multilayer Perceptron (MLP). The MLP scores transitions together with the arc labels for transitions that involve adding an arc. Both the word type vectors and the BiLSTMs are trained together with the model.

3 Composing Subtree Representations

Dyer et al. (2015) looked at the impact of using a recursive composition function in their parser, which is also a transition-based parser but with an architecture different from K&G. They make use of a variant of the LSTM called a stack LSTM. A stack LSTM has push and pop operations which allow passing through states in a tree structure rather than sequentially. Stack LSTMs are used to represent the stack, the buffer, and the sequence of past parsing actions performed for a configuration.

The words of the sentence are represented by vectors of the word types, together with a vector representing the word's POS tag. In the initial configuration, the vectors of all words are in the buffer and the stack is empty. The representation of the buffer is the end state of a backward LSTM over the word vectors. As parsing evolves, the word vectors are popped from the buffer, pushed to and popped from the stack and the representations of stack and buffer get updated.

Dyer et al. (2015) define a recursive composition function and compose tree representations incrementally, as dependents get attached to their head. The composed representation c is built by concatenating the vector h of the head with the vector of the dependent d, as well as a vector r representing the label paired with the direction of the arc. That concatenated vector is passed through an affine transformation and then through a $tanh$ non-linear activation.

$$c = tanh(W[h; d; r] + b)$$

They create two versions of the parser. In the first version, when a dependent is attached to a head, the word vector of the head is replaced by a composed vector of the head and dependent. In the second version, they simply keep the vector of the head when attaching a dependent to a head. They observe that the version with composition is substantially better than the version without, by 1.3 LAS points for English (on the Penn Treebank (PTB) test set) and 2.1 for Chinese (on the Chinese Treebank (CTB) test set).

[1] Kiperwasser and Goldberg (2016b) also define a graph-based parser with similar feature extraction, but we focus on transition-based parsing.

Their parser uses POS tag information. POS tags help to disambiguate between different functional uses of a word and in this way give information about the use of the word in context. We hypothesise that the effect of using a recursive composition function is stronger when not making use of POS tags.

4 Composition in a K&G Parser

The parsing architectures of the stack LSTM parser (S-LSTM) and K&G are different but have some similarities.[2] In both cases, the configuration is represented by vectors obtained by LSTMs. In K&G, it is represented by the token vectors of top items of the stack and the first item of the buffer. In the S-LSTM, it is represented by the vector representations of the entire stack, buffer and sequence of past transitions.

Both types of parsers learn vector representations of word types which are passed to an LSTM. In K&G, they are passed to an LSTM in a feature extraction step that happens before parsing. The LSTM in this case is used to learn vectors that have information about the context of each word, a token vector. In the S-LSTM, word type vectors are passed to Stack LSTMs as parsing evolves. In this case, LSTMs are used to learn vector representations of the stack and buffer (as well as one which learns a representation of the parsing action history).

When composition is not used in the S-LSTM, word vectors represent word types. When composition is used, as parsing evolves, the stack and buffer vectors get updated with information about the subtrees they contain, so that they gradually become contextualised. In this sense, those vectors become more like token vectors in K&G. More specifically, as explained in the previous section, when a dependent is attached to its head, the composition function is applied to the vectors of head and dependent and the vector of the head is replaced by this composed vector.

We cannot apply composition on type vectors in the K&G architecture, since they are not used after the feature extraction step and hence cannot influence the representation of the configuration. Instead, we apply composition on the token vectors. We embed those composed representations in the same space as the token vectors.

In K&G, like in the S-LSTM, we can create a composition function and compose the representation of subtrees as parsing evolves. We create two versions of the parser, one where word tokens are represented by their token vector. The other where they are represented by their token vector and the vector of their subtree c_i, which is initially just a copy of the token vector ($v_i = f(x_{1:n}, i) \circ c_i$). When a dependent word d is attached to a word h with a relation and direction r, c_i is computed with the same composition function as in the S-LSTM defined in the previous section, repeated below.[3]

This composition function is a simple recurrent cell. Simple RNNs have known shortcomings which have been addressed by using LSTMs, as proposed by Hochreiter and Schmidhuber (1997). A natural extension to this composition function is therefore to replace it with an LSTM cell. We also try this variant. We construct LSTMs for subtrees. We initialise a new LSTM for each new subtree that is formed, that is, when a dependent d is attached to a head h which does not have any dependent yet. Each time we attach a dependent to a head, we construct a vector which is a concatenation of h, d and r. We pass this vector to the LSTM of h. c is the output state of the LSTM after passing through that vector. We denote those models with $+rc$ for the one using an ungated recurrent cell and with $+lc$ for the one using an LSTM cell.

$$c = tanh(W[h; d; r] + b)$$
$$c = \text{LSTM}([h; d; r])$$

As results show (see § 5), neither type of composition seems useful when used with the K&G parsing model, which indicates that BiLSTMs capture information about subtrees. To further investigate this and in order to isolate the conditions under which composition is helpful, we perform different model ablations and test the impact of recursive composition on these ablated models.

First, we ablate parts of the BiLSTMs: we ablate either the forward or the backward LSTM. We therefore build parsers with 3 different feature functions $f(x, i)$ over the word type vectors x_i in the sentence x: a BiLSTM (bi) (our baseline), a backward LSTM (bw) (i.e., ablating the forward LSTM) and a forward LSTM (fw) (i.e., ablating

[2] Note that we use S-LSTM to denote the stack LSTM parser, not the stack LSTM as an LSTM type.

[3] Note that, in preliminary experiments, we tried replacing the vector of the head by the vector of its subtree instead of concatenating the two but concatenating gave much better results.

the backward LSTM):

$$\begin{aligned} bi(x, i) &= \text{BiLSTM}(x_{1:n}, i) \\ bw(x, i) &= \text{LSTM}(x_{n:1}, i) \\ fw(x, i) &= \text{LSTM}(x_{1:n}, i) \end{aligned}$$

K&G parsers with unidirectional LSTMs are, in some sense, more similar to the S-LSTM than those with a BiLSTM, since the S-LSTM only uses unidirectional LSTMs. We hypothesise that composition will help the parser using unidirectional LSTMs in the same way it helps an S-LSTM.

We additionally experiment with the vector representing the word at the input of the LSTM. The most complex representation consists of a concatenation of an embedding of the word type $e(w_i)$, an embedding of the (predicted) POS tag of w_i, $p(w_i)$ and a character representation of the word obtained by running a BiLSTM over the characters $ch_{1:m}$ of w_i (BiLSTM$(ch_{1:m})$).

$$x_i = e(w_i) \circ p(w_i) \circ \text{BiLSTM}(ch_{1:m})$$

Without a POS tag embedding, the word vector is a representation of the word type. With POS information, we have some information about the word in the context of the sentence and the tagger has had access to the full sentence. The representation of the word at the input of the BiLSTM is therefore more contextualised and it can be expected that a recursive composition function will be less helpful than when POS information is not used. Character information has been shown to be useful for dependency parsing first by Ballesteros et al. (2015). Ballesteros et al. (2015) and Smith et al. (2018b) among others have shown that POS and character information are somewhat complementary. Ballesteros et al. (2015) used similar character vectors in the S-LSTM parser but did not look at the impact of composition when using these vectors. Here, we experiment with ablating either or both of the character and POS vectors. We look at the impact of using composition on the full model as well as these ablated models. We hypothesise that composition is most helpful when those vectors are not used, since they give information about the functional use of the word in context.

Parser We use UUParser, a variant of the K&G transition-based parser that employs the arc-hybrid transition system from Kuhlmann et al. (2011) extended with a SWAP transition and a Static-Dynamic oracle, as described in de Lhoneux et al. (2017b)[4]. The SWAP transition is used to allow the construction of non-projective dependency trees (Nivre, 2009). We use default hyperparameters. When using POS tags, we use the universal POS tags from the UD treebanks which are coarse-grained and consistent across languages. Those POS tags are predicted by UDPipe (Straka et al., 2016) both for training and parsing. This parser obtained the 7th best LAS score on average in the 2018 CoNLL shared task (Zeman et al., 2018), about 2.5 LAS points below the best system, which uses an ensemble system as well as ELMo embeddings, as introduced by Peters et al. (2018). Note, however, that we use a slightly impoverished version of the model used for the shared task which is described in Smith et al. (2018a): we use a less accurate POS tagger (UDPipe) and we do not make use of multi-treebank models. In addition, Smith et al. (2018a) use the three top items of the stack as well as the first item of the buffer to represent the configuration, while we only use the two top items of the stack and the first item of the buffer. Smith et al. (2018a) also use an *extended feature set* as introduced by Kiperwasser and Goldberg (2016b) where they also use the rightmost and leftmost children of the items of the stack and buffer that they consider. We do not use that extended feature set. This is to keep the parser settings as simple as possible and avoid adding confounding factors. It is still a near-SOTA model. We evaluate parsing models on the development sets and report the average of the 5 best results in 30 epochs and 5 runs with different random seeds.

Data We test our models on a sample of treebanks from Universal Dependencies v2.1 (Nivre et al., 2017). We follow the criteria from de Lhoneux et al. (2017c) to select our sample: we ensure typological variety, we ensure variety of domains, we verify the quality of the treebanks, and we use one treebank with a large amount of non-projective arcs. However, unlike them, we do not use extremely small treebanks. Our selection is the same as theirs but we remove the tiny treebanks and replace them with 3 others. Our final set is: Ancient Greek (PROIEL), Basque, Chinese, Czech, English, Finnish, French, Hebrew and Japanese.

[4] The code can be found at https://github.com/mdelhoneux/uuparser-composition

5 Results

First, we look at the effect of our different recursive composition functions on the full model (i.e., the model using a BiLSTM for feature extraction as well as both character and POS tag information). As can be seen from Figure 1, recursive composition using an LSTM cell ($+lc$) is generally better than recursive composition with a recurrent cell ($+rc$), but neither technique reliably improves the accuracy of a BiLSTM parser.

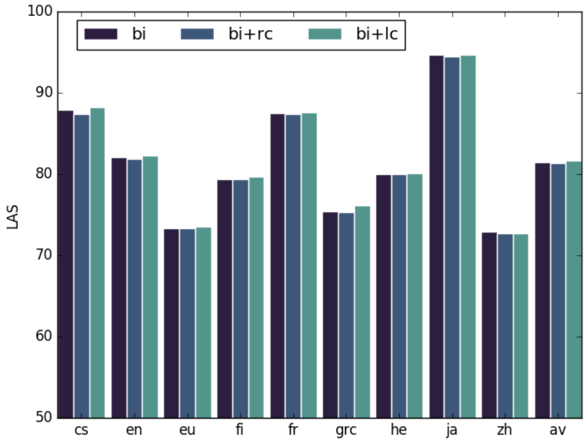

Figure 1: LAS of models using a BiLSTM (bi) without composition, with a recurrent cell ($+rc$) and with an LSTM cell ($+lc$). Bar charts truncated at 50 for visualization purposes.

5.1 Ablating the forward and backward LSTMs

Second, we only consider the models using character and POS information and look at the effect of ablating parts of the BiLSTM on the different languages. The results can be seen in Figure 2. As expected, the BiLSTM parser performs considerably better than both unidirectional LSTM parsers, and the backward LSTM is considerably better than the forward LSTM, on average. It is, however, interesting to note that using a forward LSTM is much more hurtful for some languages than others: it is especially hurtful for Chinese and Japanese. This can be explained by language properties: the right-headed languages suffer more from ablating the backward LSTM than other languages. We observe a correlation between how hurtful a forward model is compared to the baseline and the percentage of right-headed content dependency relations ($R = -0.838$, $p < .01$), see Figure 3.[5]

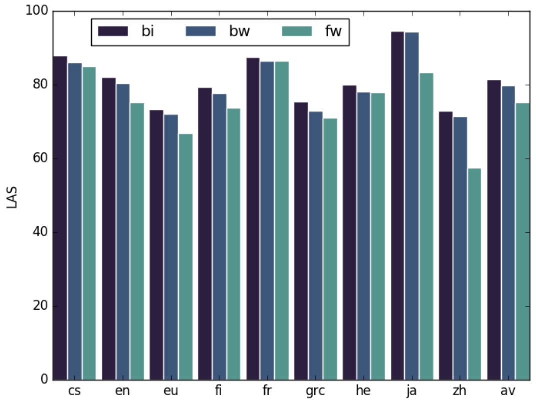

Figure 2: LAS of models using a BiLSTM (bi), backward LSTM (bw) and forward LSTM (fw).

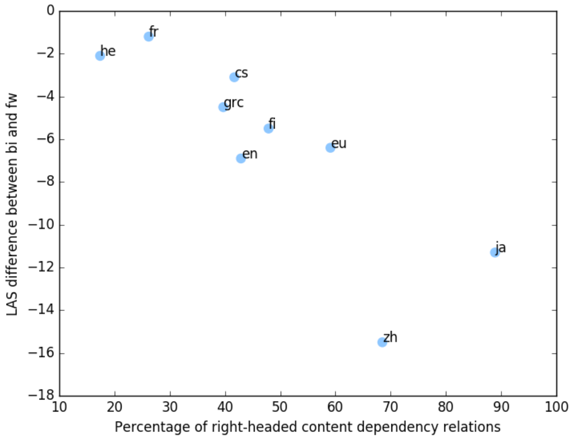

Figure 3: Correlation between how hurtful it is to ablate the backward LSTM and right-headedness of languages.

There is no significant correlation between how hurtful ablating the forward LSTM is and the percentage of left-headed content dependency relations ($p > .05$) indicating that its usefulness is not dependent on language properties. We hypothesise that dependency length or sentence length can play a role but we also find no correlation between how hurtful it is to ablate the forward LSTM and average dependency or sentence length in treebanks. It is finally also interesting to note that the backward LSTM performance is close to the BiLSTMs performance for some languages (Japanese and French).

[5] The reason we only consider content dependency relations is that the UD scheme focuses on dependency relations between content words and treats function words as features of content words to maximise parallelism across languages (de Marneffe et al., 2014).

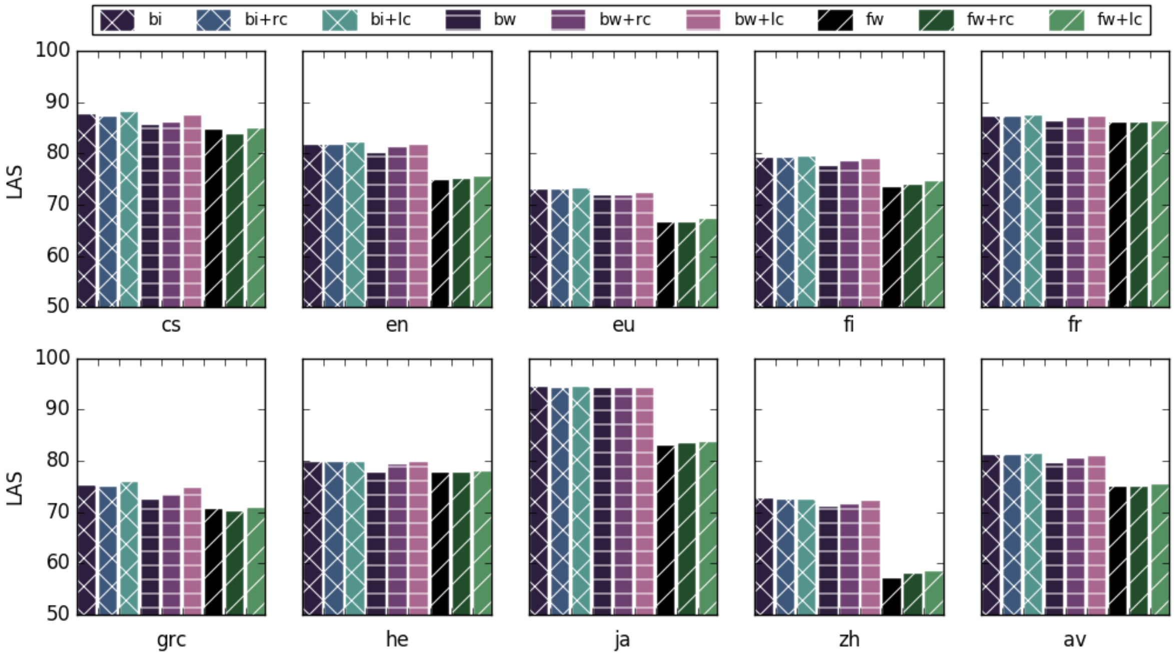

Figure 4: LAS of models using a BiLSTM (bi), backward LSTM (bw) and forward LSTM (fw), without recursive composition, with a recurrent cell ($+rc$) and with a LSTM cell ($+lc$). Bar charts truncated at 50 for visualization purposes.

We now look at the effect of using recursive composition on these ablated models. Results are given in Figure 4. First of all, we observe unsurprisingly that composition using an LSTM cell is much better than using a simple recurrent cell. Second, both types of composition help the backward LSTM case, but neither reliably helps the bi models. Finally, the recurrent cell does not help the forward LSTM case but the LSTM cell does to some extent. It is interesting to note that using composition, especially using an LSTM cell, bridges a substantial part of the gap between the bw and the bi models.

These results can be related to the literature on transition-based dependency parsing. Transition-based parsers generally rely on two types of features: *history-based features* over the emerging dependency tree and *lookahead features* over the buffer of remaining input. The former are based on a hierarchical structure, the latter are purely sequential. McDonald and Nivre (2007) and McDonald and Nivre (2011) have shown that history-based features enhance transition-based parsers as long as they do not suffer from error propagation. However, Nivre (2006) has also shown that lookahead features are absolutely crucial given the greedy left-to-right parsing strategy.

In the model architectures considered here, the backward LSTM provides an improved lookahead. Similarly to the lookahead in statistical parsing, it is sequential. The difference is that it gives information about upcoming words with unbounded length. The forward LSTM in this model architecture provides history-based information but unlike in statistical parsing, that information is built sequentially rather than hierarchically: the forward LSTM passes through the sentence in the linear order of the sentence. In our results, we see that lookahead features are more important than the history-based ones. It hurts parsing accuracy more to ablate the backward LSTM than to ablate the forward one. This is expected given that some history-based information is still available through the top tokens on the stack, while the lookahead information is almost lost completely without the backward LSTM.

A composition function gives hierarchical information about the history of parsing actions. It makes sense that it helps the backward LSTM model most since that model has no access to any information about parsing history. It helps the forward LSTM slightly which indicates that there can be gains from using structured information about parsing history rather than sequential information. We could then expect that composition should help the BiLSTM model which, how-

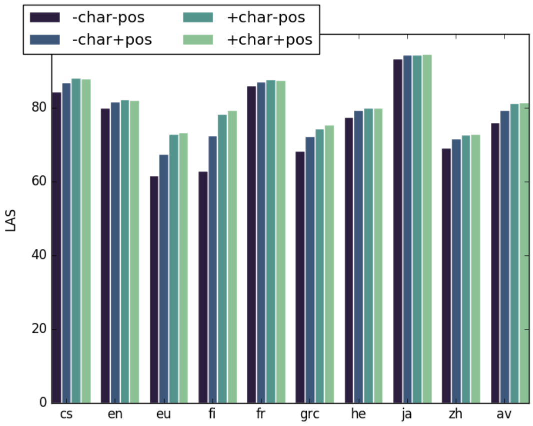

Figure 5: LAS of baseline, using char and/or POS tags to construct word representations

ever, is not the case. This might be because the BiLSTM constructs information about parsing history and lookahead into a unique representation. In any case, this indicates that BiLSTMs are powerful feature extractors which seem to capture useful information about subtrees.

5.2 Ablating POS and character information

Next, we look at the effect of the different word representation methods on the different languages, as represented in Figure 5. As is consistent with the literature (Ballesteros et al., 2015; de Lhoneux et al., 2017a; Smith et al., 2018b), using character-based word representations and/or POS tags consistently improves parsing accuracy but has a different impact in different languages and the benefits of both methods are not cumulative: using the two combined is not much better than using either on its own. In particular, character models are an efficient way to obtain large improvements in morphologically rich languages.

We look at the impact of recursive compositions on all combinations of ablated models, see Table 1. We only look at the impact of using an LSTM cell rather than a recurrent cell since it was a better technique across the board (see previous section).

Looking first at BiLSTMs, it seems that composition does not reliably help parsing accuracy, regardless of access to POS and character information. This indicates that the vectors obtained from the BiLSTM already contain information that would otherwise be obtained by using composition.

Turning to results with either the forward or the backward LSTM ablated, we see the expected pattern. Composition helps more when the model lacks POS tags, indicating that there is some redundancy between these two methods of building contextual information. Composition helps recover a substantial part of the gap of the model with a backward LSTM with or without POS tag. It recovers a much less substantial part of the gap in other cases which means that, although there is some redundancy between these different methods of building contextual information, they are still complementary and a recursive composition function cannot fully compensate for the lack of a backward LSTM or POS and/or character information. There are some language idiosyncracies in the results. While composition helps recover most of the gap for the backward LSTM models without POS and/or character information for Czech and English, it does it to a much smaller extent for Basque and Finnish. We hypothesise that arc depth might impact the usefulness of composition, since more depth means more matrix multiplications with the composition function. However, we find no correlation between average arc depth of the treebanks and usefulness of composition. It is an open question why composition helps some languages more than others.

Note that we are not the first to use composition over vectors obtained from a BiLSTM in the context of dependency parsing, as this was done by Qi and Manning (2017). The difference is that they compose vectors before scoring transitions. It was also done by Kiperwasser and Goldberg (2016a) who showed that using BiLSTM vectors for words in their Tree LSTM parser is helpful but they did not compare this to using BiLSTM vectors without the Tree LSTM.

Recurrent and recursive LSTMs in the way they have been considered in this paper are two ways of constructing contextual information and making it available for local decisions in a greedy parser. The strength of recursive LSTMs is that they can build this contextual information using hierarchical context rather than linear context. A possible weakness is that this makes the model sensitive to error propagation: a wrong attachment leads to using the wrong contextual information. It is therefore possible that the benefits and drawbacks of using this method cancel each other out in the context of BiLSTMs.

	pos+char+						pos+char-					
	bi	bi+lc	bw	bw+lc	fw	fw+lc	bi	bi+lc	bw	bw+lc	fw	fw+lc
cs	87.9	88.2	85.9	**87.7**	84.9	85.0	86.7	87.0	84.5	**86.2**	83.6	83.6
en	82.0	82.3	80.3	**81.9**	75.1	**75.6**	81.5	81.5	79.7	**81.4**	74.3	**75.0**
eu	73.3	73.5	72.0	72.4	66.8	**67.4**	67.4	67.6	65.6	**66.3**	59.6	**60.5**
fi	79.3	79.7	77.7	**79.2**	73.7	**74.7**	72.5	72.7	69.8	**71.7**	66.7	**67.4**
fr	87.5	87.6	86.4	**87.5**	86.3	86.4	87.1	87.2	85.8	**86.9**	85.7	85.9
grc	75.4	**76.1**	72.8	**75.0**	70.9	71.1	72.2	72.5	69.6	**71.4**	67.4	67.8
he	80.0	80.1	78.0	**80.0**	77.9	78.2	79.4	79.2	77.2	**79.0**	76.9	77.3
ja	94.6	94.6	94.4	94.5	83.3	**83.9**	94.3	94.3	94.2	94.3	83.0	**83.6**
zh	72.9	72.7	71.3	**72.4**	57.4	**58.7**	71.5	71.3	69.9	**70.8**	56.4	**57.9**
av	81.4	81.6	79.8	**81.2**	75.1	**75.7**	79.2	79.2	77.4	**78.7**	72.6	**73.2**

	pos-char+						pos-char-					
	bi	bi+lc	bw	bw+lc	fw	fw+lc	bi	bi+lc	bw	bw+lc	fw	fw+lc
cs	88.1	88.4	86.0	**87.8**	84.7	84.9	84.3	84.5	81.3	**83.1**	79.9	79.8
en	82.2	82.1	79.8	**81.6**	73.2	**73.8**	80.0	79.9	77.5	**79.2**	70.5	**71.5**
eu	72.8	72.9	71.5	71.8	65.4	**66.4**	61.6	62.0	57.7	**59.5**	48.7	**51.2**
fi	78.2	78.6	75.8	**77.9**	72.0	**73.0**	62.8	63.1	56.6	**60.2**	52.8	**54.7**
fr	87.6	87.7	86.1	**87.4**	85.4	85.7	85.9	85.8	83.7	**85.3**	83.1	83.3
grc	74.4	74.8	71.3	**73.7**	69.2	69.6	68.3	**69.0**	64.6	**67.3**	62.6	**63.4**
he	79.9	80.1	77.4	**79.9**	76.5	**77.3**	77.5	77.4	74.4	**77.2**	74.2	74.7
ja	94.2	94.4	94.2	94.4	81.3	**81.8**	93.2	93.3	92.7	93.1	79.5	**80.2**
zh	72.7	72.5	70.8	**72.2**	56.5	**58.2**	69.1	69.3	66.7	**68.1**	53.4	**55.0**
av	81.1	81.3	79.2	**80.8**	73.8	**74.5**	75.9	76.0	72.8	**74.8**	67.2	**68.2**

Table 1: LAS for bi, bw and fw, without and with composition ($+lc$) with an LSTM. Difference > 0.5 with $+lc$ in bold.

	bi	full	-bi	-[bi+lc]	-bw	-[bw+lc]	-fw	-[fw+lc]
cs	90.9	92.0	91.8	91.8	91.8	91.8	92.1	92.0
en	85.8	87.1	86.7	86.7	86.8	86.7	87.2	87.2
eu	78.7	80.9	80.3	80.2	80.4	80.3	80.9	81.0
fi	83.5	85.5	85.4	85.4	85.3	85.2	85.6	85.5
fr	89.8	90.8	90.8	90.6	90.8	90.7	90.8	90.8
grc	81.2	83.5	83.0	83.1	83.3	83.0	83.4	83.6
he	86.2	87.6	87.6	87.4	87.5	87.2	87.6	87.7
ja	95.9	96.1	95.8	95.7	95.9	95.8	96.3	96.2
zh	78.3	79.3	78.4	78.6	78.4	78.7	79.8	79.9
av	85.6	87.0	86.6	86.6	86.7	86.6	87.1	87.1

Table 2: UAS ensemble (full) and ablated experiments.

5.3 Ensemble

To investigate further the information captured by BiLSTMs, we ensemble the 6 versions of the models with POS and character information with the different feature extractors (bi, bw, fw) with ($+lc$) and without composition. We use the (unweighted) reparsing technique of Sagae and Lavie (2006)[6] and ignoring labels. As can be seen from the UAS scores in Table 2, the ensemble (*full*) largely outperforms the parser using only a BiLSTM, indicating that the information obtained from the different models is complementary. To investigate the contribution of each of the 6 models, we ablate each one by one. As can be seen from Table 2, ablating either of the BiLSTM models or the backward LSTM using composition, results in the least effective of the ablated models, further strengthening the conclusion that BiLSTMs are powerful feature extractors.

6 Conclusion

We investigated the impact of composing the representation of subtrees in a transition-based parser. We observed that composition does not reliably help a parser that uses a BiLSTM for feature extraction, indicating that vectors obtained from the BiLSTM might capture subtree information, which is consistent with the results of Linzen et al. (2016). However, we observe that, when ablating the backward LSTM, performance drops and recursive composition does not help to recover much of this gap. We hypothesise that this is because the backward LSTM primarily improves the lookahead for the greedy parser. When ablating the forward LSTM, performance drops to a smaller extent and recursive composition recovers a substantial part of the gap. This indicates that a forward LSTM and a recursive composition function capture similar information, which we take to be related to the rich history-based features crucial for a transition-based parser. To capture this infor-

[6] This method scores all arcs by the number of parsers predicting them and extracts a maximum spanning tree using the Chu-Liu-Edmonds algorithm (Edmonds, 1967).

mation, a recursive composition function is better than a forward LSTM on its own, but it is even better to have a forward LSTM as part of a BiLSTM. We further find that recursive composition helps more when POS tags are ablated from the model, indicating that POS tags and a recursive composition function are partly redundant ways of constructing contextual information. Finally, we correlate results with language properties, showing that the improved lookahead of a backward LSTM is especially important for head-final languages.

Acknowledgments

We acknowledge the computational resources provided by CSC in Helsinki and Sigma2 in Oslo through NeIC-NLPL (www.nlpl.eu). We thank Sara Stymne and Aaron Smith for many discussions about this paper.

References

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 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 349–359.

Timothy Dozat and Christopher Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 5th International Conference on Learning Representations.*

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 (ACL)*, pages 334–343.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of NAACL-HLT*, pages 199–209.

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

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. *arXiv preprint arXiv:1803.11138.*

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

Eliyahu Kiperwasser and Yoav Goldberg. 2016a. Easy-first dependency parsing with hierarchical tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461.

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

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 (ACL)*, pages 673–682.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258. Association for Computational Linguistics.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1426–1436.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017a. From raw text to universal dependencies - look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217, Vancouver, Canada. Association for Computational Linguistics.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017b. Arc-hybrid non-projective dependency parsing with a static-dynamic oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104, Pisa, Italy. Association for Computational Linguistics.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017c. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *Proceedings of the 15th Treebanks and Linguistic Theories Workshop (TLT).*

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford Dependencies: A cross-linguistic typology. In *Proceedings of the 9th International*

Conference on Language Resources and Evaluation (LREC), pages 4585–4592.

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.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, pages 197–230.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer.

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

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 (ACL-IJCNLP)*, pages 351–359.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Silvie Cinková, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Radu Ion, Elena Irimia, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shinsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Robert Östling, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jonathan North Washington, Mats Wirén, Tak-sum Wong, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. 2017. Universal dependencies 2.1. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–117. Association for Computational Linguistics.

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.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018a. 82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual*

Parsing from Raw Text to Universal Dependencies, pages 113–123. Association for Computational Linguistics.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018b. An investigation of the interactions between pre-trained word embeddings, character models and pos tags in dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720. Association for Computational Linguistics.

Richard Socher, John Bauer, Christopher D Manning, et al. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 455–465.

Pontus Stenetorp. 2013. Transition-based dependency parsing using recursive neural networks. In *Deep Learning Workshop at the 2013 Conference on Neural Information Processing Systems (NIPS)*, Lake Tahoe, Nevada, USA.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia. European Language Resources Association.

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*.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gökırmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Hěctor Martínez Alonso, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadova, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics.