Improved Transition-Based Parsing by Modeling Characters instead of Words with LSTMs

Miguel Ballesteros[◇♠] **Chris Dyer**[♣♠] **Noah A. Smith**[♡]
[◇]NLP Group, Pompeu Fabra University, Barcelona, Spain
[♠]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[♣]Marianas Labs, Pittsburgh, PA, USA
[♡]Computer Science & Engineering, University of Washington, Seattle, WA, USA
miguel.ballesteros@upf.edu, chris@marianaslabs.com, nasmith@cs.washington.edu

Abstract

We present extensions to a continuous-state dependency parsing method that makes it applicable to morphologically rich languages. Starting with a high-performance transition-based parser that uses long short-term memory (LSTM) recurrent neural networks to learn representations of the parser state, we replace lookup-based word representations with representations constructed from the orthographic representations of the words, also using LSTMs. This allows statistical sharing across word forms that are similar on the surface. Experiments for morphologically rich languages show that the parsing model benefits from incorporating the character-based encodings of words.

1 Introduction

At the heart of natural language parsing is the challenge of representing the "state" of an algorithm—what parts of a parse have been built and what parts of the input string are not yet accounted for—as it incrementally constructs a parse. Traditional approaches rely on independence assumptions, decomposition of scoring functions, and/or greedy approximations to keep this space manageable. **Continuous-state** parsers have been proposed, in which the state is embedded as a vector (Titov and Henderson, 2007; Stenetorp, 2013; Chen and Manning, 2014; Dyer et al., 2015; Zhou et al., 2015; Weiss et al., 2015). Dyer et al. reported state-of-the-art performance on English and Chinese benchmarks using a transition-based parser whose continuous-state embeddings were constructed using LSTM recurrent neural networks (RNNs) whose parameters were estimated to maximize the probability of a gold-standard sequence of parse actions.

The primary contribution made in this work is to take the idea of continuous-state parsing a step further by making the word embeddings that are used to construct the parse state sensitive to the morphology of the words.[1] Since it it is well known that a word's form often provides strong evidence regarding its grammatical role in morphologically rich languages (Ballesteros, 2013, *inter alia*), this has promise to improve accuracy and statistical efficiency relative to traditional approaches that treat each word type as opaque and independently modeled. In the traditional parameterization, words with similar grammatical roles will only be embedded near each other if they are observed in similar contexts with sufficient frequency. Our approach reparameterizes word embeddings using the same RNN machinery used in the parser: a word's vector is calculated based on the sequence of orthographic symbols representing it (§3).

Although our model is provided no supervision in the form of explicit morphological annotation, we find that it gives a large performance increase when parsing morphologically rich languages in the SPMRL datasets (Seddah et al., 2013; Seddah and Tsarfaty, 2014), especially in agglutinative languages and the ones that present extensive case systems (§4). In languages that show little morphology, performance remains good, showing that the RNN composition strategy is capable of capturing both morphological regularities and arbitrariness in the sense of Saussure (1916). Finally, a particularly noteworthy result is that we find that character-based word embeddings in some cases obviate explicit POS information, which is usually found to be indispensable for accurate parsing.

A secondary contribution of this work is to show that the continuous-state parser of Dyer et al. (2015) can learn to generate nonprojective trees. We do this by augmenting its transition operations

[1]Software for replicating the experiments is available from `https://github.com/clab/lstm-parser`.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 349–359,
Lisbon, Portugal, 17-21 September 2015.

with a SWAP operation (Nivre, 2009) (§2.4), enabling the parser to produce nonprojective dependencies which are often found in morphologically rich languages.

2 An LSTM Dependency Parser

We begin by reviewing the parsing approach of Dyer et al. (2015) on which our work is based.

Like most transition-based parsers, Dyer et al.'s parser can be understood as the sequential manipulation of three data structures: a buffer B initialized with the sequence of words to be parsed, a stack S containing partially-built parses, and a list A of actions previously taken by the parser. In particular, the parser implements the arc-standard parsing algorithm (Nivre, 2004).

At each time step t, a transition action is applied that alters these data structures by pushing or popping words from the stack and the buffer; the operations are listed in Figure 1.

Along with the discrete transitions above, the parser calculates a vector representation of the states of B, S, and A; at time step t these are denoted by $\mathbf{b}_t$, $\mathbf{s}_t$, and $\mathbf{a}_t$, respectively. The total parser state at t is given by

$$\mathbf{p}_t = \max\{\mathbf{0}, \mathbf{W}[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{d}\} \quad (1)$$

where the matrix $\mathbf{W}$ and the vector $\mathbf{d}$ are learned parameters. This continuous-state representation $\mathbf{p}_t$ is used to decide which operation to apply next, updating B, S, and A (Figure 1).

We elaborate on the design of $\mathbf{b}_t$, $\mathbf{s}_t$, and $\mathbf{a}_t$ using RNNs in §2.1, on the representation of partial parses in S in §2.2, and on the parser's decision mechanism in §2.3. We discuss the inclusion of SWAP in §2.4.

2.1 Stack LSTMs

RNNs are functions that read a sequence of vectors incrementally; at time step t the vector $\mathbf{x}_t$ is read in and the hidden state $\mathbf{h}_t$ computed using $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$. In principle, this allows retaining information from time steps in the distant past, but the nonlinear "squashing" functions applied in the calcluation of each $\mathbf{h}_t$ result in a decay of the error signal used in training with backpropagation. LSTMs are a variant of RNNs designed to cope with this "vanishing gradient" problem using an extra memory "cell" (Hochreiter and Schmidhuber, 1997; Graves, 2013).

Past work explains the computation within an LSTM through the metaphors of deciding how much of the current input to pass into memory ($\mathbf{i}_t$) or forget ($\mathbf{f}_t$). We refer interested readers to the original papers and present only the recursive equations updating the memory cell $\mathbf{c}_t$ and hidden state $\mathbf{h}_t$ given $\mathbf{x}_t$, the previous hidden state $\mathbf{h}_{t-1}$, and the memory cell $\mathbf{c}_{t-1}$:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \mathbf{1} - \mathbf{i}_t \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where σ is the component-wise logistic sigmoid function and $\odot$ is the component-wise (Hadamard) product. Parameters are all represented using $\mathbf{W}$ and $\mathbf{b}$. This formulation differs slightly from the classic LSTM formulation in that it makes use of "peephole connections" (Gers et al., 2002) and defines the forget gate so that it sums with the input gate to $\mathbf{1}$ (Greff et al., 2015). To improve the representational capacity of LSTMs (and RNNs generally), they can be stacked in "layers." In these architectures, the input LSTM at higher layers at time t is the value of $\mathbf{h}_t$ computed by the lower layer (and $\mathbf{x}_t$ is the input at the lowest layer).

The **stack LSTM** augments the left-to-right sequential model of the conventional LSTM with a stack pointer. As in the LSTM, new inputs are added in the right-most position, but the stack pointer indicates which LSTM cell provides $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$ for the computation of the next iterate. Further, the stack LSTM provides a pop operation that moves the stack pointer to the previous element. Hence each of the parser data structures (B, S, and A) is implemented with its own stack LSTM, each with its own parameters. The values of $\mathbf{b}_t$, $\mathbf{s}_t$, and $\mathbf{a}_t$ are the $\mathbf{h}_t$ vectors from their respective stack LSTMs.

2.2 Composition Functions

Whenever a REDUCE operation is selected, two tree fragments are popped off of S and combined to form a new tree fragment, which is then popped back onto S (see Figure 1). This tree must be embedded as an input vector $\mathbf{x}_t$.

To do this, Dyer et al. (2015) use a recursive neural network g_r (for relation r) that composes

$\textbf{Stack}_t$	$\textbf{Buffer}_t$	Action	$\textbf{Stack}_{t+1}$	$\textbf{Buffer}_{t+1}$	Dependency
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-RIGHT(r)	$(g_r(\mathbf{u}, \mathbf{v}), u), S$	B	$u \xrightarrow{r} v$
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-LEFT(r)	$(g_r(\mathbf{v}, \mathbf{u}), v), S$	B	$u \xleftarrow{r} v$
S	$(\mathbf{u}, u), B$	SHIFT	$(\mathbf{u}, u), S$	B	—
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	SWAP	$(\mathbf{u}, u), S$	$(\mathbf{v}, v), B$	—

Figure 1: Parser transitions indicating the action applied to the stack and buffer and the resulting stack and buffer states. Bold symbols indicate (learned) embeddings of words and relations, script symbols indicate the corresponding words and relations. Dyer et al. (2015) used the SHIFT and REDUCE operations in their continuous-state parser; we add SWAP.

the representations of the two subtrees popped from S (we denote these by $\mathbf{u}$ and $\mathbf{v}$), resulting in a new vector $g_r(\mathbf{u}, \mathbf{v})$ or $g_r(\mathbf{v}, \mathbf{u})$, depending on the direction of attachment. The resulting vector embeds the tree fragment in the same space as the words and other tree fragments. This kind of composition was thoroughly explored in prior work (Socher et al., 2011; Socher et al., 2013b; Hermann and Blunsom, 2013; Socher et al., 2013a); for details, see Dyer et al. (2015).

2.3 Predicting Parser Decisions

The parser uses a probabilistic model of parser decisions at each time step t. Letting $\mathcal{A}(S, B)$ denote the set of allowed transitions given the stack S and buffer S (i.e., those where preconditions are met; see Figure 1), the probability of action $z \in \mathcal{A}(S, B)$ defined using a log-linear distribution:

$$p(z \mid \mathbf{p}_t) = \frac{\exp\left(\mathbf{g}_z^\top \mathbf{p}_t + q_z\right)}{\sum_{z' \in \mathcal{A}(S,B)} \exp\left(\mathbf{g}_{z'}^\top \mathbf{p}_t + q_{z'}\right)} \quad (2)$$

(where $\mathbf{g}_z$ and q_z are parameters associated with each action type z).

Parsing proceeds by always choosing the most probable action from $\mathcal{A}(S, B)$. The probabilistic definition allows parameter estimation for all of the parameters ($\mathbf{W}_*$, $\mathbf{b}_*$ in all three stack LSTMs, as well as $\mathbf{W}$, $\mathbf{d}$, $\mathbf{g}_*$, and q_*) by maximizing the conditional likelihood of each correct parser decisions given the state.

2.4 Adding the SWAP Operation

Dyer et al. (2015)'s parser implemented the most basic version of the arc-standard algorithm, which is capable of producing only projective parse trees. In order to deal with nonprojective trees, we also add the SWAP operation which allows nonprojective trees to be produced.

The SWAP operation, first introduced by Nivre (2009), allows a transition-based parser to produce nonprojective trees. Here, the inclusion of the SWAP operation requires breaking the linearity of the stack by removing tokens that are not at the top of the stack. This is easily handled with the stack LSTM. Figure 1 shows how the parser is capable of moving words from the stack (S) to the buffer (B), breaking the linear order of words. Since a node that is swapped may have already been assigned as the head of a dependent, the buffer (B) can now also contain tree fragments.

3 Word Representations

The main contribution of this paper is to change the word representations. In this section, we present the standard word embeddings as in Dyer et al. (2015), and the improvements we made generating word embeddings designed to capture morphology based on orthographic strings.

3.1 Baseline: Standard Word Embeddings

Dyer et al.'s parser generates a word representation for each input token by concatenating two vectors: a vector representation for each word type ($\mathbf{w}$) and a representation ($\mathbf{t}$) of the POS tag of the token (if it is used), provided as auxiliary input to the parser.[2] A linear map ($\mathbf{V}$) is applied to the resulting vector and passed through a component-wise ReLU:

$$\mathbf{x} = \max\{\mathbf{0}, \mathbf{V}[\mathbf{w}; \mathbf{t}] + \mathbf{b}\}$$

For out-of-vocabulary words, the parser uses an "UNK" token that is handled as a separate word during parsing time. This mapping can be shown schematically as in Figure 2.

[2] Dyer et al. (2015), included a third input representation learned from a neural language model ($\tilde{\mathbf{w}}_{\text{LM}}$). We do not include these pretrained representations in our experiments, focusing instead on character-based representations.

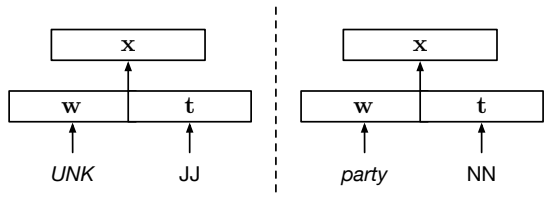

Figure 2: Baseline model word embeddings for an in-vocabulary word that is tagged with POS tag NN (right) and an out-of-vocabulary word with POS tag JJ (left).

3.2 Character-Based Embeddings of Words

Following Ling et al. (2015), we compute character-based continuous-space vector embeddings of words using bidirectional LSTMs (Graves and Schmidhuber, 2005). When the parser initiates the learning process and populates the buffer with all the words from the sentence, it reads the words character by character from left to right and computes a continuous-space vector embedding the character sequence, which is the $\mathbf{h}$ vector of the LSTM; we denote it by $\overrightarrow{\mathbf{w}}$. The same process is also applied in reverse (albeit with different parameters), computing a similar continuous-space vector embedding starting from the last character and finishing at the first ($\overleftarrow{\mathbf{w}}$); again each character is represented with an LSTM cell. After that, we concatenate these vectors and a (learned) representation of their tag to produce the representation $\mathbf{w}$. As in §3.1, a linear map ($\mathbf{V}$) is applied and passed through a component-wise ReLU.

$$\mathbf{x} = \max\left\{\mathbf{0}, \mathbf{V}[\overrightarrow{\mathbf{w}}; \overleftarrow{\mathbf{w}}; \mathbf{t}] + \mathbf{b}\right\}$$

This process is shown schematically in Figure 3.

Note that under this representation, out-of-vocabulary words are treated as bidirectional LSTM encodings and thus they will be "close" to other words that the parser has seen during training, ideally close to their more frequent, syntactically similar morphological relatives. We conjecture that this will give a clear advantage over a single "UNK" token for all the words that the parser does not see during training, as done by Dyer et al. (2015) and other parsers without additional resources. In §4 we confirm this hypothesis.

4 Experiments

We applied our parsing model and several variations of it to several parsing tasks and report results below.

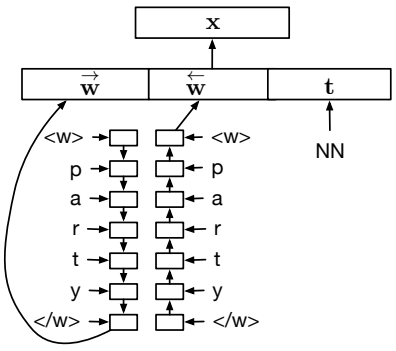

Figure 3: Character-based word embedding of the word *party*. This representation is used for both in-vocabulary and out-of-vocabulary words.

4.1 Data

In order to find out whether the character-based representations are capable of learning the morphology of words, we applied the parser to morphologically rich languages specifically the treebanks of the SPMRL shared task (Seddah et al., 2013; Seddah and Tsarfaty, 2014): Arabic (Maamouri et al., 2004), Basque (Aduriz et al., 2003), French (Abeillé et al., 2003), German (Seeker and Kuhn, 2012), Hebrew (Sima'an et al., 2001), Hungarian (Vincze et al., 2010), Korean (Choi, 2013), Polish (Świdziński and Woliński, 2010) and Swedish (Nivre et al., 2006b). For all the corpora of the SPMRL Shared Task we used predicted POS tags as provided by the shared task organizers.[3] For these datasets, evaluation is calculated using `eval07.pl`, which includes punctuation.

We also experimented with the Turkish dependency treebank[4] (Oflazer et al., 2003) of the CoNLL-X Shared Task (Buchholz and Marsi, 2006). We used gold POS tags, as is common with the CoNLL-X data sets.

To put our results in context with the most recent neural network transition-based parsers, we run the parser in the same Chinese and English

[3]The POS tags were calculated with the MarMot tagger (Müller et al., 2013) by the best performing system of the SPMRL Shared Task (Björkelund et al., 2013). Arabic: 97.38. Basque: 97.02. French: 97.61. German: 98.10. Hebrew: 97.09. Hungarian: 98.72. Korean: 94.03. Polish: 98.12. Swedish: 97.27.

[4]Since the Turkish dependency treebank does not have a development set, we extracted the last 150 sentences from the 4996 sentences of the training set as a development set.

setups as Chen and Manning (2014) and Dyer et al. (2015). For Chinese, we use the Penn Chinese Treebank 5.1 (CTB5) following Zhang and Clark (2008b),[5] with gold POS tags. For English, we used the Stanford Dependency (SD) representation of the Penn Treebank[6] (Marcus et al., 1993; Marneffe et al., 2006).[7]. Results for Turkish, Chinese, and English are calculated using the CoNLL-X `eval.pl` script, which ignores punctuation symbols.

4.2 Experimental Configurations

In order to isolate the improvements provided by the LSTM encodings of characters, we run the stack LSTM parser in the following configurations:

- **Words**: words only, as in §3.1 (but without POS tags)
- **Chars**: character-based representations of words with bidirectional LSTMs, as in §3.2 (but without POS tags)
- **Words + POS**: words and POS tags (§3.1)
- **Chars + POS**: character-based representations of words with bidirectional LSTMs plus POS tags (§3.2)

None of the experimental configurations include pretrained word-embeddings or any additional data resources. All experiments include the SWAP transition, meaning that nonprojective trees can be produced in any language.

Dimensionality. The full version of our parsing model sets dimensionalities as follows. LSTM hidden states are of size 100, and we use two layers of LSTMs for each stack. Embeddings of the parser actions used in the composition functions have 20 dimensions, and the output embedding size is 20 dimensions. The learned word representations embeddings have 32 dimensions when used, while the character-based representations have 100 dimensions, when used. Part of speech embeddings have 12 dimensions. These dimensionalities were chosen after running several tests with different values, but a more careful selection of these values would probably further improve results.

[5]Training: 001–815, 1001–1136. Development: 886–931, 1148–1151. Test: 816–885, 1137–1147.

[6]Training: 02–21. Development: 22. Test: 23.

[7]The POS tags are predicted by using the Stanford Tagger (Toutanova et al., 2003) with an accuracy of 97.3%.

4.3 Training Procedure

Parameters are initialized randomly—refer to Dyer et al. (2015) for specifics—and optimized using stochastic gradient descent (without minibatches) using derivatives of the negative log likelihood of the sequence of parsing actions computed using backpropagation. Training is stopped when the learned model's UAS stops improving on the development set, and this model is used to parse the test set. No pretraining of any parameters is done.

4.4 Results and Discussion

Tables 1 and 2 show the results of the parsers for the development sets and the final test sets, respectively. Most notable are improvements for agglutinative languages—Basque, Hungarian, Korean, and Turkish—both when POS tags are included and when they are not. Consistently, across all languages, **Chars** outperforms **Words**, suggesting that the character-level LSTMs are learning representations that capture similar information to parts of speech. On average, **Chars** is on par with **Words + POS**, and the best average of labeled attachment scores is achieved with **Chars + POS**.

It is common practice to encode morphological information in treebank POS tags; for instance, the Penn Treebank includes English number and tense (e.g., NNS is plural noun and VBD is verb in past tense). Even if our character-based representations are capable of encoding the same kind of information, existing POS tags suffice for high accuracy. However, the POS tags in treebanks for morphologically rich languages do not seem to be enough.

Swedish, English, and French use suffixes for the verb tenses and number,[8] while Hebrew uses prepositional particles rather than grammatical case. Tsarfaty (2006) and Cohen and Smith (2007) argued that, for Hebrew, determining the correct morphological segmentation is dependent on syntactic context. Our approach sidesteps this step, capturing the same kind of information in the vectors, and learning it from syntactic context. Even for Chinese, which is not morphologically rich, **Chars** shows a benefit over **Words**, perhaps by capturing regularities in syllable structure within words.

[8]Tense and number features provide little improvement in a transition-based parser, compared with other features such as case, when the POS tags are included (Ballesteros, 2013).

UAS

Language	Words	Chars	Words + POS	Chars + POS
Arabic	86.14	**87.20**	**87.44**	87.07
Basque	78.42	**84.97**	83.49	**85.58**
French	84.84	**86.21**	**87.00**	86.33
German	88.14	**90.94**	91.16	**91.23**
Hebrew	79.73	**79.92**	**81.99**	80.76
Hungarian	72.38	**80.16**	78.47	**80.85**
Korean	78.98	**88.98**	87.36	**89.14**
Polish	73.29	**85.69**	**89.32**	88.54
Swedish	73.44	**75.03**	**80.02**	78.85
Turkish	71.10	**74.91**	77.13	**77.96**
Chinese	79.43	**80.36**	**85.98**	85.81
English	91.64	**91.98**	**92.94**	92.49
Average	79.79	**83.86**	85.19	**85.38**

LAS

Language	Words	Chars	Words + POS	Chars + POS
Arabic	82.73	**84.34**	**84.81**	84.36
Basque	67.08	**78.22**	74.31	**79.52**
French	80.32	**81.70**	**82.71**	81.51
German	85.36	**88.68**	**89.04**	88.83
Hebrew	69.42	**70.58**	**74.11**	72.18
Hungarian	62.14	**75.61**	69.50	**76.16**
Korean	67.48	**86.80**	83.80	**86.88**
Polish	65.13	**78.23**	**81.84**	80.97
Swedish	64.77	**66.74**	**72.09**	69.88
Turkish	53.98	**62.91**	62.30	**62.87**
Chinese	75.64	**77.06**	**84.36**	84.10
English	88.60	**89.58**	**90.63**	90.08
Average	71.89	**78.37**	79.13	**79.78**

Table 1: Unlabeled attachment scores (left) and labeled attachment scores (right) on the **development** sets (not a standard development set for Turkish). In each table, the first two columns show the results of the parser with word lookup (**Words**) vs. character-based (**Chars**) representations. The last two columns add POS tags. Boldface shows the better result comparing **Words** vs. **Chars** and comparing **Words + POS** vs. **Chars + POS**.

UAS

Language	Words	Chars	Words + POS	Chars + POS
Arabic	85.21	**86.08**	86.05	**86.07**
Basque	77.06	**85.19**	82.92	**85.22**
French	83.74	**85.34**	**86.15**	85.78
German	82.75	**86.80**	**87.33**	87.26
Hebrew	77.62	**79.93**	**80.68**	80.17
Hungarian	72.78	**80.35**	78.64	**80.92**
Korean	78.70	**88.39**	86.85	**88.30**
Polish	72.01	**83.44**	**87.06**	85.97
Swedish	76.39	**79.18**	**83.43**	83.24
Turkish	71.70	**76.32**	75.32	**76.34**
Chinese	79.01	**79.94**	**85.96**	85.30
English	91.16	**91.47**	**92.57**	91.63
Average	79.01	**85.36**	84.41	**84.68**

LAS

Language	Words	Chars	Words + POS	Chars + POS
Arabic	82.05	**83.41**	**83.46**	83.40
Basque	66.61	**79.09**	73.56	**78.61**
French	79.22	**80.92**	**82.03**	81.08
German	79.15	**84.04**	**84.62**	84.49
Hebrew	68.71	**71.26**	**72.70**	72.26
Hungarian	61.93	**75.19**	69.31	**76.34**
Korean	67.50	**86.27**	83.37	**86.21**
Polish	63.96	**76.84**	**79.83**	78.24
Swedish	67.69	**71.19**	**76.40**	74.47
Turkish	54.55	**64.34**	61.22	**62.28**
Chinese	74.79	**76.29**	**84.40**	83.72
English	88.42	**88.94**	**90.31**	89.44
Average	71.22	**78.15**	78.43	**79.21**

Table 2: Unlabeled attachment scores (left) and labeled attachment scores (right) on the **test** sets. In each table, the first two columns show the results of the parser with word lookup (**Words**) vs. character-based (**Chars**) representations. The last two columns add POS tags. Boldface shows the better result comparing **Words** vs. **Chars** and comparing **Words + POS** vs. **Chars + POS**.

4.4.1 Learned Word Representations

Figure 4 visualizes a sample of the character-based bidirectional LSTMs's learned representations (**Chars**). Clear clusters of past tense verbs, gerunds, and other syntactic classes are visible. The colors in the figure represent the most common POS tag for each word.

4.4.2 Out-of-Vocabulary Words

The character-based representation for words is notably beneficial for out-of-vocabulary (OOV) words. We tested this specifically by comparing **Chars** to a model in which all OOVs are replaced by the string "UNK" during parsing. This always has a negative effect on LAS (average -4.5 points, -2.8 UAS). Figure 5 shows how this drop varies with the development OOV rate across treebanks; most extreme is Korean, which drops 15.5 LAS. A similar, but less pronounced pattern, was observed for models that include POS.

Interestingly, this artificially impoverished model is still consistently better than **Words** for all languages (e.g., for Korean, by 4 LAS). This implies that not all of the improvement is due to OOV words; statistical sharing across orthographically close words is beneficial, as well.

4.4.3 Computational Requirements

The character-based representations make the parser slower, since they require composing the character-based bidirectional LSTMs for each

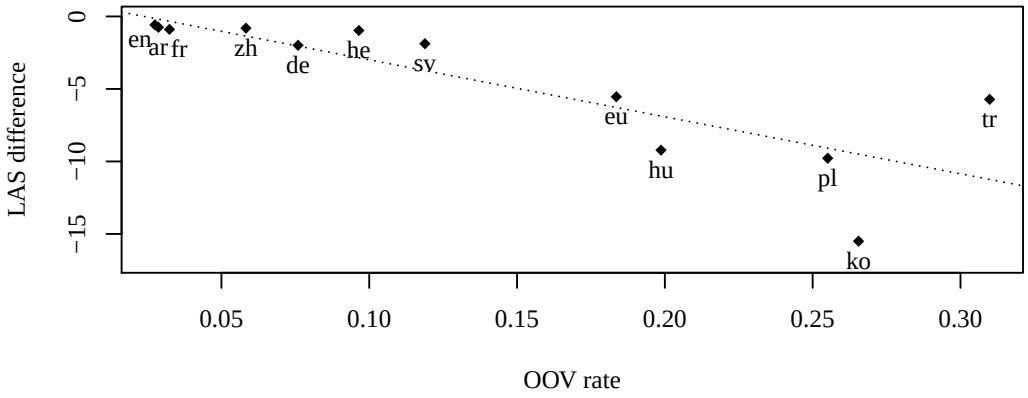

Figure 5: On the x-axis is the OOV rate in development data, by treebank; on the y-axis is the difference in development-set LAS between **Chars** model as described in §3.2 and one in which all OOV words are given a single representation.

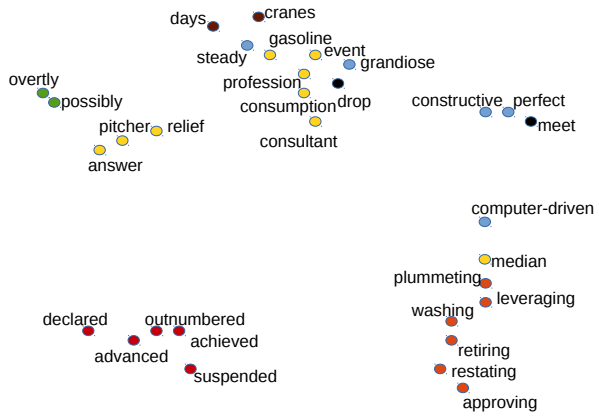

Figure 4: Character-based word representations of 30 random words from the English development set (**Chars**). Dots in red represent past tense verbs; dots in orange represent gerund verbs; dots in black represent present tense verbs; dots in blue represent adjectives; dots in green represent adverbs; dots in yellow represent singular nouns; dots in brown represent plural nouns. The visualization was produced using t-SNE; see `http://lvdmaaten.github.io/tsne/`.

word of the input sentence; however, at test time these results could be cached. On average, **Words** parses a sentence in 44 ms, while**Chars** needs 130 ms.[9] Training time is affected by the same constant, needing some hours to have a competitive model. In terms of memory, **Words** requires on average 300 MB of main memory for both training and parsing, while **Chars** requires 450 MB.

4.4.4 Comparison with State-of-the-Art

Table 3 shows a comparison with state-of-the-art parsers. We include greedy transition-based parsers that, like ours, do not apply a beam search (Zhang and Clark, 2008b) or a dynamic oracle (Goldberg and Nivre, 2013). For all the SPMRL languages we show the results of Ballesteros (2013), who reported results after carrying out a careful automatic morphological feature selection experiment. For Turkish, we show the results of Nivre et al. (2006a) which also carried out a careful manual morphological feature selection. Our parser outperforms these in most cases. Since those systems rely on morphological features, we believe that this comparison shows even more that the character-based representations are capturing morphological information, though without explicit morphological features. For English and Chinese, we report (Dyer et al., 2015) which is **Words + POS** but with pretrained word embeddings.

We also show the best reported results on these datasets. For the SPMRL data sets, the best performing system of the shared task is either Björkelund et al. (2013) or Björkelund et al. (2014), which are consistently better than our sys-

[9] We are using a machine with 32 Intel Xeon CPU E5-2650 at 2.00GHz; the parser runs on a single core.

	This Work			Best Greedy Result			Best Published Result		
Language	**UAS**	**LAS**	**System**	**UAS**	**LAS**	**System**	**UAS**	**LAS**	**System**
Arabic	86.08	83.41	**Chars**	84.57	81.90	B'13	88.32	86.21	B+'13
Basque	85.22	78.61	**Chars + POS**	84.33	78.58	B'13	89.96	85.70	B+'14
French	86.15	82.03	**Words + POS**	83.35	77.98	B'13	89.02	85.66	B+'14
German	87.33	84.62	**Words + POS**	85.38	82.75	B'13	91.64	89.65	B+'13
Hebrew	80.68	72.70	**Words + POS**	79.89	73.01	B'13	87.41	81.65	B+'14
Hungarian	80.92	76.34	**Chars + POS**	83.71	79.63	B'13	89.81	86.13	B+'13
Korean	88.39	86.27	**Chars**	85.72	82.06	B'13	89.10	87.27	B+'14
Polish	87.06	79.83	**Words + POS**	85.80	79.89	B'13	91.75	87.07	B+'13
Swedish	83.43	76.40	**Words + POS**	83.20	75.82	B'13	88.48	82.75	B+'14
Turkish	76.32	64.34	**Chars**	75.82	65.68	N+'06a	77.55	n/a	K+'10
Chinese	85.96	84.40	**Words + POS**	87.20	85.70	D+'15	87.20	85.70	D+'15
English	92.57	90.31	**Words + POS**	93.10	90.90	D+'15	94.08	92.19	W+'15

Table 3: Test-set performance of our best results (according to UAS or LAS, whichever has the larger difference), compared to state-of-the-art greedy transition-based parsers ("Best Greedy Result") and best results reported ("Best Published Result"). All of the systems we compare against use explicit morphological features and/or one of the following: pretrained word embeddings, unlabeled data and a combination of parsers; our models do not. B'13 is Ballesteros (2013); N+'06a is Nivre et al. (2006a); D+'15 is Dyer et al. (2015); B+'13 is Björkelund et al. (2013); B+'14 is Björkelund et al. (2014); K+'10 is Koo et al. (2010); W+'15 is Weiss et al. (2015).

tem for all languages. Note that the comparison is harsh to our system, which does not use unlabeled data or explicit morphological features nor any combination of different parsers. For Turkish, we report the results of Koo et al. (2010), which only reported unlabeled attachment scores. For English, we report (Weiss et al., 2015) and for Chinese, we report (Dyer et al., 2015) which is **Words + POS** but with pretrained word embeddings.

5 Related Work

Character-based representations have been explored in other NLP tasks; for instance, dos Santos and Zadrozny (2014) and dos Santos and Guimarães (2015) learned character-level neural representations for POS tagging and named entity recognition, getting a large error reduction in both tasks. Our approach is similar to theirs. Others have used character-based models as features to improve existing models. For instance, Chrupała (2014) used character-based recurrent neural networks to normalize tweets.

Botha and Blunsom (2014) show that stems, prefixes and suffixes can be used to learn useful word representations but relying on an external morphological analyzer. That is, they learn the morpheme-meaning relationship with an additive model, whereas we do not need a morphological analyzer. Similarly, Chen et al. (2015) proposed joint learning of character and word embeddings for Chinese, claiming that characters contain rich information.

Methods for joint morphological disambiguation and parsing have been widely explored Tsarfaty (2006; Cohen and Smith (2007; Goldberg and Tsarfaty (2008; Goldberg and Elhadad (2011). More recently, Bohnet et al. (2013) presented an arc-standard transition-based parser that performs competitively for joint morphological tagging and dependency parsing for richly inflected languages, such as Czech, Finnish, German, Hungarian, and Russian. Our model seeks to achieve a similar benefit to parsing without explicitly reasoning about the internal structure of words.

Zhang et al. (2013) presented efforts on Chinese parsing with characters showing that Chinese can be parsed at the character level, and that Chinese word segmentation is useful for predicting the correct POS tags (Zhang and Clark, 2008a).

To the best of our knowledge, previous work has not used character-based embeddings to improve dependency parsers, as done in this paper.

6 Conclusion

We have presented several interesting findings. First, we add new evidence that character-based representations are useful for NLP tasks. In this paper, we demonstrate that they are useful for transition-based dependency parsing, since they are capable of capturing morphological information crucial for analyzing syntax.

The improvements provided by the character-based representations using bidirectional LSTMs are strong for agglutinative languages, such as

Basque, Hungarian, Korean, and Turkish, comparing favorably to POS tags as encoded in those languages' currently available treebanks. This outcome is important, since annotating morphological information for a treebank is expensive. Our finding suggests that the best investment of annotation effort may be in dependencies, leaving morphological features to be learned implicitly from strings.

The character-based representations are also a way of overcoming the out-of-vocabulary problem; without any additional resources, they enable the parser to substantially improve the performance when OOV rates are high. We expect that, in conjunction with a pretraing regime, or in conjunction with distributional word embeddings, further improvements could be realized.

Acknowledgments

MB was supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA). This research was supported by the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-10-1-0533 and NSF IIS-1054319. This work was completed while NAS was at CMU. Thanks to Joakim Nivre, Bernd Bohnet, Fei Liu and Swabha Swayamdipta for useful comments.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In *Treebanks*. Springer.

Itziar Aduriz, María Jesús Aranzabe, Jose Mari Arriola, Aitziber Atutxa, Arantza Díaz de Ilarraza, Aitzpea Garmendia, and Maite Oronoz. 2003. Construction of a Basque dependency treebank. In *Proc of TLT*.

Miguel Ballesteros. 2013. Effective morphological feature selection with maltoptimizer at the SPMRL 2013 shared task. In *Proc. of SPMRL-EMNLP*.

Anders Björkelund, Ozlem Cetinoglu, Richárd Farkas, Thomas Mueller, and Wolfgang Seeker. 2013. (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *SPMRL-EMNLP*.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szántó. 2014. Introducing the IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 Shared Task: Reranking and Morpho-syntax meet Unlabeled Data. In *SPMRL-SANCL*.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richard Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. *TACL*, 1.

Jan A. Botha and Phil Blunsom. 2014. Compositional Morphology for Word Representations and Language Modelling. In *ICML*.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X. In *Proc of CoNLL*.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. EMNLP*.

Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. 2015. Joint learning of character and word embeddings. In *Proc. IJCAI*.

Jinho D. Choi. 2013. Preparing Korean Data for the Shared Task on Parsing Morphologically Rich Languages. *ArXiv e-prints*, September.

Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proc of ACL*.

Shay B. Cohen and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proc. EMNLP-CoNLL*.

Cicero Nogueira dos Santos and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. *Arxiv*.

Cicero dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proc of ICML-14*.

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

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. 2002. Learning precise timing with LSTM recurrent networks. *JMLR*.

Yoav Goldberg and Michael Elhadad. 2011. Joint Hebrew segmentation and parsing using a PCFG-LA lattice parser. In *Proc of ACL*.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*.

Yoav Goldberg and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proc of ACL*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6).

Alex Graves. 2013. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

Karl Moritz Hermann and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Proc. ACL*.

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

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proc of EMNLP*.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. EMNLP*.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. 2004. The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus. In *NEMLAR Conference on Arabic Language Resources and Tools*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330.

Marie-Catherine De Marneffe, Bill Maccartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proc of LREC*.

Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proc of EMNLP*.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006a. Labeled pseudo-projective dependency parsing with support vector machines. In *Proc of CoNLL*.

Joakim Nivre, Jens Nilsson, and Johan Hall. 2006b. Talbanken05: A Swedish treebank with phrase structure and dependency annotation. In *Proc of LREC*, Genoa, Italy.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proc of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proc of ACL*.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish treebank. In *Treebanks*, pages 261–277. Springer.

Ferdinand Saussure. 1916. Nature of the linguistic sign. In *Course in General Linguistics*.

Djamé Seddah and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. *SPMRL-SANCL 2014*.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 shared task: cross-framework evaluation of parsing morphologically rich languages. In *SPMRL-EMNLP 2013*.

Wolfgang Seeker and Jonas Kuhn. 2012. Making Ellipses Explicit in Dependency Conversion for a German Treebank. In *Proc of LREC*.

Khalil Sima'an, Alon Itai, Yoad Winter, Alon Altman, and Noa Nativ. 2001. Building a Tree-Bank for Modern Hebrew Text. In *Traitement Automatique des Langues*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proc of NIPS*.

Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. 2013a. Grounded compositional semantics for finding and describing images with sentences. *TACL*.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc of EMNLP*.

Pontus Stenetorp. 2013. Transition-based dependency parsing using recursive neural networks. In *Proc of NIPS Deep Learning Workshop*.

Marek Świdziński and Marcin Woliński. 2010. Towards a bank of constituent parse trees for Polish. In *Proc of TSD*.

Ivan. Titov and James. Henderson. 2007. A latent variable model for generative dependency parsing. In *Proc of IWPT*.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc of NAACL*.

Reut Tsarfaty. 2006. Integrated morphological and syntactic disambiguation for Modern Hebrew. In *Proc of ACL Student Research Workshop*.

Veronika Vincze, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin, and János Csirik. 2010. Hungarian dependency treebank. In *Proc of LREC*.

David Weiss, Christopher Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proc of ACL*.

Yue Zhang and Stephen Clark. 2008a. Joint word segmentation and POS tagging using a single perceptron. In *Proc of ACL*.

Yue Zhang and Stephen Clark. 2008b. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proc of EMNLP*.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013. Chinese parsing exploiting characters. In *Proc of ACL*.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A Neural Probabilistic Structured-Prediction Model for Transition-Based Dependency Parsing. In *Proc of ACL*.