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 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

1Software for replicating the experiments is available from https://github.com/clab/lstm-parser.
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 parser state at \( t \) noted by \( s_t \), \( A_t \), and \( h_t \), respectively. The total parser state at \( t \) is given by

\[
p_t = \max \{ 0, W[s_t; b_t; a_t] + d \}
\]

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

We elaborate on the design of \( b_t, s_t, \) and \( a_t \) 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 \( x_t \) is read in and the hidden state \( h_t \) computed using \( x_t \) and the previous hidden state \( h_{t-1} \). In principle, this allows retaining information from time steps in the distant past, but the nonlinear “squashing” functions applied in the calculation of each \( h_t \) 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 \((i_t)\) or forget \((f_t)\). We refer interested readers to the original papers and present only the recursive equations updating the memory cell \( c_t \) and hidden state \( h_t \) given \( x_t \), the previous hidden state \( h_{t-1} \), and the memory cell \( c_{t-1} \):

\[
\begin{align*}
i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \\
f_t &= 1 - i_t \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{ix}x_t + W_{ih}h_{t-1} + b_c) \\
o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) \\
h_t &= o_t \odot \tanh(c_t),
\end{align*}
\]

where \( \sigma \) is the component-wise logistic sigmoid function and \( \odot \) is the component-wise (Hadamard) product. Parameters are all represented using \( W \) and \( b \). 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 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 \( h_t \) computed by the lower layer (and \( x_t \) 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 \( c_{t-1} \) and \( h_{t-1} \) 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 \( b_t, s_t, \) and \( a_t \) are the \( h_t \) 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 \( x_t \).

To do this, Dyer et al. (2015) use a recursive neural network \( g_r \) (for relation \( r \)) that composes
the representations of the two subtrees popped from $S$ (we denote these by $u$ and $v$), resulting in a new vector $g_r(u,v)$ or $g_l(v,u)$, 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 $A(S,B)$ denote the set of allowed transitions given the stack $S$ and buffer $B$ (i.e., those where preconditions are met; see Figure 1), the probability of action $z \in A(S,B)$ defined using a log-linear distribution:

$$
p(z | \mathbf{p}_t) = \frac{\exp \left( \mathbf{g}_z^T \mathbf{p}_t + q_z \right)}{\sum_{z' \in A(S,B)} \exp \left( \mathbf{g}_{z'}^T \mathbf{p}_t + q_{z'} \right)} \quad (2)
$$

(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 $A(S,B)$. The probabilistic definition allows parameter estimation for all of the parameters ($\mathbf{W}_w$, $\mathbf{b}_w$ in all three stack LSTMs, as well as $\mathbf{W}$, $\mathbf{d}$, $\mathbf{g}_z$, and $q_z$) 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 dependant, 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. A linear map ($\mathbf{V}$) is applied to the resulting vector and passed through a component-wise ReLU:

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

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.

---

2Dyer et al. (2015), included a third input representation learned from a neural language model ($\mathbf{\hat{w}_{LM}}$). We do not include these pretrained representations in our experiments, focusing instead on character-based representations.
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 $h$ vector of the LSTM; we denote it by $\vec{w}$. 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 ($\vec{w}$); 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 $w$.

As in §3.1, a linear map ($V$) is applied and passed through a component-wise ReLU.

$$x = \max \left\{ 0, V \vec{w} + \vec{w} + b \right\}$$

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.

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 (Świderski 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 treebank4 (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

---


4Since 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), with gold POS tags. For English, we used the Stanford Dependency (SD) representation of the Penn Treebank (Marcus et al., 1993; Marneffe et al., 2006). 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.

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, 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.

---

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

---

8Tense 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).
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 Char$\text{s}$ 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 (Char$\text{s}$). 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 Char$\text{s}$ needs 130 ms. 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 Char$\text{s}$ 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 Balles$\text{t}$eros (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 Bj"orkelund et al. (2013) or Bj"orkelund et al. (2014), which are consistently better than our sys-
<table>
<thead>
<tr>
<th>Language</th>
<th>UAS</th>
<th>LAS</th>
<th>System</th>
<th>UAS</th>
<th>LAS</th>
<th>System</th>
<th>UAS</th>
<th>LAS</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>86.08</td>
<td>83.41</td>
<td>Chars</td>
<td>84.57</td>
<td>81.90</td>
<td>B’13</td>
<td>88.32</td>
<td>86.21</td>
<td>B+’13</td>
</tr>
<tr>
<td>Basque</td>
<td>85.22</td>
<td>78.61</td>
<td>Chars + POS</td>
<td>84.33</td>
<td>78.58</td>
<td>B’13</td>
<td>89.96</td>
<td>85.70</td>
<td>B+’14</td>
</tr>
<tr>
<td>French</td>
<td>86.15</td>
<td>82.03</td>
<td>Words + POS</td>
<td>83.35</td>
<td>77.98</td>
<td>B’13</td>
<td>89.02</td>
<td>85.66</td>
<td>B+’14</td>
</tr>
<tr>
<td>German</td>
<td>87.33</td>
<td>84.62</td>
<td>Words + POS</td>
<td>85.38</td>
<td>82.75</td>
<td>B’13</td>
<td>91.64</td>
<td>89.65</td>
<td>B+’13</td>
</tr>
<tr>
<td>Hebrew</td>
<td>80.68</td>
<td>72.70</td>
<td>Words + POS</td>
<td>79.89</td>
<td>73.01</td>
<td>B’13</td>
<td>87.41</td>
<td>81.65</td>
<td>B+’14</td>
</tr>
<tr>
<td>Hungarian</td>
<td>80.92</td>
<td>76.34</td>
<td>Chars + POS</td>
<td>83.71</td>
<td>79.63</td>
<td>B’13</td>
<td>89.81</td>
<td>86.13</td>
<td>B+’15</td>
</tr>
<tr>
<td>Korean</td>
<td>88.39</td>
<td>86.27</td>
<td>Chars</td>
<td>85.72</td>
<td>82.06</td>
<td>B’13</td>
<td>89.10</td>
<td>87.27</td>
<td>B+’14</td>
</tr>
<tr>
<td>Polish</td>
<td>87.06</td>
<td>79.83</td>
<td>Words + POS</td>
<td>85.80</td>
<td>79.89</td>
<td>B’13</td>
<td>91.75</td>
<td>87.07</td>
<td>B+’13</td>
</tr>
<tr>
<td>Swedish</td>
<td>83.43</td>
<td>76.40</td>
<td>Words + POS</td>
<td>83.20</td>
<td>75.82</td>
<td>B’13</td>
<td>88.48</td>
<td>82.75</td>
<td>B+’14</td>
</tr>
<tr>
<td>Turkish</td>
<td>76.32</td>
<td>64.34</td>
<td>Chars</td>
<td>75.82</td>
<td>65.68</td>
<td>N+’06a</td>
<td>77.55</td>
<td>n/a</td>
<td>K+’10</td>
</tr>
<tr>
<td>Chinese</td>
<td>85.96</td>
<td>84.40</td>
<td>Words + POS</td>
<td>87.20</td>
<td>85.70</td>
<td>D+’15</td>
<td>87.20</td>
<td>85.70</td>
<td>D+’15</td>
</tr>
<tr>
<td>English</td>
<td>92.57</td>
<td>90.31</td>
<td>Words + POS</td>
<td>93.10</td>
<td>90.90</td>
<td>D+’15</td>
<td>94.08</td>
<td>92.19</td>
<td>W+’15</td>
</tr>
</tbody>
</table>

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).

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 pretraining 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**


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


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.


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.


358


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.