Greedy Transition-Based Dependency Parsing with Stack LSTMs

Miguel Ballesteros*[1]
IBM T. J. Watson Research Center

Chris Dyer**
Carnegie Mellon University

Yoav Goldberg†
Bar-Ilan University

Noah A. Smith‡
University of Washington

We introduce a greedy transition-based parser that learns to represent parser states using recurrent neural networks. Our primary innovation that enables us to do this efficiently is a new control structure for sequential neural networks—the stack long short-term memory unit (LSTM). Like the conventional stack data structures used in transition-based parsers, elements can be pushed to or popped from the top of the stack in constant time, but, in addition, an LSTM maintains a continuous space embedding of the stack contents. Our model captures three facets of the parser's state: (i) unbounded look-ahead into the buffer of incoming words, (ii) the complete history of transition actions taken by the parser, and (iii) the complete contents of the stack of partially built tree fragments, including their internal structures. In addition, we compare two different word representations: (i) standard word vectors based on look-up tables and (ii) character-based models of words. Although standard word embedding models work well in all languages, the character-based models improve the handling of out-of-vocabulary words, particularly in morphologically rich languages. Finally, we discuss the use of dynamic oracles in training the parser. During training, dynamic oracles alternate between sampling parser states from the training data and from the model as it is being learned, making the model more robust to the kinds of errors that will be made at test time. Training our model with dynamic oracles yields a linear-time greedy parser with very competitive performance.

* IBM T. J. Watson Research Center, 1101 Kitchawan Road, Route 134, Yorktown Heights, NY 10598. E-mail: miguel.ballesteros@upf.edu.
** Language Technologies Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA. E-mail: cdyer@cs.cmu.edu. Now affiliated with DeepMind, London, UK.
† Computer Science Department, Bar-Ilan University, Ramat Gan, 5290002 Israel. E-mail: yoav.goldberg@gmail.com.
‡ Computer Science & Engineering, University of Washington, Box 352350, Seattle, WA 98195, USA. E-mail: nasmith@cs.washington.edu.
1 This work was done while Miguel was at Universitat Pompeu Fabra and Carnegie Mellon University.

Submission received: 16 September 2015; revised version received: 6 April 2016; accepted for publication: 14 June 2016.

doi:10.1162/COLI_a_00285

1. Introduction

Natural language parsing can be formulated as a series of decisions that read words in sequence and incrementally combine them to form syntactic structures; this formulation is known as transition-based parsing, and is often coupled with a greedy inference procedure (Yamada and Matsumoto 2003; Nivre 2003, 2004, 2008). Greedy transition-based parsing is attractive because the number of operations required to build any projective parse tree is linear in the length of the sentence, making greedy versions of transition-based parsing computationally efficient relative to graph- and grammar-based alternative formalisms, which usually require solving superlinear search problems. The challenge in transition-based parsing is modeling which action should be taken in each of the states encountered as the parsing algorithm progresses.[2]

Because the parser state involves the complete sentence—which is unbounded in length—the representational challenge faced by the modeler is to find a finite encoding of an infinite space. This challenge is usually dealt with by making assumptions about state equivalence by selecting features of the state believed by the model-builder to be most informative (e.g., the next three words in the sequence, the roots of the two most recently built subtrees and their left- and right-most children, and so on).[3]

In this article, we advocate a recursive approach, in which complex parser states are represented as the composition of simpler ones, and these are constructed incrementally as parsing proceeds. Thus, we operate in a paradigm in which the model builder designs recursive architectures that automatically discover effective views of the complete parser's evolving state rather than explicitly engineering which features of the state to focus on. This construction enables us to avoid making explicit state-equivalence assumptions. We present one such architecture based on advances in recursively defined neural networks. Our state representation integrates information from a wide range of sources:

- the complete remaining unprocessed words in the input buffer,
- the complete history of parser transition actions on this sequence, and
- the complete contents of the stack of partially constructed syntactic structures.

This global sensitivity of the state representation contrasts with most previous work in transition-based parsing that considers only a "narrow view" of the parser state when extracting features used to predict actions. Our work is complementary to previous approaches that develop alternative transition operations to simplify the modeling problem and enable better attachment decisions (Nivre 2007, 2008, 2009; Bohnet and Nivre 2012; Choi and McCallum 2013) and to feature engineering (Zhang and Nivre 2011; Ballesteros and Bohnet 2014; Chen, Zhang, and Zhang 2014; Ballesteros and Nivre 2016).

Our model is related to recent work that uses neural networks in dependency parsing (Chen and Manning 2014; Weiss et al. 2015; Zhou et al. 2015; Andor et al. 2016).

2 The term "state" refers to the collection of previous decisions (sometimes called the history), resulting partial structures, which are stored in a stack data structure, and the words remaining to be processed.

3 These state equivalence assumptions are similar to the Markov assumptions made in modeling stochastic processes, according to which a potentially unbounded state history can be represented only in terms of a finite number of the most recent states.

That work can broadly be understood as replacing a conventional linear classifier with a neural network classifier but still only considering a "narrow view" of the parser state, whereas our work uses recursively defined networks to incorporate sensitivity to the complete parser state.

Using recursive/recurrent neural networks to compute representations of unboundedly large histories has been proposed previously (Henderson 2003; Titov and Henderson 2007b; Stenetorp 2013; Yazdani and Henderson 2015). Our innovation is to make the architecture of the neural network identical with the structure of stack data structures used to store the parser state. To do so, the technical challenge we must solve is being able to access, in constant time, a fixed-size representation of a stack as it evolves during the course of parsing. The technical innovation that lets us do this is a variation of recurrent neural networks with long short-term memory units (LSTMs), which we call **stack LSTMs** (Section 2). These can be understood as LSTMs augmented with a stack pointer that is manipulated by push and pop operations (in contrast to classic LSTMs that only ever read inputs sequentially).

Our parsing model uses three stack LSTMs to construct the parser state representation: one containing the unprocessed input tokens, one containing the stack of partial syntactic trees, and one containing the history of parse actions (Section 3). Because the stack of partial syntactic trees may contain both individual tokens and partial syntactic structures, representations of individual tree fragments are computed compositionally with recursive neural networks, similar to Paulus et al. (2014).

The parser depends on vector representations of word tokens. Such representations are most commonly learned as a dictionary lookup function (i.e., a map of each word type to a vector), but they can take a parameterized form that is, for example, sensitive to orthographic or morphological properties of a word's surface form (Botha and Blunsom 2014). For languages with rich inflective morphology, surface clues can be quite helpful for parsing (Ballesteros 2013). In Section 4, we compare the classical lookup-table approach to a recently proposed model that combines representations of individual characters (using sequential LSTMs) to obtain a representation of the word. With no explicit morphological annotation, we find that the latter representation gives a large performance increase when parsing Statistical Parsing of Morphologically Rich Languages (SPMRL) data sets in languages with agglutinative morphology or extensive case systems (Seddah et al. 2013; Seddah, Kübler, and Tsarfaty 2014), and performs as well as the table method in analytic languages. We further find that, without part-of-speech tags, this technique benefits all languages, and that in some cases this approach obviates the need for explicit part-of-speech information.

This article also explores two different approaches to training. Transition-based parsers are trained to optimize the score of the correct parsing action, given a parser state. The simplest strategy for choosing pairs of states and actions for training is one that follows a deterministic static oracle that maps each dependency tree from a treebank into a sequence of states and actions. However, this strategy only generates states that result from a history of correct parsing actions. As a result, models learned with this kind of training may not be robust to errors made at test time. We therefore also investigate a training procedure that also aims to teach the parser to make good predictions in sub-optimal states, facilitated by the use of dynamic oracles (Goldberg and Nivre 2012). The experiments in this article demonstrate that by coupling stack LSTM-based global state representations with dynamic oracle training, parsing with greedy decoding can achieve state-of-the-art parsing accuracies, rivaling the accuracies of previous parsers that rely on exhaustive search procedures.

This article is an extension of publications by Dyer et al. (2015) and Ballesteros et al. (2015). It contains a more detailed background about LSTMs and parsing, a discussion about the effect of random initialization, more extensive experiments with standard data sets, experiments including explicit morphological information that yields much higher results than the ones reported before, and a new training algorithm following dynamic oracles that yield state-of-the-art performance while maintaining a fast parsing speed with a greedy model. An open-source implementation of our parser, in all its different instantiations, is available from `https://github.com/clab/lstm-parser`.

2. Stack LSTMs

In this section we provide a review of LSTMs (Section 2.1), the core component of our parsing algorithm, and then define stack LSTMs (Section 2.2).

Notation. We follow the convention that vectors are written with lowercase, boldface letters (e.g., $\mathbf{v}$ or $\mathbf{v}_w$); matrices are written with uppercase, boldface letters (e.g., $\mathbf{M}$, $\mathbf{M}_a$, or $\mathbf{M}_{ab}$), and scalars are written as lowercase letters (e.g., s or q_z). Structured objects such as sequences of discrete symbols are written with lowercase, bold, italic letters (e.g., w refers to a sequence of input words). We use a semicolon to denote vector concatenation (e.g., $[\mathbf{a}; \mathbf{b}]$). Discussion of dimensionality is deferred to the experiments section (Section 6).

2.1 Long Short-Term Memories

Recurrent neural networks (RNNs) are non-linear functions from vector sequences to vector sequences. The term "recurrent" evokes the iterative application of the same function, given each prefix of the input, to calculate a "hidden state" vector. Concretely, let $\mathbf{x}_t$ be the tth input vector and $\mathbf{h}_t$ be the tth state vector. The incremental calculation is:

$$\mathbf{h}_t = \sigma(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{d})$$

$$\mathbf{W}_x \in \mathbb{R}^{d_{out} \times d_{in}} \quad \mathbf{W}_h \in \mathbb{R}^{d_{out} \times d_{out}} \quad \mathbf{d}, \mathbf{h}_t \in \mathbb{R}^{d_{out}} \quad \mathbf{x}_t \in \mathbb{R}^{d_{in}}$$

That is, the vector $\mathbf{x}_t$ is "read" in and a linear map is applied to it alongside the previous state vector $\mathbf{h}_{t-1}$, then passed through a nonlinear function (here, a component-wise logistic sigmoid, denoted σ). The state may then be further transformed to produce an output, either at each timestep or just at the end of the sequence.

The parameters of the RNN—here, $\mathbf{W}_x$, $\mathbf{W}_h$, and $\mathbf{d}$—are estimated to minimize a loss function on the training data. The gradient for this loss tends to "vanish" (go to zero) as we trace it back to earlier iterates in a long sequence. This means that long-range dependencies along the input sequence are hard to capture in a learned RNN (Bengio, Simard, and Frasconi 1994).

LSTMs are a variant of RNN designed to cope with the vanishing gradient problem (Hochreiter and Schmidhuber 1997; Graves 2013). They introduce an extra memory "cell," a vector $\mathbf{c}_t$ at each timestep. More specifically, LSTM cells process inputs with three multiplicative gates that control what proportion of the current input to pass into

the memory cell ($\mathbf{i}_t$) and what proportion of the previous memory cell to "forget" ($\mathbf{f}_t$). The updated value of the memory cell after an input $\mathbf{x}_t$ is computed as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{d}_i)$$

$$\mathbf{f}_t = \mathbf{1} - \mathbf{i}_t$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{d}_c)$$

where $\odot$ is the component-wise (Hadamard) product. The value $\mathbf{h}_t$ of the LSTM at each timestep is controlled by a third gate ($\mathbf{o}_t$) that is applied to the result of the application of a nonlinearity to the memory cell contents:

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{d}_o)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

$$\mathbf{W}_{\circ x} \in \mathbb{R}^{d_{out} \times d_{in}} \quad \mathbf{W}_{\circ h}, \mathbf{W}_{\circ c} \in \mathbb{R}^{d_{out} \times d_{out}} \quad \mathbf{i}_t, \mathbf{f}_t, \mathbf{c}_t, \mathbf{o}_t, \mathbf{h}_t, \mathbf{d}_\circ \in \mathbb{R}^{d_{out}} \quad \mathbf{x}_t \in \mathbb{R}^{d_{in}}$$

Note that our formulation differs slightly from the classic LSTM formulation in that it makes use of "peephole connections" that connect both cell values (the **c**s) and hidden states (the **h**s) to the gates (Gers, Schraudolph, and Schmidhuber 2003) and that it 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), LSTMs can be stacked in "layers" (Pascanu et al. 2013). 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).

Finally, output is produced at each timestep from the $\mathbf{h}_t$ value at the top layer:

$$\mathbf{y}_t = g(\mathbf{h}_t)$$

$$\mathbf{y}_t, \mathbf{h}_t \in \mathbb{R}^{d_{out}}$$

where g is an arbitrary differentiable function. We use the identity function in this article.

2.2 Stack Long Short-Term Memories

Conventional LSTMs model sequences in a left-to-right order.[4] Our innovation is to augment the LSTM with a "stack pointer." Like a conventional LSTM, new inputs are always added in the right-most position, but in stack LSTMs, the current location of the stack pointer determines which cell in the LSTM provides $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$ when computing the new memory cell contents.

In addition to adding elements to the end of the sequence, the stack LSTM provides a pop operation that moves the stack pointer to the previous element (i.e., the previous element that was extended, not necessarily the next-right-most element). Thus, the

4 Ours is not the first deviation from a strict left-to-right order: previous variations include bidirectional LSTMs (Graves and Schmidhuber 2005) and multidimensional LSTMs (Graves, Fernández, and Schmidhuber 2007).

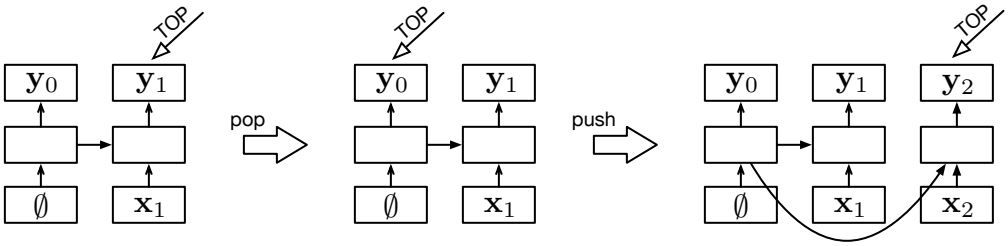

Figure 1
A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as TOP in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (right). The boxes in the lowest rows represent stack contents, which are the inputs to the LSTM, the upper rows are the outputs of the LSTM (in this article, only the output pointed to by TOP is ever accessed), and the middle rows are the memory cells (the $\mathbf{c}_i$s and $\mathbf{h}_i$s) and gates. Arrows represent function applications (usually affine transformations followed by a nonlinearity); refer to Section 2.1 for specifics.

LSTM can be understood as a stack implemented so that contents are never overwritten; that is, push always adds a new entry at the end of the list that contains a back-pointer to the previous top, and pop only updates the stack pointer.[5] This control structure is schematized in Figure 1.

By querying the output vector to which the stack pointer points (i.e., $\mathbf{h}_{\text{TOP}}$), a continuous-space "summary" of the contents of the current stack configuration is available. We refer to this value as the "stack summary." Note that this structure allows a given sequence history to have various different "continuations" that are independent from each other but are all linked to the same history. Signals from each continuation can then be backpropagated to the same shared history. Although elements near the top of the stack are likely to influence the stack summary the most, the LSTM has additional flexibility allowing it to learn to extract information from arbitrary points in the stack (Hochreiter and Schmidhuber 1997).

Although this architecture is novel, it is closely related to the recurrent neural network pushdown automaton of Das et al. (1992), which added an external stack memory to an RNN; the continuous stack transducers of Grefenstette et al. (2015); or the one proposed by Joulin and Mikolov (2015). Our formulation preserves the discrete push and pop operations of conventional stack data structures, whereas prior work uses continuous-valued relaxations of these.

3. Dependency Parser

We now turn to the problem of learning representations of dependency parser states. We preserve the standard data structures of a transition-based dependency parser, namely, a buffer of words to be processed (B) and a stack (S) of partially constructed syntactic elements. Each stack element is augmented with a continuous-space vector embedding representing a word and, in the case of S, any of its syntactic dependents. Additionally, we introduce a third stack (A) to represent the history of transition actions taken by the

5 Goldberg et al. (2013) propose a similar stack construction to prevent stack operations from invalidating existing references to the stack in a beam-search parser that must (efficiently) maintain a priority queue of stacks.

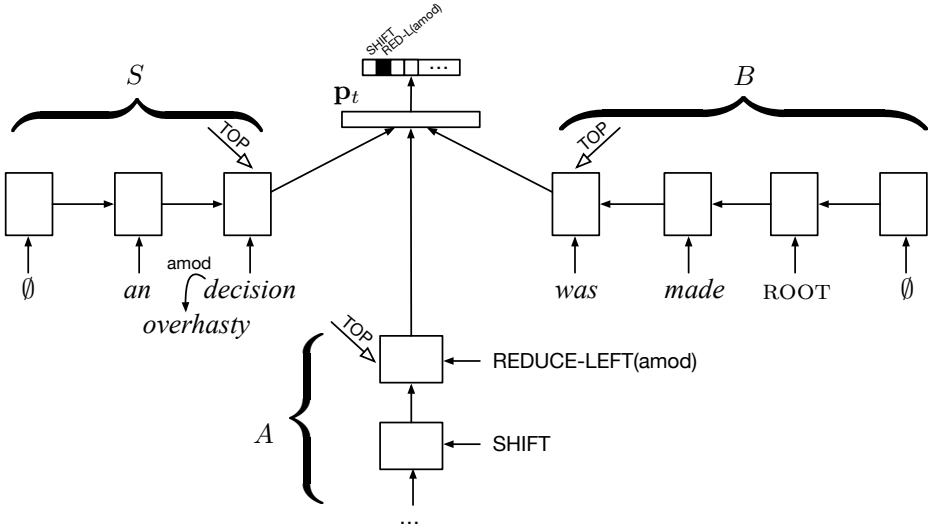

Figure 2
Parser state computation encountered while parsing the sentence *an overhasty decision was made*. Here *S* designates the stack of partially constructed dependency subtrees and its LSTM encoding; *B* is the buffer of words remaining to be processed and its LSTM encoding; and *A* is the stack representing the history of actions taken by the parser. These are linearly transformed, passed through a rectified linear unit nonlinearity to produce the parser state embedding $\mathbf{p}_t$. An affine transformation of this embedding is passed to a softmax layer to give a distribution over parsing decisions that can be taken.

parser.[6] Each of these stacks is associated with a stack LSTM that provides an encoding of its current contents. The full architecture is illustrated in Figure 2, and we will review each of the components in turn.

3.1 Parser Operation

The dependency parser is initialized by pushing the words and their representations (we discuss word representations in Section 4) of the input sentence in reverse order onto *B* such that the first word is at the top of *B* and the ROOT symbol is at the bottom, and *S* and *A* each contain an empty-stack token. At each timestep, the parser computes a composite representation of the stack states (as determined by the current configurations of *B*, *S*, and *A*) and uses that to predict an action to take, which updates the stacks. Processing completes when *B* is empty (except for the empty-stack symbol), *S* contains two elements, one representing the full parse tree headed by the ROOT symbol and the other the empty-stack symbol,[7] and *A* is the history of transition operations taken by the parser.

6 The *A* stack is only ever pushed to; our use of a stack here is purely for implementational and expository convenience.

7 The parser has the root as the last token to be included in the buffer, following Ballesteros and Nivre (2013).

The parser state representation at time t, which we write $\mathbf{p}_t$, which is used to determine the transition to take, is defined as follows:

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

$$\mathbf{s}_t, \mathbf{b}_t, \mathbf{a}_t \in \mathbb{R}^{d_{out}} \quad \mathbf{p}_t, \mathbf{d} \in \mathbb{R}^{d_{state}} \quad \mathbf{W} \in \mathbb{R}^{d_{state} \times 3d_{out}}$$

where $\mathbf{W}$ is a learned parameter matrix, $\mathbf{d}$ is a learned bias term, and $\mathbf{b}_t$, $\mathbf{s}_t$, and $\mathbf{a}_t$ are the stack LSTM encodings of B, S, and A, respectively. These are then passed through a component-wise rectified linear unit (ReLU) nonlinearity (Glorot, Bordes, and Bengio 2011).[8]

Finally, the parser state $\mathbf{p}_t$ is used to compute the probability of the parser action at time t as:

$$p(z_t \mid \mathbf{p}_t) = \frac{\exp\left(\mathbf{g}_{z_t}^\top \mathbf{p}_t + q_{z_t}\right)}{\sum_{z' \in \mathcal{A}(S,B)} \exp\left(\mathbf{g}_{z'}^\top \mathbf{p}_t + q_{z'}\right)} \tag{1}$$

$$\mathbf{p}_t, \mathbf{g}_z \in \mathbb{R}^{d_{state}}$$

where $\mathbf{g}_z$ is a column vector representing the (output) embedding of the parser action z, and q_z is a bias term for action z. The set $\mathcal{A}(S, B)$ represents the valid transition actions that may be taken given the current contents of the stack and buffer.[9] Because $\mathbf{p}_t = f(\mathbf{s}_t, \mathbf{b}_t, \mathbf{a}_t)$ encodes information about all previous decisions made by the parser, the chain rule may be invoked to write the probability of any valid sequence of parse transitions z conditional on the input as:

$$p(z \mid w) = \prod_{t=1}^{|z|} p(z_t \mid \mathbf{p}_t) \tag{2}$$

3.2 Composition Functions

Recursive neural network models enable complex phrases to be represented compositionally in terms of their parts and the relations that link them (Hermann and Blunsom 2013; Socher et al. 2011, 2013a, 2013b). We follow previous work in embedding dependency tree fragments that are present in the stack S in the same vector space as the token embeddings discussed in Section 4.

A particular challenge here is that a syntactic head may, in general, have an arbitrary number of dependents. To simplify the parameterization of our composition function, we combine head-modifier pairs one at a time, building up more complicated structures

8 In preliminary experiments, we tried several nonlinearities and found ReLU to work slightly better than the others.

9 In general, $\mathcal{A}(S, B)$ is the complete set of parser actions discussed in Section 3.3, but in some cases not all actions are available. For example, when S is empty and words remain in B, a SHIFT transition is obligatory (Nivre 2008).

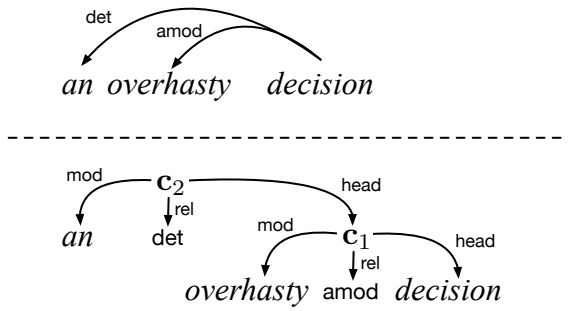

Figure 3
The representation of a dependency subtree (top) is computed by recursively applying composition functions to ⟨head, modifier, relation⟩ triples. In the case of multiple dependents of a single head, the recursive branching order is imposed by the order of the parser's reduce transition (bottom).

in the order they are "reduced" in the parser, as illustrated in Figure 3. Each node in this expanded syntactic tree has a value computed as a function of its three arguments: the syntactic head (**h**), the dependent (**m**), and the syntactic relation being satisfied (**r**). We define this by concatenating the vector embeddings of the head, dependent, and relation and applying a linear operator and a component-wise nonlinearity as follows:

$$\mathbf{c} = \tanh\left(\mathbf{U}[\mathbf{h}; \mathbf{m}; \mathbf{r}] + \mathbf{e}\right)$$

$$\mathbf{c}, \mathbf{e}, \mathbf{h}, \mathbf{m} \in \mathbb{R}^{d_{in}} \quad \mathbf{r} \in \mathbb{R}^{d_{rel}} \quad \mathbf{U} \in \mathbb{R}^{d_{in} \times (2d_{in} + d_{rel})}$$

For the relation vector, we use an embedding of the parser action that was applied to construct the relation (i.e., the syntactic relation paired with the direction of attachment); **U** and **e** are additional parameters learned alongside those of the stack LSTMs.

3.3 Transition Systems

Our parser implements various transition systems. The original version (Dyer et al. 2015) implements arc-standard (Nivre 2004) in its most basic set-up. In Ballesteros et al. (2015), we augmented the arc-standard system with the SWAP transition of Nivre (2009), allowing for nonprojective parsing. For the dynamic oracle training strategy, we moved to an arc-hybrid system (Kuhlmann, Gómez-Rodríguez, and Satta 2011) for which an efficient dynamic oracle is available. It is worth noting that the stack LSTM parameterization is mostly orthogonal to the transition system being used (providing that the system can be expressed as operating on stacks), making it easy to substitute transition systems.

3.3.1 Arc-Standard. The arc-standard transition inventory (Nivre 2004) is given in Figure 4. The SHIFT transition moves a word from the buffer to the stack, and the REDUCE-LEFT and REDUCE-RIGHT transitions pop the two items from the top of the stack, make one of them the parent of the other, and push the resulting subtree back to

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

Figure 4
Parser transitions of the *arc-standard* system (with swap, Section 3.3.2) 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. $(g_r(\mathbf{x}, \mathbf{y}), x)$ refers to the composition function presented in 3.2.

the stack. The arc-standard system allows building all and only projective trees. In order to parse nonprojective trees, this can be combined with the pseudo-projective approach (Nivre and Nilsson 2005) or follow what is presented in Section 3.3.2.

3.3.2 Arc-Standard with Swap. In order to deal with nonprojectivity, the arc-standard system can be augmented with a SWAP transition (Nivre 2009). The SWAP transition removes the second-to-top item from the stack and pushes it back to the buffer, allowing for the creation of nonprojective trees. We only use this transition when the training data set contains nonprojective trees. The inclusion of the SWAP transition requires breaking the linearity of the stack by removing tokens that are not at the top of the stack; however, this is easily handled with the stack LSTM. Figure 4 shows how the parser is capable of moving words from the stack (S) to the buffer (B), breaking the linear order of words. Because 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.

3.3.3 Arc-Hybrid. For the dynamic oracle training scenario, described in Section 5.2, we switch to the arc-hybrid transition system, which is amenable to an efficient dynamic oracle (Goldberg and Nivre 2013). The arc-hybrid system is summarized in Figure 5. The SHIFT and REDUCE-RIGHT transitions are the same as in arc-standard. However, the REDUCE-LEFT transition pops the top of the stack and attaches it as a child of the first item in the buffer. Although it is possible to extend the arc-hybrid system to support nonprojectivity by adding a SWAP transition, this extension would invalidate an important guarantee enabling efficient dynamic oracles.[10] We therefore restrict the dynamic-oracle experiments to the fully projective English and Chinese treebanks. In order to parse nonprojective trees, this can be combined with the pseudo-projective approach (Nivre and Nilsson 2005).

4. Word Representations

Our parser's architecture makes heavy use of word representations. We describe two variants. The first is a "standard" approach in which each word is represented

10 Specifically: For every possible parser configuration $\mathbf{p}$ and group of arcs A, if each arc in A can be derived from $\mathbf{p}$, then a valid tree structure containing *all* of the arcs in A can also be derived from $\mathbf{p}$. This is a sufficient condition, but whether it is necessary is unknown; hence the question of an efficient, $O(1)$ dynamic oracle for the augmented system is an open research problem.

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

Figure 5
Parser transitions of the arc-hybrid system (Section 3.3.3). **Bold** symbols indicate (learned) embeddings of words and relations, italic symbols indicate the corresponding words and relations. $(g_r(\mathbf{x}, \mathbf{y}), x)$ refers to the composition function presented in 3.2.

as a vector, which is pretrained on unannotated data (Section 4.1). The second is a character-based representation, in which the word's representation is highly dependent on its orthography. These character-based representations are trained without additional resources and work well for implicitly capturing the morphology of words (Section 4.2).

4.1 Standard Word Representations

To represent each input token, we concatenate three vectors: a learned vector representation for each word type ($\mathbf{w}$), a fixed vector representation from a neural language model ($\tilde{\mathbf{w}}_{\text{LM}}$), and a learned representation ($\mathbf{t}$) of the POS tag of the token, 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:

$$\mathbf{x} = \max\{\mathbf{0}, \mathbf{V}[\mathbf{w}; \tilde{\mathbf{w}}_{\text{LM}}; \mathbf{t}] + \mathbf{d}\}$$

$$\mathbf{x}, \mathbf{d} \in \mathbb{R}^{d_{in}} \quad \mathbf{w} \in \mathbb{R}^{d_{word}} \quad \tilde{\mathbf{w}}_{\text{LM}} \in \mathbb{R}^{d_{lm}} \quad \mathbf{t} \in \mathbb{R}^{d_{pos}} \quad \mathbf{V} \in \mathbb{R}^{d_{in} \times (d_{word} + d_{lm} + d_{pos})}$$

This mapping can be shown schematically as in Figure 6.

If there are morphological features available, containing information about gender, number or case, then we can also concatenate with a fourth vector that is a learned

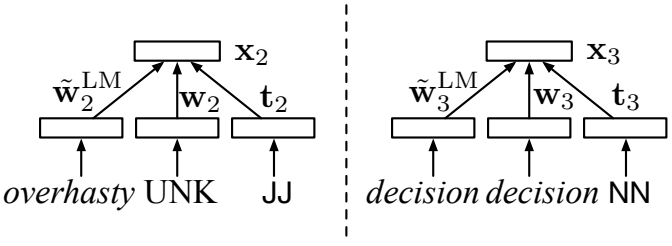

Figure 6
Token embedding of the words *decision*, which is present in both the parser's training data and the language model data, and *overhasty*, an adjective that is not present in the parser's training data but is present in the LM data.

representation (**f**) of the morphological tags of the token, again provided as auxiliary input to the parser. The linear map (**V**) would then be:

$$\mathbf{x} = \max\{\mathbf{0}, \mathbf{V}[\mathbf{w}; \tilde{\mathbf{w}}_{\text{LM}}; \mathbf{t}; \mathbf{f}] + \mathbf{d}\}$$

$$\mathbf{x}, \mathbf{d} \in \mathbb{R}^{d_{in}} \quad \mathbf{w} \in \mathbb{R}^{d_{word}} \quad \tilde{\mathbf{w}}_{\text{LM}} \in \mathbb{R}^{d_{lm}} \quad \mathbf{t} \in \mathbb{R}^{d_{pos}} \quad \mathbf{f} \in \mathbb{R}^{d_{morph}}$$

$$\mathbf{V} \in \mathbb{R}^{d_{in} \times (d_{word} + d_{lm} + d_{pos} + d_{morph})}$$

This representation lets us deal flexibly with out-of-vocabulary (OOV) words—those that are OOV in the very limited parsing data but present in the pretraining language model (LM), and words that are OOV in both. To ensure we have estimates of the OOVs in the parsing training data, we stochastically replace (with probability 0.5) each singleton word type in the parsing training data with the UNK token in each training iteration.

Pretrained Word Embeddings. There are several options for creating word embeddings, meaning numerous pretraining options for $\tilde{\mathbf{w}}_{\text{LM}}$ are available. However, for syntax modeling problems, embedding approaches that discard order perform less well (Bansal, Gimpel, and Livescu 2014); therefore, we used a variant of the skip *n*-gram model introduced by Ling et al. (2015a), named "structured skip *n*-gram," where a different set of parameters is used to predict each context word depending on its position relative to the target word.

4.2 Modeling Characters Instead of Words

Following Ling et al. (2015b), 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 $\overrightarrow{\mathbf{w}}$. 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 **w**. As in Section 4.1, a linear map (**V**) is applied and passed through a component-wise ReLU.

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

$$\mathbf{x}, \mathbf{d} \in \mathbb{R}^{d_{in}} \quad \overrightarrow{\mathbf{w}}, \overleftarrow{\mathbf{w}} \in \mathbb{R}^{d_{out}} \quad \mathbf{t} \in \mathbb{R}^{d_{pos}} \quad \mathbf{V} \in \mathbb{R}^{d_{in} \times 2d_{out} + d_{pos}}$$

This process is shown schematically in Figure 7.

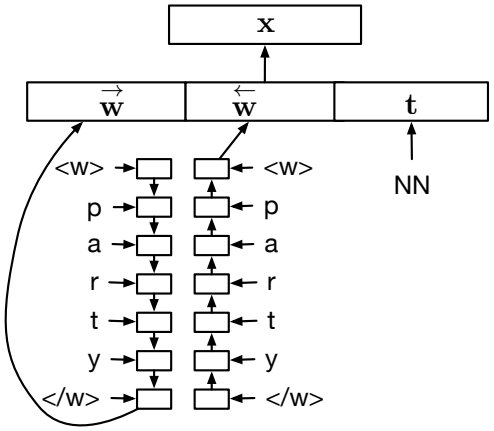

Figure 7
Character-based word embedding of the word *party*. This representation is used for both in-vocabulary and out-of-vocabulary words.

As with the standard word representations, it is possible to concatenate with a fourth vector which is a learned representation ($\mathbf{f}$) of the morphological tags of the token, again provided as auxiliary input to the parser:

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

$$\mathbf{x}, \mathbf{d} \in \mathbb{R}^{d_{in}} \quad \overrightarrow{\mathbf{w}}, \overleftarrow{\mathbf{w}} \in \mathbb{R}^{d_{out}} \quad \mathbf{t} \in \mathbb{R}^{d_{pos}} \quad \mathbf{f} \in \mathbb{R}^{d_{morph}} \quad \mathbf{V} \in \mathbb{R}^{d_{in} \times 2d_{out} + d_{pos} + d_{morph}}$$

Note that under this representation, OOV 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 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 in Section 4.1 and other parsers without additional resources. In Section 6 we confirm this hypothesis.

Learned Word Representations. Figure 8 visualizes a sample of the character-based bidirectional LSTMs learned representations (using the model referred to as **Chars** in Section 6.4). 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.

5. Training Procedure

The parser is trained to maximize the conditional probability of taking a "correct" action at each parsing state. (The definition of what constitutes a "correct" action is the major difference between static oracle and dynamic oracle training.) We begin with the simpler strategy of training with a static oracle (Section 5.1), then turn to dynamic oracles (Section 5.2).

Regardless of the oracle, our training implementation constructs a computation graph (nodes that represent values, linked by directed edges from each function's inputs to its outputs) for the negative log probability for the oracle transition sequence as a function of the current model parameters and uses forward- and backpropagation to obtain the gradients with respect to the model parameters (LeCun et al. 1998,

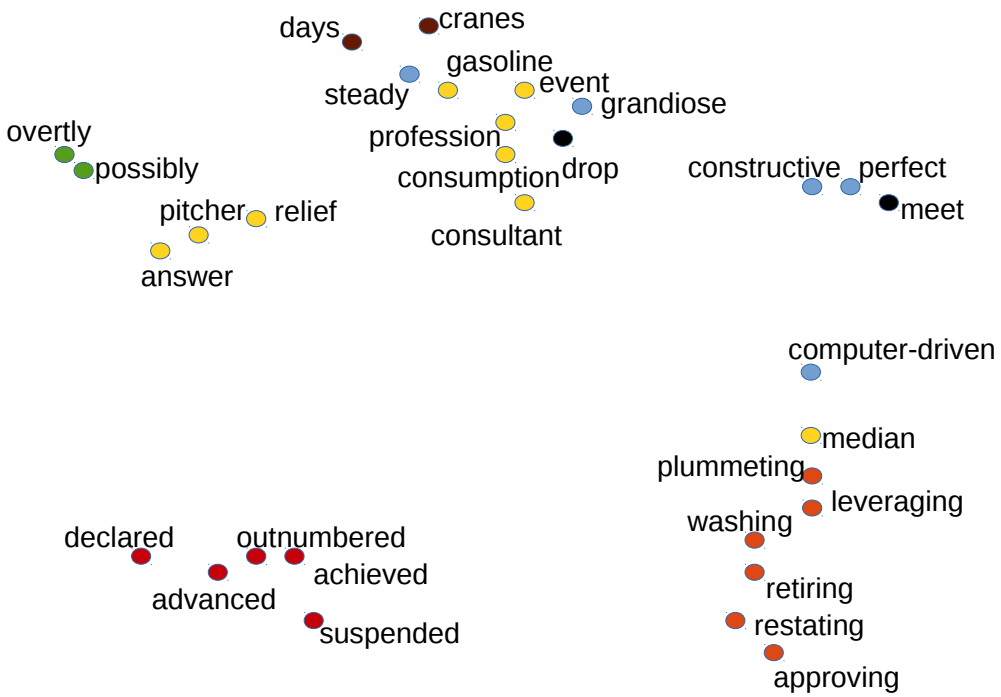

Figure 8
Character-based word representations of 30 random words from the English development set (**Chars** model; see Section 6.4). Dots in red represent past tense verbs; dots in orange represent gerund verbs; dots in black represent present tense verbs; dots in blue represent adjectives; dots in 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/`.

section 4). The computations for a single parsing model were run on a single thread on a CPU. Using the dimensions discussed in the next section, we required between 8 and 12 hours to reach convergence on a held-out development set.[11]

5.1 Static Oracle Training

With a static oracle, the training procedure computes a canonical reference series of transitions for each gold parse tree. It then runs the parser through this canonical sequence of transitions, while keeping track of the state representation $\mathbf{p}_t$ at each step t, as well as the distribution over transitions $p(z_t \mid \mathbf{p}_t)$ that is predicted by the current classifier for the state representation. Once the end of the sentence is reached, the parameters are updated towards minimizing the cross-entropy of the reference transition sequence (Equation (2)) by minimizing the cross-entropy (i.e., maximizing the log-likelihood) of the correct transition $p(z_t \mid \mathbf{p_t})$ at each state along the path.

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

5.2 Training with Exploration using Dynamic Oracles

In the static oracle case, the parser is trained to predict the best transition to take at each parsing step, assuming all previous transitions were correct. Because the parser is likely to make mistakes at test time and encounter states it has not seen during training, this training criterion is problematic (Daumé III, Langford, and Marcu 2009; Ross, Gordon, and Bagnell 2011; Goldberg and Nivre 2012, 2013, inter alia). Instead, we would prefer to train the parser to behave optimally also after making a mistake (under the constraints that it cannot backtrack or fix any previous decision). We thus need to include in the training examples states that result from wrong parsing decisions, together with the optimal transitions to take in these states. To this end we reconsider which training examples to show, and what it means to behave optimally on these training examples. The dynamic oracles framework for training with exploration, suggested by Goldberg and Nivre (2012, 2013), provides answers to these questions.

Although the application of dynamic oracle training is relatively straightforward, some adaptations were needed to accommodate the probabilistic training objective. These adaptations mostly follow Goldberg (2013).

5.2.1 Dynamic Oracles. A **dynamic oracle** is the component that, given a gold parse tree, provides the optimal set of possible actions to take under a given parser state. In contrast to static oracles that derive a canonical sequence for each gold parse tree and say nothing about parsing states that do not stem from this canonical path, the dynamic oracle is well-defined for states that result from parsing mistakes, and may produce more than a single gold action for a given state. Under the dynamic oracle framework, a parsing action is said to be optimal in a given state if the best tree that can be reached after taking the action is no worse (in terms of accuracy with respect to the gold tree) than the best tree that could be reached prior to taking that action.

Goldberg and Nivre (2013) define the arc-decomposition property of transition systems, and show how to derive efficient dynamic oracles for transition systems that are arc-decomposable (see footnote 10). Unfortunately, the arc-standard transition system does not have this property. Although it is possible to compute dynamic oracles for the arc-standard system (Goldberg, Sartorio, and Satta 2014), the computation relies on a dynamic programming algorithm that is polynomial in the length of the stack. As the dynamic oracle has to be queried for each parser state seen during training, the use of this dynamic oracle will make the training times several times longer. We chose instead to switch to the arc-hybrid transition system (Kuhlmann, Gómez-Rodríguez, and Satta 2011), which is very similar to the arc-standard system but is arc-decomposable and hence admits an efficient $O(1)$ dynamic oracle, resulting in only negligible burden on the training time. We implemented the dynamic oracle to the arc-hybrid system as described by Goldberg (2013).

5.2.2 Training with Exploration. In order to expose the parser to configurations that are likely to result from incorrect parsing decisions, we make use of the probabilistic nature of the classifier. During training, instead of following the gold action, we sample the next transition according to the output distribution the classifier assigns to the current configuration. Another option, taken by Goldberg and Nivre, is to follow the one-best action predicted by the classifier. However, initial experiments showed that the one-best approach did not work well. Because the neural-network classifier becomes accurate early on in the training process, the one-best action is likely to be correct, and the parser is then exposed to very few error states in its training process. By sampling from the

predicted distribution, we are effectively increasing the chance of straying from the gold path during training, while still focusing on mistakes that receive relatively high parser scores. We believe further formal analysis of this method will reveal connections to reinforcement learning and, perhaps, other methods for learning complex policies.

Taking this idea further, we could increase the number of error-states observed in the training process by changing the sampling distribution so as to bias it toward more low-probability states. We do this by raising each probability to the power of α ($0 < \alpha \leq 1$) and re-normalizing. This transformation keeps the relative ordering of the events, while shifting probability mass towards less frequent events. As we show subsequently, this turns out to be very beneficial for the configurations that make use of external embeddings. Indeed, these configurations achieve high accuracies and shared class distributions early on in the training process.

The parser is trained to optimize the log-likelihood of a correct action z_g at each parsing state $\mathbf{p}_t$ according to Equation (1). When using the dynamic oracle, a state $\mathbf{p}_t$ may admit multiple correct actions $z_g = \{z_{g_i}, \ldots, z_{g_k}\}$. Our objective in such cases is that the set of correct actions receives high probability. We optimize for the log of $p(z_g \mid \mathbf{p}_t)$, defined as:

$$p(z_g \mid \mathbf{p}_t) = \sum_{z_{g_i} \in z_g} p(z_{g_i} \mid \mathbf{p}_t) \tag{3}$$

A similar approach was taken by Riezler et al. (2000), Charniak and Johnson (2005), and Goldberg (2013) in the context of log-linear probabilistic models.

6. Experiments

After describing the implementation details of our optimization procedure (Section 6.1) and the data used in our experiments (Section 6.2), we turn to four sets of experiments:

1. First, we assess the quality of our greedy, global-state stack LSTM parsers on a wide range of data sets, showing that it is highly competitive with the state of the art (Section 6.3).
2. We compare the performance of the two different word representations (Section 4) across different languages, finding that they are quite beneficial in some cases and harmless in most others (Section 6.4).
3. We compare dynamic oracles to static oracles, showing that training with dynamic oracles attains even better performance (Section 6.5).
4. We compare our parser with other existing methods on the CoNLL-2009 data sets (Section 6.6).

6.1 Optimization Procedure Details

Parameter optimization was performed using stochastic gradient descent with an initial learning rate of $\eta_0 = 0_1$; the learning rate was updated on each pass through the training data as $\eta_t = \eta_0/(1 + \rho t)$, with $\rho = 0.1$ and where t is the number of epochs completed. No momentum was used. To mitigate the effects of "exploding" gradients, we clipped the ℓ_2 norm of the gradient to 5 before applying the weight update rule

(Graves 2013; Sutskever, Vinyals, and Le 2014). An ℓ_2 penalty of 1×10^{-6} was applied to all weights.

Matrix and vector parameters were initialized with uniform samples in $\pm\sqrt{6/(r+c)}$, where r and c were the number of rows and columns in the structure (Glorot and Bengio 2010). We use mini-batches of 1,000 sentences each and update every 25 iterations.

Dimensionality. The full version of our parsing model sets dimensionalities as follows. LSTM hidden states (d_{out}) are of size 100, and we use two layers of LSTMs for each stack. Embeddings of the parser actions used in the composition functions (d_{state}) have 16 dimensions, and the output embedding (d_{out}) size is 20 dimensions. Pretrained word embeddings (d_{dlm}) have 100 dimensions (English) and 80 dimensions (Chinese), and the learned word embeddings (d_{word}) have 32 dimensions. Part of speech and morphological tag embeddings (d_{pos} and d_{morph}) have 12 dimensions, when included. The character-based embeddings are of size 100, when included. The combined representations of the inputs have 100 dimensions.

These dimensions were chosen based on intuitively reasonable values (words should have higher dimensionality than parsing actions, POS tags, and relations; LSTM states should be relatively large), and it was confirmed on development data that they performed well.[12] Future work might more carefully optimize the parser's hyperparameters. A variety of automated techniques have been developed for hyperparameter optimization, including ones that enable evidence-sharing across related tasks (Yogatama and Mann 2014; Yogatama, Kong, and Smith 2015). Our architecture strikes a balance between minimizing computational expense and finding solutions that work.

Pretrained Word Vectors. The hyperparameters of the structured skip n-gram model are the same as in the skip n-gram model defined in the widely used `word2vec` implementation (Mikolov et al. 2013). We set the window size to 5, used a negative sampling rate to 10, and ran 5 epochs through unannotated corpora described in Section 6.2.

6.2 Data

We use different data depending on the targeted experiment; in this section we describe the data sets that we used to test the basic model, the character-based representations that are geared towards morphology, and the dynamic oracles. Finally, we also report results with the CoNLL 2009 data sets (Hajič et al. 2009) to make a complete comparison with similar systems.

6.2.1 Data for the Simplest Model that Uses Word Representations and Static Oracle for Training. We used the same data set-up as Chen and Manning (2014), namely, an English task and a Chinese task. This baseline configuration was chosen because they likewise used a neural parameterization to predict actions in an arc-standard transition-based parser.

- For English, we used the Stanford Dependency (SD) treebank (Marneffe, MacCartney, and Manning 2006) used in Chen and Manning (2014), which

12 We did perform preliminary experiments with LSTM states of 32, 50, and 80, but the other dimensions were our initial guesses.

is the closest model published, with the same splits.[13] The part-of-speech tags are predicted by using the Stanford POS tagger (Toutanova et al. 2003) with an accuracy of 97.3%. This treebank contains a negligible number of nonprojective arcs (Chen and Manning 2014).

- For Chinese, we use the Penn Chinese Treebank 5.1 (CTB5) following Zhang and Clark (2008),[14] with gold part-of-speech tags, which is also the same as in Chen and Manning (2014).

Language model word embeddings were generated from the AFE portion of the English Gigaword corpus (version 5), and from the complete Chinese Gigaword corpus (version 2), as segmented by the Stanford Chinese Segmenter (Tseng et al. 2005).

6.2.2 Data to Test the Character-Based Representations and Static Oracle for Training. For the character-based representations we applied our model to the treebanks of the SPMRL Shared Task (Seddah et al. 2013; Seddah, Kübler, and Tsarfaty 2014): Arabic (Maamouri et al. 2004), Basque (Aduriz et al. 2003), French (Abeillé, Clément, and Toussenel 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, Nilsson, and Hall 2006). For all the corpora of the SPMRL Shared Task, we used predicted POS tags as provided by the shared task organizers.[15] We also ran the experiment with the Turkish dependency treebank[16] (Oflazer et al. 2003) of the CoNLL-X Shared Task (Buchholz and Marsi 2006) and we use gold POS tags when used as it is common with the CoNLL-X data sets. In addition to these, we include the English and Chinese data sets described in Section 6.2.1.

6.2.3 Data for the Dynamic Oracle. Because the arc-hybrid transition-based parsing algorithm is limited to fully projective trees, we use the same data as in Section 6.2.1, which makes it comparable with the basic model that uses standard word representations and a static oracle arc-standard algorithm.

6.2.4 CoNLL-2009 Data. We also report results with all the CoNLL 2009 data sets (Hajič et al. 2009) to make a complete comparison with similar systems both for static and dynamic oracles.

6.3 Experiments with Static Oracle and Standard Word Representations

We report results on five experimental configurations per language, as well as the Chen and Manning (2014) baseline. These are: the full stack LSTM parsing model (S-LSTM), the stack LSTM parsing model without POS tags (−POS), the stack LSTM parsing model without pretrained language model embeddings (−pretraining), the

13 Training: 02-21. Development: 22. Test: 23.

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

15 The POS tags were calculated with MarMot tagger (Mueller, Schmid, and Schütze 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.

16 Because the Turkish dependency treebank does not have a development set, we extracted the last 150 sentences from the 4,996 sentences of the training set as a development set.

Table 1
Unlabeled attachment scores and labeled attachment scores on the **development** sets (top) and the final **test** sets (bottom) for Chinese and English (SD). The columns marked with S-LSTM show the results of the parser with the whole S-LSTM model. The columns marked with −POS show the results of the parser without POS tags. The columns marked with −pretraining show the results of the parser without pretraining. The columns marked with S-RNN show the results of the parser in which the S-LSTM is replaced by a recurrent neural network. The columns marked with C&M (2014) show the results of Chen and Manning (2014).

Development

	S-LSTM		−POS		−pretraining		−composition		S-RNN		C&M (2014)	
Language	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Chinese	**87.23**	**85.87**	82.65	79.84	85.96	84.37	86.32	84.59	86.30	84.70	84.0	82.4
English	**93.11**	**90.80**	92.58	89.96	92.58	90.24	92.50	89.79	92.80	90.40	92.2	89.7

Test

	S-LSTM		−POS		−pretraining		−composition		S-RNN		C&M (2014)	
Language	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Chinese	**86.85**	**85.36**	82.15	79.04	85.48	83.94	86.20	84.49	86.10	84.60	83.9	82.4
English	**93.04**	**90.87**	92.57	90.21	92.40	90.04	92.10	89.61	92.30	90.10	91.8	89.6

stack LSTM parsing model that uses just head words on the stack instead of composed representations (−composition), and the full parsing model where rather than an LSTM, a simple recurrent neural network in which normal sigmoidal hidden units are used instead of the LSTM, but augmented with a stack pointer (S-RNN).

6.3.1 Results. Following Chen and Manning (2014), we exclude punctuation symbols for evaluation. Table 1 shows parsing scores comparable to Chen and Manning (2014),[17] and we show that our model is better than their model on both the development set and the test set.

Overall, our parser substantially outperforms the baseline neural network parser of Chen and Manning (2014), both in the full configuration and in the various ablated conditions we report. The one exception to this is the −POS condition for the Chinese parsing task, in which we underperform their baseline (which used gold POS tags), although we do still obtain reasonable parsing performance in this limited case. We note that predicted POS tags in English add very little value—suggesting that we can think of parsing sentences directly without first tagging them. We also find that using composed representations of dependency tree fragments outperforms using representations of head words alone. Finally, we find that whereas LSTMs outperform classical RNNs, the latter are still quite capable of learning good representations.

Training and Testing Times. Training is performed with mini-batches and requires around 20 hours to achieve convergence in the English data set although it depends on the effect of initialization; the parser achieves an end-to-end runtime of 40,539.8 milliseconds to parse the entire English test data (2.4k sentences) on a single CPU core. Other data sets require similar runtime per sentence.

17 The results reported by Dyer et al. (2015) were obtained with an earlier version of the neural network machinery and are similar, though not exactly the same, as those reported in this article.

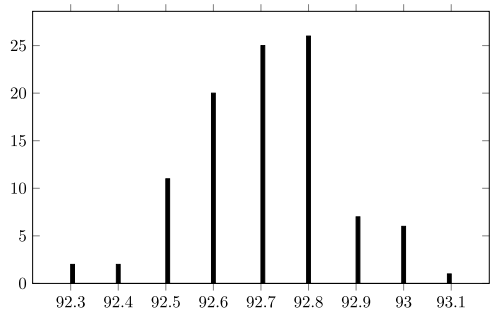

Figure 9
Histogram with the results obtained with 100 different initializations. The *x*-axis plots the unlabeled attachment score on the test set, and the *y*-axis plots the number of times a result occurred.

Effect of Initialization. The parser uses random seeds to create the mini-batches that it uses to train the model, which has an effect on the parsing results. In order to see the effect of initialization, we trained the whole S-LSTM parser with POS tags, pretrained word embeddings, and composition functions on the English SD data set, for the same amount of time (around 20 hours) using different random seeds. Figure 9 shows the variation due to initialization for 100 different random seeds and their results on the English test set. We observe that even though the results are very similar with a minimum of 92.3 and a maximum of 93.1, there are differences. The modal result is 92.8.[18]

Effect of Beam Size. Beam search was determined to have minimal impact on scores (absolute improvements of $\leq 0.3\%$ were observed with small beams). Therefore, all results we report used greedy decoding—Chen and Manning (2014) likewise only report results with greedy decoding. This finding is in line with previous work that generates sequences from recurrent networks (Grefenstette et al. 2014), although Vinyals et al. (2015),[19] Zhou et al. (2015), and Weiss et al. (2015) did report much more substantial improvements with beam search on their parsers.

6.4 Experiments with Character-Based Word Representations

In order to isolate the improvements provided by the LSTM encodings of characters described in Section 4.2, we run the stack LSTM parser in the following configurations:

- **Words**: words only, as in Section 4.1 (but without POS tags)
- **Chars**: character-based representations of words with bidirectional LSTMs, as in Section 4.2 (but without POS tags)
- **Words + POS**: words and POS tags (Section 4.1)

18 The results shown in Table 1 for the whole S-LSTM parser and English-SD are the best results for the development set, even though the histogram in Figure 9 shows a couple of models with better results on the test set.

19 Although superficially similar to ours, Vinyals et al. (2015) is a phrase-structure parser and adaptation to the dependency parsing scenario would have been nontrivial. We discuss their work in Section 7.

- **Chars + POS**: character-based representations of words with bidirectional LSTMs plus POS tags (Section 4.2)
- **Words + POS + Morph**: words, POS tags and morphological tags (Section 4.1)
- **Chars + POS + Morph**: character-based representations of words with bidirectional LSTMs plus POS tags and morphological tags (Section 4.2)

None of the experimental configurations include pretrained word-embeddings or any additional data resources. All experiments include SWAP transition, meaning that nonprojective trees could be produced in any language. Finally, in addition to the dimensionality described in Section 6.1, the character-based representations of words have 100 dimensions.

6.4.1 Results and Discussion. Tables 2 and 3 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**.

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. This is evidenced by the results including explicit morphological features, which outperforms the ones without them in most cases, and outperforms the results using character-based representations. It is interesting to see that for some languages, such as German, Hungarian, Korean, and Swedish, the character-based representations seem to be enough to approach or even surpass the results of the parser by using rich morphological features. In Korean, **Chars** is the best model, which suggests that the character-based word representations encode as much useful information for parsing for that particular language as is currently available.

Swedish, English, and French use suffixes for the verb tenses and number,[20] and Hebrew uses a root-and-pattern system. Even for Chinese, which is not morphologically rich, **Chars** shows a benefit over **Words**, perhaps by capturing regularities in syllable structure within words.

6.4.2 Out-of-Vocabulary Words. The character-based representation for words is notably beneficial for 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 10 shows how this drop varies with the development OOV rate across treebanks; the most extreme is Korean, which drops 15.5 LAS. A similar, but less pronounced, pattern was observed for models that include POS.

20 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).

Table 2
Unlabeled attachment scores (top) and labeled attachment scores (bottom) 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. Boldface shows the better result comparing **Words** vs. **Chars**, comparing **Words + POS** vs. **Chars + POS**, and comparing **Words + POS + Morph** vs. **Chars + POS + Morph**.

UAS:

Language	Words	Chars	Words + POS	Chars + POS	Words + POS + Morph	Chars + POS + Morph
Arabic	86.14	**87.20**	**87.44**	87.07	**87.35**	86.78
Basque	78.42	**84.97**	83.49	**85.58**	**87.20**	86.90
French	84.84	**86.21**	**87.00**	86.33	**87.19**	86.57
German	88.14	**90.94**	91.16	**91.23**	91.44	**91.45**
Hebrew	79.73	**79.92**	**81.99**	80.76	**82.10**	80.69
Hungarian	72.38	**80.16**	78.47	**80.85**	80.29	**80.55**
Korean	78.98	**88.98**	87.36	**89.14**	87.48	**89.04**
Polish	73.29	**85.69**	**89.32**	88.54	**90.34**	89.36
Swedish	73.44	**75.03**	**80.02**	78.85	**80.14**	79.79
Turkish	71.10	**74.91**	77.13	**77.96**	**79.28**	79.18
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	Words + POS + Morph	Chars + POS + Morph
Arabic	82.73	**84.34**	**84.81**	84.36	**84.71**	83.95
Basque	67.08	**78.22**	74.31	**79.52**	**81.50**	80.85
French	80.32	**81.70**	**82.71**	81.51	**82.68**	82.12
German	85.36	**88.68**	**89.04**	88.83	**89.42**	89.31
Hebrew	69.42	**70.58**	**74.11**	72.18	**73.98**	72.37
Hungarian	62.14	**75.61**	69.50	**76.16**	75.55	**75.85**
Korean	67.48	**86.80**	83.80	**86.88**	83.78	**86.92**
Polish	65.13	**78.23**	**81.84**	80.97	**84.62**	82.89
Swedish	64.77	**66.74**	**72.09**	69.88	**73.13**	70.87
Turkish	53.98	**62.91**	62.30	**62.87**	**71.42**	71.05
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**	–	–

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.

Finally, Table 4 shows the results obtained when combining the pretrained word embeddings of Section 6.3 with the character-based representations of this section for the English and Chinese Treebanks. We also run the experiments by including and

Table 3
Unlabeled attachment scores (top) and labeled attachment scores (bottom) 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. Boldface shows the better result comparing **Words** vs. **Chars**, comparing **Words + POS** vs. **Chars + POS**, and comparing **Words + POS + Morph** vs. **Chars + POS + Morph**.

UAS:

Language	**Words**	**Chars**	**Words + POS**	**Chars + POS**	**Words + POS + Morph**	**Chars + POS + Morph**
Arabic	85.21	**86.08**	86.05	**86.07**	**86.35**	85.79
Basque	77.06	**85.19**	82.92	**85.22**	**86.87**	86.02
French	83.74	**85.34**	**86.15**	85.78	**86.81**	86.21
German	82.75	**86.80**	**87.33**	87.26	87.40	**87.48**
Hebrew	77.62	**79.93**	**80.68**	80.17	**81.36**	79.83
Hungarian	72.78	**80.35**	78.64	**80.92**	**81.07**	81.06
Korean	78.70	**88.39**	86.85	**88.30**	86.84	**88.24**
Polish	72.01	**83.44**	**87.06**	85.97	**88.93**	88.44
Swedish	76.39	**79.18**	**83.43**	83.24	**83.18**	82.04
Turkish	71.70	**76.32**	75.32	**76.34**	**79.52**	78.28
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**	**Words + POS + Morph**	**Chars + POS + Morph**
Arabic	82.05	**83.41**	**83.46**	83.40	83.83	83.11
Basque	66.61	**79.09**	73.56	**78.61**	**80.83**	79.92
French	79.22	**80.92**	**82.03**	81.08	**82.65**	81.97
German	79.15	**84.04**	**84.62**	84.49	84.79	**84.86**
Hebrew	68.71	**71.26**	**72.70**	72.26	**73.67**	72.06
Hungarian	61.93	**75.19**	69.31	**76.34**	**76.66**	76.45
Korean	67.50	**86.27**	83.37	**86.21**	83.41	**86.25**
Polish	63.96	**76.84**	**79.83**	78.24	**82.67**	82.21
Swedish	67.69	**71.19**	**76.40**	74.47	**76.42**	74.41
Turkish	54.55	**64.34**	61.22	**62.28**	**70.93**	68.28
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**	–	–

excluding part-of-speech tags to see the effect of the character-based representations when they are combined with pretrained word embeddings. The main conclusion is that when we combine character-based representations with pretrained word embeddings and part-of-speech tags, they do not have the same effect as when we only apply character-based embeddings; however, the model learns better when the part-of-speech tags or the pretrained word embeddings are not included. And finally, the model that incorporates character-based embeddings and pretraining is a bit better than the model that uses pretrained embeddings and part-of-speech tags.

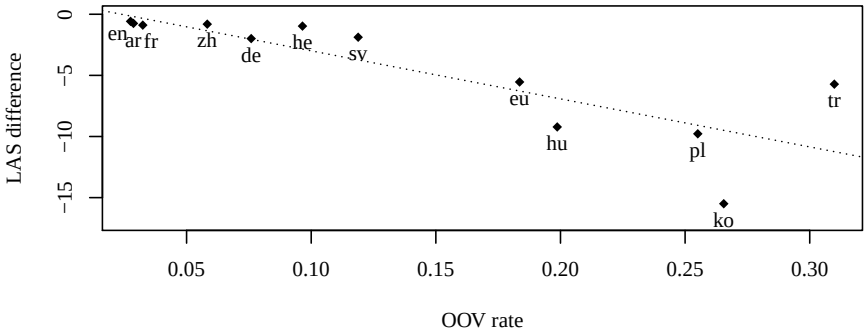

Figure 10
On the *x*-axis is the OOV rate in development data, by treebank; on the *y*-axis is the difference in development-set LAS between **Chars** model as described in Section 4.2 and one in which all OOV words are given a single representation.

Table 4
Unlabeled attachment scores and labeled attachment scores on the **development** sets (top) and the final **test** sets (bottom) for Chinese and English (SD). The columns marked with **Words** show the results of the parser with the whole S-LSTM model presented in Section 6.3, and the column marked with **Words + POS** also includes part-of-speech tags. The columns marked with **Words + Chars** show the results of the parser when combining the pretrained word embeddings of Section 6.3 with the character-based representations of this section and the columns marked with **Words + Chars + POS** also includes part-of-speech tags.

Development:

	Words		**Words + POS**		**Words + Chars**		**Words + Chars + POS**	
Language	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Chinese	82.65	79.84	**87.23**	**85.87**	82.20	79.26	86.93	85.51
English	92.58	89.96	**93.11**	**90.80**	92.74	90.35	92.87	90.49

Test:

	Words		**Words + POS**		**Words + Chars**		**Words + Chars + POS**	
Language	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Chinese	82.15	79.04	86.85	85.36	81.90	78.81	**86.92**	**85.49**
English	92.57	90.21	**93.04**	**90.87**	92.56	90.38	92.75	90.62

6.4.3 Comparison with State-of-the-Art Parsers. Table 5 shows a comparison with state-of-the-art parsers. We include greedy transition-based parsers that, like ours, do not apply beam search. 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. (2006), who also carried out a careful manual morphological feature selection. Our parser outperforms these in most cases. For English and Chinese, we report our results of the best parser on the develpment set in Section 6.3—which is **Words + POS** but with pretrained word embeddings.

We also show the best reported results on these data sets. For the SPMRL data sets, the best performing system of the shared task is either Björeklund et al. (2013) or Björkelund et al. (2014), which are better than our system. Note that the comparison is harsh to our system, which does not use unlabeled data nor any combination of different parsers. For Turkish, our system provides the best results ever reported in this data

Table 5
Test-set performance of our best results (according to UAS or LAS, whichever has the larger difference), compared with state-of-the-art greedy transition-based parsers ("Best Greedy Result") and best results reported ("Best Published Result"). Note that B+'13 and B+'14 are a combination of parsers and use unlabeled data; our models do not. We only use unlabeled data for the pretrained word embeddings of *DWPP*. **W** refers to **Words**; **C** refers to **Chars**; **WP** refers to **Words + Pos**; **WPM** refers to **Words + Pos + Morph**; **CP** refers to **Chars + Pos**; **CPM** refers to **Chars + Pos + Morph**; B'13 is (Ballesteros 2013); N+'06a is (Nivre et al. 2006); **DWPP** is our model **Word + Pos** with pretrained embeddings and dynamic oracles as in Section 6.5; **WPM** is our model **Word + Pos + Morph**; B+'13 is (Björkelund et al. 2013); B+'14 is (Björkelund et al. 2014); W+'15 is (Weiss et al. 2015).

	This Work			Best Greedy Result			Best Published Result		
Language	**UAS**	**LAS**	**System**	**UAS**	**LAS**	**System**	**UAS**	**LAS**	**System**
Arabic	86.35	83.83	**WPM**	84.57	81.90	B'13	88.32	86.21	B+'13
Basque	86.87	80.63	**WPM**	84.33	78.58	B'13	89.96	85.70	B+'14
French	86.81	82.65	**WPM**	83.35	77.98	B'13	89.02	85.66	B+'14
German	87.48	84.86	**CPM**	85.38	82.75	B'13	91.64	89.65	B+'13
Hebrew	81.36	73.57	**WPM**	79.89	73.01	B'13	87.41	81.65	B+'14
Hungarian	81.07	76.66	**WPM**	83.71	79.63	B'13	89.81	86.13	B+'13
Korean	88.39	86.27	**C**	85.72	82.06	B'13	89.10	87.27	B+'14
Polish	88.93	79.83	**WPM**	85.80	79.89	B'13	91.75	87.07	B+'13
Swedish	83.43	76.40	**WP**	83.20	75.82	B'13	88.48	82.75	B+'14
Turkish	79.52	70.93	**WPM**	75.82	65.68	N+'06a	79.52	70.93	**WPM**
Chinese	85.96	84.40	**WP**	87.65	86.21	**DWPP**	87.65	86.21	**DWPP**
English	92.57	90.31	**WP**	93.56	91.42	**DWPP**	94.08	92.19	W+'15

set. For English, we report Weiss et al. (2015), and for Chinese, we report our results in Section 6.3—which is **Words + POS** but with pretrained word embeddings and dynamic oracles (see next section).

6.5 Experiments with Dynamic Oracles

Table 6 shows the results of the parser with arc-hybrid and dynamic oracles for the English data set (with Stanford dependencies) and the Chinese CTB data sets with and without pretrained word embeddings, respectively. The table also shows the best result reported in Section 6.3 for the sake of comparison between static and dynamic training strategies.

The results achieved by the dynamic oracle for English are actually one of the best results ever reported in this data set, achieving 93.56 unlabeled attachment score. This is remarkable given that the parser uses a completely greedy inference procedure. Moreover, the Chinese numbers establish the state-of-the-art by using the same settings as in Chen and Manning (2014).

The error-exploring dynamic-oracle training always improves above static oracle training using the same transition system, but the arc-hybrid system slightly underperforms the arc-standard system when trained with static oracle. Transforming the sampling distribution towards preferring less probable events ($\alpha = 0.75$) is especially beneficial when training with pretrained word embeddings.

Table 6
Unlabeled attachment scores and labeled attachment scores on the **development** sets (top) and the final **test sets** (bottom) for Chinese and English (SD). The columns marked with "Arc-std." show the results of the parser arc-standard oracle, which are the same numbers as in Table 1. The columns marked with "Arc-hybrid" show the results of the parser with the arc-hybrid oracle. The columns marked with "static" ("dynamic") show the results of the parser with static (dynamic) oracles.

Development								
	Arc-std. (static)		Arc-hybrid (static)		Arc-hybrid (dynamic)		Arc-hybrid (dynamic, $\alpha = 0.75$)	
Language	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Chinese (–pretraining)	85.96	84.37	82.65	79.84	85.96	84.37	**86.32**	**84.59**
Chinese (+pretraining)	87.23	85.87	87.11	85.64	**87.41**	**85.99**	**87.41**	85.87
English (–pretraining)	92.58	90.24	92.64	90.26	93.01	90.68	**93.04**	**90.64**
English (+pretraining)	93.11	90.80	93.16	90.88	93.38	91.03	**93.51**	**91.29**

Test								
	Arc-std. (static)		Arc-hybrid (static)		Arc-hybrid (dynamic)		Arc-hybrid (dynamic, $\alpha = 0.75$)	
Language	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Chinese (–pretraining)	85.48	83.94	85.66	84.03	86.07	84.46	**86.13**	**84.53**
Chinese (+pretraining)	86.85	85.36	86.94	85.46	87.05	85.63	**87.65**	**86.21**
English (–pretraining)	92.40	90.04	92.08	89.80	92.66	90.43	**92.73**	**90.60**
English (+pretraining)	93.04	90.87	92.78	90.67	93.15	91.05	**93.56**	**91.42**

6.6 Experiments with CoNLL 2009 Data Sets

In order to be able to compare with similar greedy parsers (Yazdani and Henderson 2015; Andor et al. 2016)[21] we report the performance of the parser on the multilingual treebanks of the CoNLL 2009 Shared Task (Hajič et al. 2009). Because some of the treebanks contain nonprojective sentences and arc-hybrid does not allow nonprojective trees, we use the pseudo-projective approach (Nivre and Nilsson 2005). For all the experiments presented in this section we used the predicted part-of-speech tags provided by the CoNLL 2009 shared task organizers. In order to see if the pretrained word embeddings are useful for other languages we also include results with pretrained word embeddings for English, Chinese, German, and Spanish following the same training set-up as in Section 4; for English and Chinese we used the same pretrained word embeddings as in previous experiments, for German we pretrained embeddings using the monolingual training data from the WMT 2015 data set[22], and for Spanish we used the Spanish Gigaword version 3.

The results for the parser with character-based representations on these data sets (last line of the table) were published by Andor et al. (2016). In Zhang and Weiss (2016), it is also possible to find results of the same version of the parser on the Universal Dependency treebanks (Nivre et al. 2015).

21 We report the performance of these parsers in the most comparable set-up, that is, with beam = 1 or greedy.

22 `http://www.statmt.org/wmt15/translation-task.html`.

Table 7
Results of the parser in its different versions including comparison with other systems. St. refers to static oracle with the arc-standard parser and Dyn. refers to dynamic oracle with the arc-hybrid parser with $\alpha = 0.75$ because it is the top scoring system in Section 6.5. PP refers to pseudo-projective parsing, SW refers to the arc-standard algorithm including the SWAP action as described in Section 3.3.2. +pre refers to the models that incorporate pretrained word embeddings. The Chinese treebank is fully projective and this is why we do not report results with SWAP since they are equivalent to the ones with pseudo-projective parsing. Y'15 is the parser by Yazdani and Henderson (2015). A'16 is the parser with beam = 1 by Andor et al. (2016). A'16-b is the parser with beam larger than 1 by Andor et al. (2016). **Bold** numbers indicate the best results among the greedy parsers.

	Catalan		Chinese		Czech		English		German		Japanese		Spanish	
Method	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
St + PP	89.60	85.45	79.68	75.08	77.96	71.06	91.12	88.69	88.09	85.24	93.10	92.28	89.08	85.03
+ pre	–	–	82.45	78.55	–	–	91.59	89.15	88.56	86.15	–	–	90.76	87.48
St + SW	89.55	85.35	–	–	78.66	71.99	91.12	88.50	88.11	85.48	93.29	92.46	89.02	84.96
+ pre	–	–	–	–	–	–	91.53	89.14	89.21	86.96	–	–	90.43	86.99
Dyn + PP	90.45	86.38	80.74	76.52	85.68	79.38	91.62	89.23	89.80	87.29	93.47	92.70	89.53	85.69
+ pre	–	–	**83.54**	**79.66**	–	–	**92.22**	**89.87**	**90.34**	**88.17**	–	–	**91.09**	87.95
Y'15	–	–	–	–	85.2	77.5	90.13	87.26	89.6	86.0	–	–	88.3	85.4
A'16	**91.24**	**88.21**	81.29	77.29	**85.78**	**80.63**	91.44	89.29	89.12	86.95	**93.71**	**92.85**	91.01	**88.14**
A'16-b	92.67	89.83	84.72	80.85	88.94	84.56	93.22	91.23	90.91	89.15	93.65	92.84	92.62	89.95

Our parsers outperform the model by Yazdani and Henderson (2015). When comparing with Andor et al. (2016), it is worth noting that they use pretrained word embeddings, plus predicted morphological features, and the top-*K* POS tags from a CRF tagger instead of a simple predicted tags, so the results that are actually comparable are the ones in which we include pretrained word embeddings; our model is better than Andor et al. (2016), especially in the case of the dynamic oracles.

The results with Czech and the dynamic oracle model suggest that it is a much better strategy when the number of nonprojective trees is high; this result suggests that the training with exploration allows the parser to handle more complicated syntactic structures. Finally, the arc-standard parser enriched with the SWAP action performs similarly to the pseudo-projective approach for all languages, which demonstrates the effectiveness of both approaches when parsing nonprojective structures.

7. Related Work

Our approach ties together several strands of previous work. First, several kinds of stack memories have been proposed to augment neural architectures. Das, Giles, and Sun (1992) proposed a neural network with an external stack memory based on recurrent neural networks. In contrast to our model, in which the entire contents of the stack are summarized in a single value, in their model the network could only see the contents of the top of the stack. Mikkulainen (1996) proposed an architecture with a stack that had a summary feature, although the stack control was learned as a latent variable.

Several authors have used neural networks to predict parser actions in shift-reduce parsers. The earliest attempt we are aware of is Mayberry and Miikkulainen (1999). The recent resurgence of interest in neural networks has included several applications to transition-based dependency parsers (Andor et al. 2016; Yazdani and Henderson 2015; Weiss et al. 2015; Zhou et al. 2015; Chen and Manning 2014; Stenetorp 2013; Titov and Henderson 2007b). In most of these works, the conditioning structure is

manually crafted and sensitive to only certain properties of the state, with the exception of Titov and Henderson (2007b), whereas we are conditioning on the global state. Like us, Stenetorp (2013) used recursively composed representations of the tree fragments (a head and its dependents). Titov and Henderson (2007b) used a generative latent variable based on incremental sigmoid belief networks to likewise condition on the global state. Neural networks have also been used to learn representations for use in phrase-structure parsing (Henderson 2003, 2004; Titov and Henderson 2007a; Socher et al. 2013; Le and Zuidema 2014). The work by Watanabe et al. (2015) is also similar to the work presented in this article but for phrase-structure parsing.

Our model's position relative to existing neural network parsers can be understood by analogy to neural language models. The context representation is either constructed subject to an n-gram window (Bengio et al. 2003; Mnih and Hinton 2007), or it may be constructed incrementally using recurrent neural networks to obtain potentially unbounded dependencies (Mikolov et al. 2010). Likewise, during parsing, the particular algorithm state (represented by the stack, buffer, and history of actions taken) is the context used to predict the parsing action. This may be represented as a finite vector by picking out certain features (Weiss et al. 2015; Chen and Manning 2014) or by recursively composing the representation of simpler elements as parsing proceeds (Henderson 2003, 2004; Titov and Henderson 2007b; this work).

LSTMs have recently been demonstrated as a mechanism for generating phrase structure parses. Vinyals et al. (2015) used a sequence-to-sequence architecture with attention to predict a sequence of tree building actions, conditional on the sentence being parsed. Other works have used LSTMs for better semantic representations (Tai, Socher, and Manning 2015), and others to solve similar tasks such as phrase-structure parsing (Dyer et al. 2016).

In terms of token representations, it is worth noting that 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, obtaining an error reduction in both tasks. Their approach is similar to ours. Many approaches have used character-based models as additional features to improve existing models. For instance, Chrupala (2014) tried character-based recurrent neural networks to normalize tweets. Similarly, Botha and Blunsom (2014) show that stems, prefixes, and suffixes can be used to learn useful word representations. Our approach is learning similar representations by using the bidirectional LSTMs. Chen et al. (2015) propose joint learning of character and word embeddings for Chinese, claiming that characters contain rich internal information. Constructing word representations from characters using convolutional neural networks has also been proposed (Kim et al. 2016).

Tsarfaty (2006), Cohen and Smith (2007), Goldberg and Tsarfaty (2008), and Goldberg and Elhadad (2011) presented methods for joint segmentation and phrase-structure parsing, by segmenting the words in useful morphologically oriented units. 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.

Training greedy parsers on non-gold outcomes, facilitated by dynamic oracles, has been explored by several researchers in different ways (Goldberg and Nivre 2012, 2013; Goldberg, Sartorio, and Satta 2014; Honnibal, Goldberg, and Johnson 2013; Honnibal and Johnson 2014; Gómez-Rodríguez, Sartorio, and Satta 2014; Björkelund and Nivre 2015; Tokgöz and Eryiğit 2015; Gómez-Rodríguez and Fernández-González 2015; Vaswani and Sagae 2016). More generally, training greedy inference systems by

paying attention to the expected classifier behavior during test time has been explored under the imitation learning and Learning to Search (SEARN) frameworks (Abbeel and Ng 2004; Daumé III and Marcu 2005; Vlachos 2012; He, Eisner, and Daume 2012; Daumé III, Langford, and Marcu 2009; Ross, Gordon, and Bagnell 2011). In the context of recurrent neural networks, the discrepancy between training time and test time prediction was recently explored by Bengio et al. (2015) for the task of language modeling. However, to the best of our knowledge this is the first time that such a training procedure is explored together with neural networks for parsing. The results show that the consistent gains in accuracy due to dynamic oracle training that were observed in previous work persist also for the very strong stack LSTM parser, although some care had to be taken to adapt the training procedures to the probabilistic nature of the objective, and to the high accuracy levels of the underlying model.

We conclude by remarking that stack memory offers intriguing possibilities for learning to solve general information processing problems (Mikkulainen 1996). Here, we learned from observable stack manipulation operations (i.e., supervision from a treebank), and the computed embeddings of final parser states were not used for any further prediction.

Such an extension of the work would make it an alternative to architectures that have an explicit external memory such as neural Turing machines (Graves, Wayne, and Danihelka 2014) and other memory networks (Weston, Chopra, and Bordes 2014; Kumar et al. 2015). However, as with those models, without supervision of the stack operations, formidable computational challenges must be solved (e.g., marginalizing over all latent stack operations), but sampling techniques and techniques from reinforcement learning have promise here (Zaremba and Sutskever 2015), making this an intriguing avenue for future work.

8. Conclusions

We presented stack LSTMs, recurrent neural networks for sequences, with push and pop operations, and used them to implement a state-of-the-art transition-based dependency parser. The parser uses a greedy learning strategy which potentially provides very high parsing speed while still achieving state-of-the-art results.

We demonstrate that word representations "composed" from representations of characters are sufficient for successful transition-based dependency parsing. The results are stronger for agglutinative languages, such as Basque, Hungarian, Korean, and Turkish. This outcome is important, because annotating morphological information is expensive; our model suggests that only dependency annotations are necessary and relevant morphological features can be learned from strings. Moreover, character-based representations are also a way of overcoming the out-of-vocabulary problem even in morphologically simpler languages, such as English. Additionally, inspired by Bohnet et al. (2013), an orthogonal experiment to the one presented in this paper would be to apply character-based representations to joint tagging and dependency parsing. We expect the character-based representations to be powerful enough to catch morphosyntactic information in a joint model. Similar improvements may be achieved in an out-of-domain scenario.

Even though the parser is greedy, it provides very consistent results comparable with the best parsers of the state-of-the-art. We even obtained further improvement as demonstrated with the experiments with the dynamic oracles, that provide a push over the baseline while still maintaining very fast (and greedy) parsing speed.

Finally, it is worth noting that a modified version of the same parser has been used to create a polyglot dependency parser (Ammar et al. 2016) with the universal dependency treebanks (Nivre et al. 2015), and the stack LSTMs have also been applied to solve other core natural language processing tasks, such as named entity recognition (Lample et al. 2016), phrase-structure parsing and language modeling (Dyer et al. 2016), and joint syntactic and semantic parsing (Swayamdipta et al. 2016).

Acknowledgments

We would like to thank Lingpeng Kong and Jacob Eisenstein for comments on an earlier version of this article and Danqi Chen for assistance with the parsing data sets. We would also like to thank Bernd Bohnet and Joakim Nivre for their help with the parsing algorithms. This work was sponsored in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-10-1-0533, and in part by NSF CAREER grant IIS-1054319. Miguel Ballesteros is supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA). Yoav Goldberg is supported by The Israeli Science Foundation (grant number 1555/15) and a Google Research Award.

References

Abbeel, Pieter and Andrew Y. Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–8, New York, NY.

Abeillé, Anne, Lionel Clément, and François Toussenel. 2003. *Building a Treebank for French*. Springer Netherlands, Dordrecht, pages 165–187.

Aduriz, I., M. J. Aranzabe, J. M. Arriola, A. Atutxa, A. Díaz de Ilarraza, A. Garmendia, and M. Oronoz. 2003. Construction of a Basque dependency treebank. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories*, pages 201–204, Växjö.

Ammar, Waleed, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. Arxiv.1603.06042.

Ballesteros, Miguel. 2013. Effective morphological feature selection with MaltOptimizer at the SPMRL 2013 shared task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 63–70, Seattle, WA.

Ballesteros, Miguel and Bernd Bohnet. 2014. Automatic feature selection for agenda-based dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 794–805, Dublin.

Ballesteros, Miguel, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon.

Ballesteros, Miguel and Joakim Nivre. 2013. Going to the roots of dependency parsing. *Computational Linguistics*, 39(1):5–13.

Ballesteros, Miguel and Joakim Nivre. 2016. Maltoptimizer: Fast and effective parser optimization. *Natural Language Engineering*, 22:187–213, 3.

Bansal, Mohit, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 809–815, Baltimore, MD.

Bengio, Samy, Oriol Vinyals, Navdeep Jaitly, and Noam M. Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems, 28*. Curran Associates, Inc., pages 1171–1179.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Björkelund, Anders, 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 *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 97–102, Dublin.

Björkelund, Anders, 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 *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145, Seattle, WA.

Björkelund, Anders and Joakim Nivre. 2015. Non-deterministic oracles for unrestricted non-projective transition-based dependency parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86, Bilbao.

Bohnet, Bernd and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju Island.

Bohnet, Bernd, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics (TACL)*, 1:415–428.

Botha, Jan A. and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014*, pages 1899–1907, Beijing.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York, NY.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, MI. Association for Computational Linguistics.

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

Chen, Wenliang, Yue Zhang, and Min Zhang. 2014. Feature embedding for dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 816–826, Dublin.

Chen, Xinxiong, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huan-Bo Luan. 2015. Joint learning of character and word embeddings. In Qiang Yang and Michael Wooldridge, editors, *IJCAI*, pages 1236–1242. AAAI Press.

Choi, Jinho D. 2013. Preparing Korean data for the shared task on parsing morphologically rich languages. ArXiv.1309.1649.

Choi, Jinho D. and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia.

Chrupała, Grzegorz. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 680–686, Baltimore, MD.

Cohen, Shay B. and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 208–217, Prague.

Das, Sreerupa, C. Lee Giles, and Guo zheng Sun. 1992. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Conference of the Cognitive Science Society*. Morgan Kaufmann Publishers, pages 791–795.

Daumé III, Hal, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75:297–325.

Daumé III, Hal and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the*

International Conference on Machine Learning (ICML), pages 169–176, Bonn.
dos Santos, Cicero and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. In *Proceedings of the Fifth Named Entity Workshop*, pages 25–33, Beijing.
dos Santos, Cícero.Nogueira and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014*, pages 1818–1826, Beijing.
Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing.
Dyer, Chris, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, CA.
Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. 2003. Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, 3:115–143.
Glorot, Xavier and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010*, pages 249–256, Sardinia.
Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, pages 315–323, Ft. Lauderdale, FL.
Goldberg, Yoav. 2013. Dynamic-oracle transition-based parsing with calibrated probabilistic output. In *Proceedings of International Conference on Parsing Technologies (IWPT)*, pages 82–90, Nara.
Goldberg, Yoav and Michael Elhadad. 2011. Joint Hebrew segmentation and parsing using a PCFG-LA lattice parser. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 704–709, Portland, OR.
Goldberg, Yoav and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai.
Goldberg, Yoav and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics (TACL)*, 1:403–414.
Goldberg, Yoav, Francesco Sartorio, and Giorgio Satta. 2014. A tabular method for dynamic oracles in transition-based parsing. *Transactions of the Association for Computational Linguistics*, 2:120–130.
Goldberg, Yoav and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proceedings of ACL-08: HLT*, pages 371–379, Columbus, OH.
Goldberg, Yoav, Kai Zhao, and Liang Huang. 2013. Efficient implementation of beam-search incremental parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 628–633, Sofia.
Gómez-Rodríguez, Carlos and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 256–261, Beijing.
Gómez-Rodríguez, Carlos, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927, Doha.
Graves, Alex. 2013. Generating sequences with recurrent neural networks. ArXiv.1308.0850.
Graves, Alex, Santiago Fernández, and Jürgen Schmidhuber. 2007. Multi-dimensional recurrent neural networks. In *Artificial Neural Networks (ICANN)*. Springer, pages 549–558.
Graves, Alex and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610.
Graves, Alex, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing machines. ArXiv.1410.5401.
Grefenstette, Edward, Karl Moritz Hermann, Georgiana Dinu, and Phil Blunsom. 2014.

New directions in vector space models of meaning. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: Tutorials*, page 8, Baltimore, MD.

Grefenstette, Edward, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., pages 1828–1836.

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

Hajič, Jan, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, CO.

He, He, Jason Eisner, and Hal Daume. 2012. Imitation learning by coaching. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., pages 3149–3157.

Henderson, James. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 24–31, Edmonton.

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

Hermann, Karl Moritz and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Proceedings of the ACL*, pages 1–11, Sofia.

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

Honnibal, Matthew, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning (CoNLL)*, pages 163–172, Sofia.

Honnibal, Matthew and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics (TACL)*, 2:131–142.

Joulin, Armand and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. ArXiv.1503.01007.

Kim, Yoon, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2741–2749, Phoenix, AZ.

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

Kumar, Ankit, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. ArXiv.1506.07285.

Lample, Guillaume, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, CA.

Le, Phong and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 729–739, Doha.

LeCun, Yann, L. Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 11:2278–2324.

Ling, Wang, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015a. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, pages 1299–1304, Denver, CO.

Ling, Wang, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015b. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon.

Maamouri, Mohamed, 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*, pages 466–467, Cairo.

Marneffe, Marie-Catherine De, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of Language Resources and Evaluation Conference (LREC)*, pages 449–454, Genoa.

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

Mikkulainen, Risto. 1996. Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, 20:47–73.

Mikolov, Tomas, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048, Makuhari.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 3111–3119.

Mnih, Andriy and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 641–648, New York, NY.

Mueller, Thomas, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, WA.

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

Nivre, Joakim. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, IncrementParsing '04, pages 50–57, Stroudsburg, PA.

Nivre, Joakim. 2007. Incremental non-projective dependency parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 396–403, Rochester, NY.

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

Nivre, Joakim. 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, Singapore.

Nivre, Joakim, Željko Agić, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, et al. 2015. Universal dependencies 1.2. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague.

Nivre, Joakim, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225, New York, NY.

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106, Ann Arbor, MI.

Nivre, Joakim, Jens Nilsson, and Johan Hall. 2006. Talbanken05: A Swedish treebank with phrase structure and dependency annotation. In *Proceedings of Language*

Resources and Evaluation Conference (LREC), pages 1392–1395, Genoa.

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

Pascanu, Razvan, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2013. How to construct deep recurrent neural networks. ArXiv.1312.6026.

Paulus, Romain, Richard Socher, and Christopher D. Manning. 2014. Global belief recursive neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pages 2888–2896.

Riezler, Stefan, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 480–487, Hong Kong.

Ross, Stéphane, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, pages 627–635, Ft. Lauderdale, FL.

Seddah, Djamé, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin.

Seddah, Djamé, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, et al. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, WA.

Seeker, Wolfgang and Jonas Kuhn. 2012. Making ellipses explicit in dependency conversion for a German treebank. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3132–3139, Istanbul.

Sima'an, Khalil, Alon Itai, Yoad Winter, Alon Altman, and Noa Nativ. 2001. Building a tree-bank for modern Hebrew text. In *Traitement Automatique des Langues*, 42:347–380.

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

Socher, Richard, Eric H. Huang, Jeffrey Pennin, Christopher D. Manning, and Andrew Y. Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., pages 801–809.

Socher, Richard, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. 2013a. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, pages 207–218.

Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, WA.

Stenetorp, Pontus. 2013. Transition-based dependency parsing using recursive neural networks. In *Proceedings of the NIPS Deep Learning Workshop*, pages 1–9, Lake Tahoe, CA.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pages 3104–3112.

Swayamdipta, Swabha, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, joint syntactic-semantic parsing with stack LSTMs. In *Proceedings of the 53rd Annual Meeting of the Conference of Natural Language Learning (CoNLL)*. Berlin.

Świdziński, Marek and Marcin Woliński. 2010. Towards a bank of constituent parse trees for Polish. In *Text, Speech and*

Dialogue: 13th International Conference (TSD), pages 197–204, Brno.

Tai, Kai Sheng, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing.

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

Titov, Ivan and James Henderson. 2007b. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155, Prague.

Tokgöz, Alper and Gülşen Eryiğit. 2015. Transition-based dependency dag parsing using dynamic oracles. In *Proceedings of the Student Research Workshop of the Association for Computational Linguistics (ACL)*, pages 22–27, Beijing.

Toutanova, Kristina, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003, Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the North American Chapter of the Association for Computational Linguistiscs – Human Language Technologies (NAACL–HLT)*, pages 173–180, Edmonton.

Tsarfaty, Reut. 2006, Integrated morphological and syntactic disambiguation for modern Hebrew. In *Proceedings of the COLING/ACL 2006 Student Research Workshop*, pages 49–54, Sydney.

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

Vaswani, Ashish and Kenji Sagae. 2016. Efficient structured inference for transition-based parsing with neural networks and error states. *Transactions of the Association for Computational Linguistics*, 4:183–196.

Vincze, Veronika, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin and János Csirik. 2010. Hungarian dependency treebank. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta.

Vinyals, Oriol, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., pages 2773–2781.

Vlachos, Andreas. 2012. An investigation of imitation learning algorithms for structured prediction. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, pages 143–154, Edinburgh.

Watanabe, Taro and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1169–1179, Beijing.

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

Weston, Jason, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. ArXiv.1410.3916.

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

Yazdani, Majid and James Henderson. 2015. Incremental recurrent neural network dependency parser with search-based discriminative training. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 142–152, Beijing.

Yogatama, Dani, Lingpeng Kong, and Noah A. Smith. 2015. Bayesian optimization of text representations. In *Proceedings of the 2015 Conference on*

Empirical Methods in Natural Language Processing, pages 2100–2105, Lisbon.

Yogatama, Dani and Gideon Mann. 2014. Efficient transfer learning method for automatic hyperparameter tuning. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 1077–1085, Reykjavik.

Zaremba, Wojciech and Ilya Sutskever. 2015. Reinforcement learning neural turing machines. ArXiv.1505.00521.

Zhang, Yuan and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. ArXiv.1603.06598.

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

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

Zhou, Hao, Yue Zhang, Shujian Huang and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing.