Efficient Structured Inference for Transition-Based Parsing with Neural Networks and Error States

Ashish Vaswani*
USC Information Sciences Institute
vaswani@usc.edu

Kenji Sagae*
USC Institute for Creative Technologies
sagae@usc.edu

Abstract

Transition-based approaches based on local classification are attractive for dependency parsing due to their simplicity and speed, despite producing results slightly below the state-of-the-art. In this paper, we propose a new approach for approximate structured inference for transition-based parsing that produces scores suitable for *global* scoring using *local* models. This is accomplished with the introduction of *error states* in local training, which add information about incorrect derivation paths typically left out completely in locally-trained models. Using neural networks for our local classifiers, our approach achieves 93.61% accuracy for transition-based dependency parsing in English.

1 Introduction

Transition-based parsing approaches based on local classification of parser actions (Nivre, 2008) remain attractive due to their simplicity, despite producing results slightly below the state-of-the-art. Although the application of online structured prediction and beam search has made transition-based parsing competitive in accuracy (Zhang and Clark, 2008; Huang et al., 2012) while retaining linear time complexity, greedy inference with locally-trained classifiers is still widely used, and techniques for improving the performance of greedy parsing have been proposed recently (Choi and Palmer, 2011; Goldberg and Nivre, 2012; Goldberg and Nivre, 2013; Honnibal et al., 2013). Recent work on the application of neural network classification to drive greedy transition-based dependency parsing has achieved high accuracy (Chen and Manning, 2014), showing how effective locally-trained neural network models are at predicting parser actions, while providing a straightforward way to improve parsing accuracy using word embeddings pre-trained using a large set of unlabeled data.

We propose a novel approach for approximate structured inference for transition-based parsing that uses locally-trained neural networks that, unlike previous local classification approaches, produce scores suitable for global scoring. This is accomplished with the introduction of *error states* in local training, which add information about incorrect derivation paths typically left out completely in locally-trained models. Our approach produces high accuracy for transition-based dependency parsing in English, surpassing parsers based on the structured perceptron (Huang and Sagae, 2010; Zhang and Nivre, 2011) by allowing seamless integration of pre-trained word embeddings, while requiring nearly none of the feature engineering typically associated with parsing with linear models. Trained without external resources or pre-trained embeddings, our neural network (NN) dependency parser outperforms the NN transition-based dependency parser from Chen and Manning (2014), which uses pre-trained word embeddings trained on external data and more features, thanks to improved search. Our experiments show that naive search produces very limited improvements in accuracy compared to greedy inference, while search in conjunction with error states that mark incorrect derivations produces substantial accuracy improvements.

2 Background: Transition-Based Parsing

Transition-based approaches are attractive in dependency parsing for their algorithmic simplicity and straightforward data-driven application. Using shift-

*Both authors contributed equally to this paper.

Transactions of the Association for Computational Linguistics, vol. 4, pp. 183–196, 2016. Action Editor: Marco Kuhlmann.
Submission batch: 5/2015; Revision batch: 8/2015; Published 5/2016.

reduce algorithms, such as those pioneered by Nivre (2008), the task of finding a dependency structure becomes that of predicting each action in the derivation of desired structure.

2.1 Arc-Standard Dependency Parsing

Our parsing models are based on a simple shift-reduce algorithm for dependency parsing known as the arc-standard dependency parsing algorithm (Nivre, 2008). An arc-standard dependency parser maintains one or more parser states T, each composed of a stack $S = [s_m, ..., s_1, s_0]$ (where the topmost item is s_0) and input buffer $W = [w_0, w_1, ..., w_n]$ (where the first element of the buffer is w_0). In its initial state T_0, the stack is empty, and the input buffer contains each token in the input sentence with its part-of-speech tag. One of three actions can be applied to a parser state T_i to create a new parser state T_j: *shift*, which takes the next word in the input buffer (with its part-of-speech tag), and places it as a tree with a single node on top of the stack (i.e. input token w_0 is consumed to create the new stack item s_0); *reduce-right*, which pops the top two items on the stack, s_0 and s_1, and pushes onto the stack a new subtree formed by attaching the root node of s_0 as a dependent of s_1; and *reduce-left*, which pops the top two items on the stack, s_0 and s_1, and pushes onto the stack a new subtree formed by attaching the root node of s_1 as a dependent of s_0. An alternative formulation keeps only word indices in the stack and input buffer, and includes an additional set of dependency arcs; the two formulations are equivalent.

A greedy arc-standard parser keeps only one parser state, choosing at each step one parser action to apply to the current state, which is replaced once application of the chosen action creates the next state. Once the current state is a final state, parsing terminates. A final state is one where the input buffer is empty, and the stack contains only one element, which is the dependency tree output. Given a way to score parser actions instead of simply choosing one action to apply, a state score can be defined on the sequence of actions resulting in the state. Keeping track of multiple states with scores resulting from the application of different valid actions for a single state creates an exponential search space. Beam search can then be applied to search for a high scoring state. With global estimation of parameters for scoring parser actions, a beam search can produce more accurate results than greedy parsing by minimizing global loss (Zhang and Clark, 2008).

2.2 Local Classification

Initial data-driven transition-based dependency parsing approaches employed locally-trained multi-class models to choose a parser action based on the parser state at each step in the derivation (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004). In these models, classification is based on a set of features extracted from the current state of the parser, and creating training examples for the classifier requires only running the transition-based algorithm to reproduce the trees in a training treebank, while recording the features and actions at each step. A classifier is then trained with the actions as classes.

While this simple procedure has allowed for training of dependency parsers using off-the-shelf classifier implementations, the resulting parsers are restricted to performing greedy search, considering only one tree out of the exponentially many. Although the distribution of class scores for each parser state can be used to create a search space for beam search, the locally normalized scores obtained with these classifiers make searching a largely futile endeavor, since action scores cannot be combined meaningfully to score entire trees or entire derivations. For example, Zhao et al. (2013) use maximum entropy classification for local classification of shift-reduce parsing actions with a dynamic programming approach based on the work of Huang and Sagae (2010). Despite using exact search, Zhao et al. report an improvement of only 0.6% in unlabeled dependency accuracy over greedy parsing, reaching an accuracy of 90.7%, far below the 92.2% obtained with a comparable structured perceptron parser with beam search and very similar features (Huang et al., 2012). Similarly, Johansson and Nugues (2006), who used probability estimates from local SVM classification to perform a beam search in transition-based parsing, report some accuracy gains when using a beam of size 2, but no further gains with larger beams. Because transition scores out of each state are normalized locally, the quality of any particular state is in no way captured by the scores that will ultimately result in the overall score for the deriva-

tion. In fact, from an incorrect parser state, more incorrect transitions may follow, due to a version of the label bias problem faced by MEMMs (Lafferty et al., 2001; McCallum et al., 2000). In Section 3, we will present our approach that significantly improves search with locally normalized models.

2.3 Structured Perceptron

One effective way to create models that score parser transitions globally and allow for effective search is to use the structured perceptron (Collins, 2002). Unlike with local classifiers, weight updates are based on entire derivations, instead of individual states. However, because exact inference is too costly for transition-based parsing with a rich feature set, in practice parsers use beam search to perform approximate inference, and care must be taken to ensure the validity of weight updates (Huang et al., 2012). A widely used approach is to employ early updates, which stop parsing and perform weight updates once the desired structure is no longer in the beam (Collins and Roark, 2004).

Transition-based dependency parsers based on the structured perceptron have reached high accuracy (Zhang and Nivre, 2011; Hatori et al., 2012), but these parsers remain in general less accurate than high-order graph-based parsers that model dependency graphs directly, instead of derivations (Zhang and McDonald, 2014; Martins et al., 2013). The drawback of these more accurate parsers is that they tend to be slower than transition-based parsers.

3 Parsing with Local Classifiers and Error States

The standard way to train local classifiers to predict actions for transition-based parsers is to run the parsing algorithm using a gold-standard sequence of actions (i.e. a sequence of actions that generates the gold-standard tree from a training set) and record the features corresponding to each parser state, where a parser state includes the parser's stack, input buffer, and set of dependencies created so far. The features corresponding to a state are then associated with the gold-standard action that should be taken from that state, and this constitutes one training example for the local action classifier. Sagae and Tsujii (2007) propose using a maximum entropy classifier to produce conditional probabilities for each action given the features of the state, and score each state using the product of the probabilities of all actions taken up to that state. However, they report that searching through the resulting space for the highest scoring parse does not consistently result in improved parser accuracy over a greedy policy (i.e. pursue only the highest scoring action at each state), suggesting that this strategy for scoring states is a poor choice. This is confirmed by Zhao et al. (2013), who report only a small improvement over greedy search despite using exact inference with this state scoring strategy.

Because action probabilities are conditioned on the features of the current state alone and normalized locally, there is no reason to expect that the product of such probabilities along a derivation path up to a state, whether or not it is a final state, should reflect the overall quality of the state. Once an incorrect action is classified as more probable than the correct action in a given state T_i, the incorrect state T_j resulting from the application of the incorrect action will have higher score than the correct state T_k resulting from the application of the correct action. From that point, the action probabilities given state T_j will sum to one, just as the action probabilities given state T_k will sum to one, and there is no reason to expect that the most probable action from T_j should be less probable than the most probable action from T_k. In other words, once an error occurs, search is of little help in recovering from it, since scores are based only on local decisions and not on any notion of state quality, and the error occurred precisely because an incorrect action was found to be more probable locally.

Our key contribution is a solution to this problem by introducing a notion of state quality in local action classification. This is done through the addition of a new *error* class to the local classification model. Unlike the other classes, the error class does not correspond to a parser action. In fact, the error class is not used at all during parsing, and serves to occupy probability mass, keeping it from the actual parser actions. Intuitively, the probability of the error class given the current state can be thought of as the probability that an error has occurred previously and the resulting state belongs to an incorrect derivation path.

3.1 Training Local Classifiers with Error States

To train a local classifier with error states, the standard way of generating classifier training examples is modified to include parser states that do not belong to the derivation of the gold-standard tree. It is these incorrect parser states that are labeled error states. Figure 1 illustrates the generation of training examples for a local action classifier with error states, assuming unlabeled arc-standard dependency parsing (Nivre, 2008), where the actions are shift, reduce-right and reduce-left. From state 2, the standard way of training local classifiers would be simply to associate features from state 2 to a shift action, generate state 3 (only), associate features from state 3 with a shift action, generate state 6, and continue in this fashion along the derivation of the gold-standard tree. To add error states, from state 2 we do not only generate state 3, but also states 4 and 5, which result from the application of incorrect actions. In addition to associating features from state 3 with shift, we associate features from state 4 with the error class, and features from state 5 with the error class. The desired effect is that any time the parser deviates from a correct derivation, the error class should become probable, while valid parser actions become less probable. Although in principle any state outside of a gold-standard derivation is an error state, we generate only error states resulting from the application of a single incorrect action, which in practice increases the number of state-action pairs used to train the classifier by approximately a factor of three. We leave an investigation of how far into incorrect derivations one should go to generate additional error states as future work.

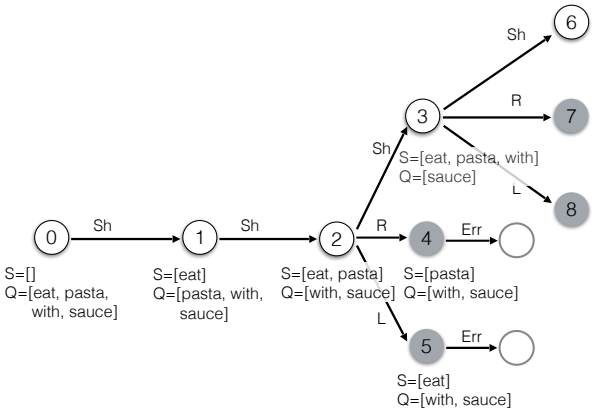

Figure 1: Training of a local classifier for parser actions with error states. In addition to collecting training examples for each of the three valid parser actions (represented as Sh for *shift*, L for *reduce-left*, and R for *reduce-right*), we collect also examples of an *error* class (Err), which corresponds to the shaded states generated after taking an incorrect action.

3.2 Parsing with Error States

Once a local classifier has been trained with error states, this classifier can be used in a transition-based parser with no modifications; the error class is simply thrown away during parsing. For example, the type of beam search typically used in transition-based parsing with the structured perceptron (Zhang and Clark, 2008; Huang and Sagae, 2010) can be used to pursue several derivations in parallel, and global score of a derivation can be decomposed as the sum of the scores of all actions in the derivation. Analogously, we score each derivation using the product of the probabilities for all actions in the derivation. Interestingly, local normalization of action scores allows the use of best-first search (Sagae and Tsujii, 2007), which has the potential to arrive at high quality solutions without having to explore as many states as a typical beam search, and even allows efficient exact or nearly exact inference (Zhao et al., 2013). Once actions are scored for the parser's current state using a classifier, the score of a new state resulting from the application of a valid action to the current state can be computed as the product of the probabilities of all actions applied up to the new state in its derivation path. In other words, the score of each new state is the score of the current state multiplied by the probability of the action applied to the current state to generate the new state. New scored states resulting from the application of each action to the current state are then placed in a priority queue. The highest scoring item in the priority queue is chosen, and the state corresponding to that item is then made the current state[1]. The local classifier is then applied to the current state, and

[1]For efficiency, items inserted in the priority queue could be simply *state scores* coupled with the corresponding action and a pointer to the current state, since the new state only needs to be generated once it becomes the current state, and often only a fraction of priority queue items ever become current states.

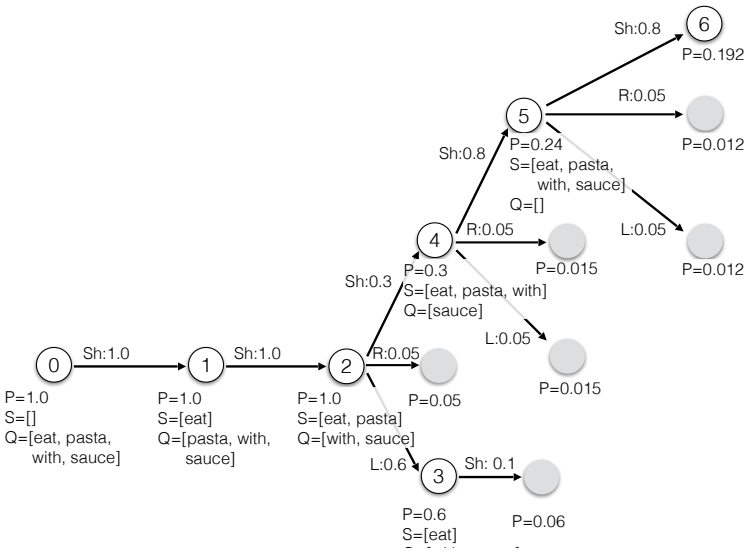

Figure 2: Exploration of parser state space using best-first search and error states. States are numbered according to the order in which they become the parser's current state. The local action classifier is trained with four classes: the three valid actions (represented as Sh for *shift*, L for *reduce-left*, and R for *reduce-right*) and an *error* class. The error class is not used by the parser and not shown in the diagram, but serves to reduce the total probability of valid parser actions by occupying some probability mass in each state, creating a way to reflect the overall quality of individual states.

the process is repeated (without clearing the priority queue, which already contains items corresponding to unexplored new states) until the current state is a final state (a state corresponding to a complete parse). This agenda-driven transition-based parsing approach, where the agenda is a priority queue, is optimal since all scores fall between 0 and 1, inclusive, but in practice a priority queue with limited capacity can be used to improve efficiency by preventing unbounded exploration of the exponential search space in cases where probabilities are nearly uniform. Figure 2 illustrates arc-standard dependency parsing with error states and best-first search. From states 0 and 1, the only possible action is shift. From state 2, the most probably action according to the model is reduce-left, which is not the correct action, but has probability 0.6. The correct action, shift, has probability 0.3. State 3 is then chosen as the current state, but when the classifier is applied to state 3, the only valid action, shift, is assigned probability 0.1. This is because the classifier assigns most of the probability mass to the error class, which the parser does not use. Because the state resulting from a shift from state 3 would have low probability, due to the low probability of shift, the search continues from state 4, and the parser has recovered from the classification error at state 2.

In the next section, we will present details of our neural network local classifiers.

4 Neural Models for Transition Based Parsing

We implement transition-based parsers with error states following two search strategies: the stepwise beam search normally used in transition-based parsers with global models (Zhang and Clark, 2008; Huang and Sagae, 2010) and best-first search (Sagae and Tsujii, 2007; Zhao et al., 2013), described in the previous section. The trainable components of our transition-based parsers are the local classifiers that predict the next action given features derived from the current state. Following Chen and Manning (2014), we train feed-forward neural networks (NNs) for local classification in our parsers. The NN is trained on pairs of features and actions, $\{\mathbf{f}^n, a^n\}_{n=1}^N$, where $\mathbf{f}^n$ is the feature vector extracted from the parser state and a^n is the corresponding correct action. For vanilla arc-standard parsing, a^n is one of $\{\textit{shift}, \textit{reduce-left}, \textit{reduce-right}\}$, and for parsing with error states, an additional *error* action.

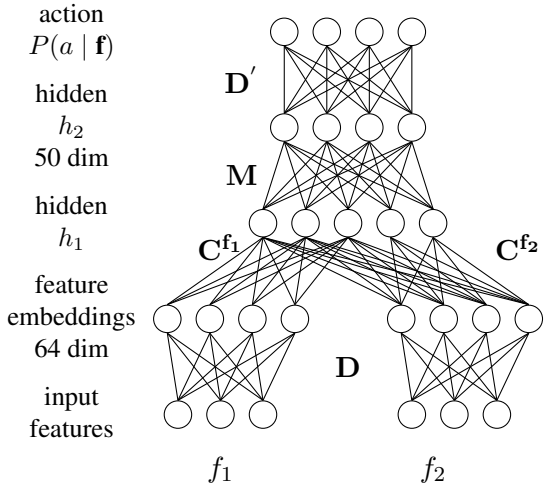

Figure 3: Basic architecture of our NN models

While parsing, we extract feature vector $\mathbf{f}$ from the current state and make a decision based on the output distribution, $P(a \mid \mathbf{f})$ computed by the NN.

We will now describe the basic architecture of our NN classifier and the features that we use. We will also describe how we pre-train embeddings from unannotated data for our word features.

4.1 Neural Network Model

Figure 3 shows the basic architecture of our neural network action prediction model for two input features $\mathbf{f} = f_1, f_2$, each of which is a *one-hot* vector with dimension equal to the number of possible feature types. The neural network has two hidden layers and the output softmax layer has the same number of units as the number of parser actions, that is, either 3 (without error states) or 4 (with error states). $\mathbf{D}$ is the input embedding matrix that is shared by all the input feature positions. Each feature position f_i has a corresponding position matrix $\mathbf{C^{f_i}}$. The two hidden layers h_1 and h_2 comprise rectified linear units (Nair and Hinton, 2010) having the activation $\max(0, x)$ (Figure 4).

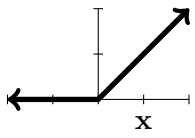

Figure 4: Activation function for a rectified linear unit.

The neural network computes the probability distribution over the parser actions, $P(a \mid \mathbf{f})$, as follows:

The first hidden layer computes,

$$h_1 = \phi\left(\sum_{f_i} \mathbf{C}^{f_i}\mathbf{D}f_i + \mathbf{b_1}\right),$$

where $\mathbf{b_1}$ is a vector of biases for h_1 and ϕ is applied elementwise.

The output of the second layer h_2 is

$$h_2 = \phi\left(\mathbf{M}h_1 + \mathbf{b_2}\right),$$

where $\mathbf{M}$ is a weight matrix between h_1 and h_2 and $\mathbf{b_2}$ is a vector of biases for h_2. The output softmax layer computes the probability of an action as:

$$P(a \mid \mathbf{f}) = \frac{\exp\left(\mathbf{v_a}\mathbf{D}'h_2 + \mathbf{b}^T\mathbf{v_a}\right)}{\sum_{a'}\exp\left(\mathbf{v_{a'}}\mathbf{D}'h_2 + \mathbf{b}^T\mathbf{v_{a'}}\right)},$$

where $\mathbf{D}'$ is the matrix of output action embeddings, $\mathbf{b}$ is a vector of action biases, and $\mathbf{v_a}$ is the one hot representation of the action a. We learn models that predict over two types of output distributions conditioned on $\mathbf{f}$: vanilla arc-standard models that predict over *shift*, *reduce-left* and *reduce-right*, and arc-standard models with error states (Section 3) that predict over *shift*, *reduce-left*, *reduce-right* and *error*.

4.2 Semi-supervised Learning: Pre-training Word Embeddings

It is often the case that large amounts of domain-specific unannotated data, i.e. raw text, is available in addition to annotated data. For both graph-based and transition-based parsing, many feature templates are defined on words from the input sentence. Previous work has shown benefits of using word representations learned from unannotated data. Koo et al. (2008) achieve significant improvement in dependency parsing on the Penn Treebank (PTB) (From 92.02% to 93.16%) by using Brown clusters (Brown et al., 1992) learned from the BLLIP corpus (Charniak et al., 2000). Chen and Manning (2014) also show 0.7% improvement on English dependency parsing on PTB using pre-trained English word embeddings from Collobert et al. (2011). We also seek to benefit from pre-trained embeddings

to initialize the input feature embeddings, $\mathbf{D}$ (Figure 3), in our neural network classifiers.

Following both Koo et al. (2008) and Chen and Manning (2014), we learn word embeddings by training a feed-forward neural network language model on a concatenation of the BLLIP corpus and sections 02–21 of the PTB corpus. We use the NPLM toolkit (http://nlg.isi.edu/software/nplm/), which implements noise contrastive estimation training of a two-hidden layer feed-forward neural network language model with rectifier linear units (Vaswani et al., 2013). We train a 7-gram neural language model with input word embedding dimension 64, 512 units in the first hidden layer, 512 units in the second hidden layer and output word-embedding dimension of 512. The neural language model is trained for 30 epochs using stochastic gradient descent and an initial learning rate of 1.0. We restrict the vocabulary to about $100k$ most-frequent words, replacing all the other words with $<unk>$. We use a validation set of about $250k$ n-grams and extract the input word embeddings from the epoch that achieves the lowest perplexity on the validation set. To avoid over-fitting, in our dependency parsing experiments, we only use pre-trained embeddings for the words that occurred at least twice in sections 02–21 of the PTB corpus. Pre-trained embeddings give us significant improvements over randomly initialized embeddings, as our results will show (section 5).

4.3 Training

We train six different types of NN classifiers for transition-based dependency parsing: one for each search algorithm with error states, with and without pre-trained word embeddings, in addition to two models with no error states, as described below.

4.3.1 Vanilla Arc-standard Parsers

We train NN classifiers for arc-standard transition-based parsers that compute probability distributions over *shift*, *reduce-left*, and *reduce-right*, using the 14 kernel features described by Huang and Sagae (2010), shown in Table 1. We trained models using both pre-trained (Section 4.2) and randomly initialized word embeddings. We denote these parsers as **Local–14–pre** and **Local–14–rand**. These models allow us to compare the use of NN classification without error states directly with the structured perceptron, and examine the impact of pre-trained word embeddings.

Word features	$s_0.w\ s_1.w\ s_2.w\ q_0.w\ q_1.w$
POS tag features	$s_0.t\ s_1.t\ s_2.t\ q_0.t\ q_1.t$
Child features	$s_0.lc.t\ s_0.rc.t\ s_1.lc.t\ s_1.rc.t$

Table 1: The 14 feature templates used in some of our models. s and q indicate the stack and the input buffer respectively, subscripts start at zero on the top of the stack or in the front of the input buffer. Finally, lc and rc indicate the leftmost left child and the rightmost right child, respectively.

4.3.2 Error State Parsers

We also train NN classifiers that differ from the ones above only in the use of error states: **ErrSt–14–pre** (error states, pre-trained embeddings), and **ErrSt–14–rand** (error states, randomly initialized embeddings). These allow us to examine the impact of using error states, and give us a way to compare our approach with NN and error states directly with an existing structured perceptron arc-standard parser (Huang and Sagae, 2010) using the same 14 kernel features and the same search approach. The differences in the two approaches are that Huang and Sagae (2010) use the structured perceptron and an extended set of features based on the kernel features, and we use NN with error states and only the 14 kernel features.

Additionally, we train two classifiers that use a more expressive expanded set of 25 features, which we use with best-first search (Sagae and Tsujii, 2007; Zhao et al., 2013) with error states: **ErrSt–25–pre** (expanded feature set, error states, pre-trained embeddings), and **ErrSt–25–rand** (expanded feature set, error states, randomly initialized embeddings). The 25 feature templates used by these classifiers are shown in Table 2. In contrast, Chen and Manning (2014) use 48 feature templates, including higher-order dependency information than has been shown to improve parsing accuracy significantly (Zhang and Nivre, 2011). It is likely that our approach would benefit similarly from the use of these features, but we leave the addition of features as future work.

Finally, for each of six types of classifiers above,

Word	$s_0.w\ s_1.w\ s_2.w\ q_0.w\ q_1.w\ q_2.w\ q_3.w$
POS tag features	$s_0.t\ s_1.t\ s_2.t\ q_0.t\ q_1.t\ q_2.t\ q_3.t$
Word child features	$s_0.lc.w\ s_0.rc.w\ s_1.lc.w\ s_1.rc.w$
POS child features	$s_0.lc.t\ s_0.rc.t\ s_1.lc.t\ s_1.rc.t$
Previous action	$previous\_action$
Distance	$dist(s_0, s_1)\ dist(q_0, s_0)$

Table 2: The expanded set of 25 feature templates used in some of our models. s and q indicate the stack and the input buffer respectively, subscripts start at zero on the top of the stack or in the front of the input buffer. lc and rc indicate the leftmost left child and the rightmost right child, respectively. $dist(a, b)$ is the signed distance between the root of a and the root of b in the input sentence, and $previous\_action$ is the action that was applied to generate the current state.

we train models using both Stanford dependencies (de Marneffe and Manning, 2008) and Yamada and Matsumoto (YM) dependencies (Yamada and Matsumoto, 2003) extracted from the Penn Treebank. We create $\{\mathbf{f}^n_{train}, a^n_{train}\}_{n=1}^{N_{train}}$ pairs on Wall Street Journal sections 02–21, and use $\{\mathbf{f}^n_{dev}, a^n_{dev}\}_{n=1}^{N_{dev}}$ pairs from section 22 as a development set, where N_{train} and N_{dev} are the number of training and dev instances. We obtain part-of-speech tags by training a CRF tagger on sections 02–21 with 4-way jackknifing, which achieves a tagging accuracy of 97.2% on section 23. We train our NN classifiers to maximize the log-likelihood of the correct actions given features,

$$\frac{1}{n}\sum_{n=1}^{N_{train}} \log P(a^n_{train} \mid \mathbf{f}^n_{train}).$$

We use mini-batch dropout training, computing gradients using the back-propagation algorithm (Hinton et al., 2012). We use the development set to tune the learning rate, halving it if the perplexity on the development set increases.

4.4 Model and Parser Selection

For each of our NN classifiers, there are a few tunable hyper-parameters: hidden layer size (h_1), mini-batch size, initial learning rate lr, dropout probabilities for h_1 and h_2 (d_{h_1}, and d_{h_2}) , and random initialization of parameters. We tuned each of these to maximize *classification accuracy* of the most likely action predicted by the classifier given the feature vector. We calculated classification accuracy as

$$\frac{\sum_{n=1}^{N_{dev}} \delta(\arg\max_a P(a \mid \mathbf{f}^n_{dev}), a^n_{dev})}{N_{dev}},$$

where $\delta(x, y)$ returns 1 if x equals y and 0 otherwise.

For each of the classifiers, we first tuned lr, mini-batch size, h_1 size, and d_{h_1} and d_{h_2} for accuracy. We tried $lr = \{1.0, 0.5, 0.25\}$, $h_1 = \{512, 1024, 1536, 2048\}$, mini-batch size $= \{128, 256, 512\}$, $d_{h_1} = \{0.0, 0.1, 0.2\}$, and $d_{h_2} = \{0.0, 0.1, 0.2\}$. For all our randomly initialized classifiers (**Local–14–rand**, **ErrSt–14–rand**, and **ErrSt–25–rand**), we chose the model that achieved the best classification accuracy on the development set for parsing the test set. We also used the same random seed to initialize our parameters. For **Local–14–pre**, **ErrSt–14–pre**, and **ErrSt–25–pre**, we trained models with different random initializations of the input embeddings ($\mathbf{D}$ in figure 3) that were not pre-trained. For each random initialization, we chose the model with the best classification accuracy on the development set. To pick the final model for parsing on test, we selected the model to maximize *parsing* accuracy on the development set. We computed our parsing accuracies using the $eval07.pl$ script from the CoNLL 2007 shared task on dependency parsing (Nivre et al., 2007), ignoring punctuation as is standard in English dependency parsing evaluation.

For both YM and Stanford dependencies, the optimal values of h_1 were 1536 for **ErrSt–25–pre** and **ErrSt–25–rand**, and either 1536, or 2048 for **Local–14–pre**, **Local–14–rand**, **ErrSt–14–pre**, and **ErrSt–14–rand**. For the best parser on Stanford and YM dependencies, (**ErrSt–25–pre**), we used a minibatch size of 256 and a initial learning rate of 0.25. For future work, we will explore a larger grid of learning rate, minibatch sizes, and dropout values.

At parsing time, we pre-multiply the input embeddings, $\mathbf{D}$ and the position matrices, $\mathbf{C^{f_i}}$, which speeds up computation significantly.

5 Results

In all experiments we use dependencies extracted from the Penn Treebank (Marcus et al., 1993) following the standard splits (WSJ sections 02 to 21

System	UAS
Local–14–pre (beam 1)	91.73
Local–14–pre (beam 4)	92.00 (+0.27)
ErrSt–14–pre (beam 1)	91.77
ErrSt–14–pre (beam 4)	92.61 (+0.84)
ErrSt–25–pre	**93.22**

Table 3: Unlabeled accuracy scores (UAS) on the development set with Stanford dependencies using transition-based models trained *with* pre-trained embeddings. **ErrSt–25–pre** uses best-first search with a priority queue of size limited to 100.

System	UAS
Local–14–rand (beam 1)	90.96
Local–14–rand (beam 4)	91.21 (+0.25)
ErrSt–14–rand (beam 1)	90.83
ErrSt–14–rand (beam 4)	91.98 (+1.15)
ErrSt–25–rand	**92.29**

Table 4: Unlabeled accuracy scores (UAS) on the development set with Stanford dependencies using models trained *without* pre-trained embeddings. **ErrSt–25–pre** uses best-first search with a priority queue of size limited to 100.

for training, 22 for development and 23 for testing), and part-of-speech tags assigned automatically using four-way jackknifing. Tables 3 and 4 present results obtained on the development set with our models trained with and without pre-trained word embeddings. Our baseline arc-standard parser using greedy search (**Local–14–pre** beam 1) is as accurate as the best NN dependency parser of Chen and Manning (2014), where both use pre-trained embeddings. In both tables, we can see that increasing the beam size from 1 (greedy parsing) to 4 gives only very modest improvements in accuracy when trained *without* error states (**Local–14–pre** and **Local–14–rand**). As mentioned in section 2.2, using beam search with vanilla arc-standard parsing with locally normalized models does not produce large improvements over greedy search due to the label bias problem. For both pre-trained and randomly initialized word embeddings, beam search with models trained *with* error states improves accuracy substantially (**ErrSt–14–pre** and **ErrSt–14–rand**). Our best parsers use pre-trained embeddings, best-first search, and a larger feature set (**ErrSt–25–pre**). In Table 4, we isolate the efficacy of training and search with error states. Even with randomly initialized embeddings, we are able to outperform Chen and Manning's NN dependency parser initialized with embeddings from external sources. Our results show that using error states in parsing can improve parsing accuracy independently of whether beam search or best-first search is used. Figures 5 and 6 show comparisons of the effects of increasing beam sizes in models trained with and without error states.

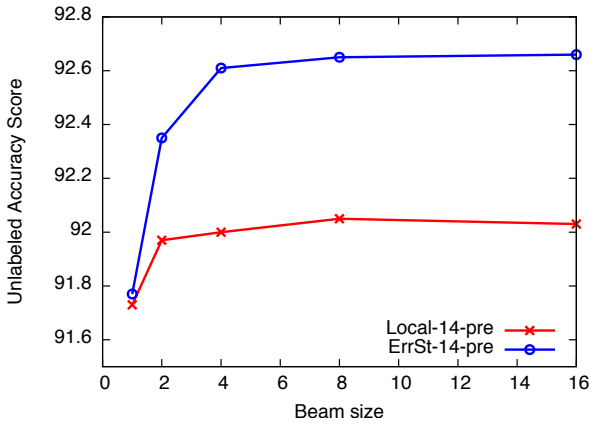

Figure 5: Effect of beam size on accuracy, using Stanford dependencies and models trained *with* pre-trained embeddings.

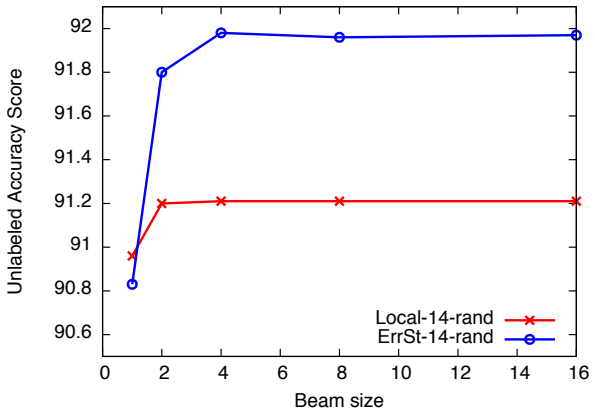

Figure 6: Effect of beam size on accuracy, using Stanford dependencies and models trained *without* pre-trained embeddings.

We additionally show that the benefits of using

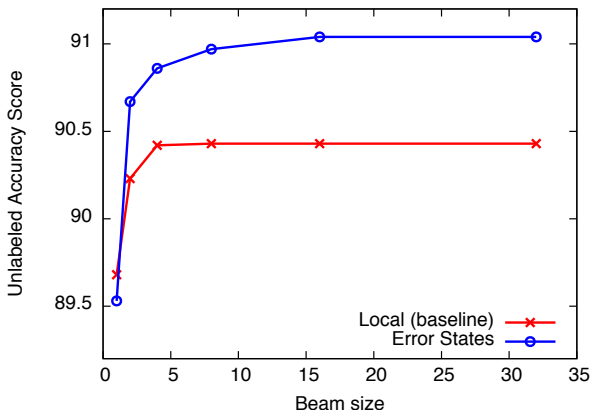

Figure 7: Unlabeled accuracy scores (UAS) obtained using maximum entropy models for local classification of parser actions with and without error states using various beam sizes.

error states are not limited to classification with neural networks. Figure 7 shows the results obtained on the development set with an arc-standard beam-search parser using maximum entropy classification[2] with L1 regularization and the full set of features used by Huang and Sagae (2010), and increasing beam sizes. The improvement obtained from beam-search with baseline local classification is limited as expected, while the improvement obtained from beam-search with error states is substantially more pronounced. Although the accuracy levels obtained with maximum entropy classification are clearly lower than those obtained with our neural network models, these results do confirm that error states are effective with linear classification.

In Table 5, we compare our best parsers with and without pre-trained word embeddings, **ErrSt–25–pre** and **ErrSt–25–rand**, against other published results. On Stanford dependencies, our parser with pre-trained embeddings performs comparably with the state-of-the-art. By using search with error states, we outperform a greedy NN parser (Chen and Manning, 2014) by a wide margin. On YM dependencies, our performance (**ErrSt–25–pre**) is comparable to that of a structured perceptron second-order graph-based parser using carefully selected features based on Brown clustering of the BLLIP

[2]We used Yoshimasa Tsuruoka's maximum entropy library, downloaded from `http://www.logos.ic.i.u-tokyo.ac.jp/~tsuruoka/maxent/`.

System	wsj23-S	wsj23-YM
ErrSt–25–rand	**92.17**	**92.16**
ErrSt–25–pre*	**93.61**	**93.21**
Chen & Manning*	91.8	–
Huang & Sagae	–	92.1
Zhang & Nivre	93.5	92.9
Weiss et al.*	93.99	–
Zhang & McDonald	93.71	93.57
Martins et al.	92.82	93.07
Koo et al. (dep2c)*	–	93.16

Table 5: Unlabeled accuracy scores (UAS) on Stanford dependencies (S) and Yamada & Matsumoto dependencies (YM) extracted from WSJ section 23 for our best transition-based parsers with error states, with and without pre-trained word embeddings. For comparison, we also include results from other transition-based approaches (Chen and Manning, 2014; Huang and Sagae, 2010; Zhang and Nivre, 2011; Weiss et al., 2015) and graph-based approaches (Zhang and McDonald, 2014; Martins et al., 2013; Koo et al., 2008). *These parsers use large sets of unlabeled data.

corpus (Koo et al., 2008). Our randomly initialized parser (no pre-trained embeddings), **ErrSt–25–rand** performs at a similar level to a structured perceptron transition-based parser (Huang and Sagae, 2010), but below that of parsers with finely tuned higher-order rich feature sets (Zhang and Nivre, 2011). While we leave the use of Zhang and Nivre's rich feature set as future work, for a direct comparison of NN models with error states and structured perceptron for transition-based dependency parsing, we additionally tested a parser (**ErrSt–14–rand**) that uses the exact same beam search and kernel features used by Huang and Sagae (2010). With NN and error states, we obtained 91.84% accuracy, compared to Huang and Sagae's 92.1%. An advantage of our approach is that we use the kernel features only, which come from the 14 templates shown in Table 1, while Huang and Sagae use additionally an extended set of features composed of carefully tuned concatenation templates involving the kernel features. The parsing speed of our parser implementation using our best model, **ErrSt–25–pre**, is

approximately 1,000 tokens (or 42 sentences) a second. Of course, such a measurement of speed is dependent on a variety of factors, such as hardware and programming language of the specific implementation, among others, so this figure observed in our experiments serves only as an illustrative sample.

6 Related Work

Our work builds on the transition-based parsing work described in Section 2, where local classifiers are trained to predict parser actions (Nivre, 2008), but provides a way to go beyond deterministic parsing. One way to create models capable of global scoring, and therefore effective search, is to parse with the structured perceptron (Zhang and Clark, 2008), which we also discuss in Section 2. Instead of performing global weight updates, our approach relies on local classifiers, but adds information about incorrect derivation paths to approximate a notion of global loss. This gives us a simple way to train neural network models for predicting parser actions locally but still perform effective search.

Our use of error states is conceptually related to the *correctness probability* estimate proposed by Yazdani and Henderson (2015), which is used only with each *shift* action of an arc-eager transition-based parsing model. This correctness probability creates a measure of quality of derivations at the point of each *shift*, which allows a combination of local action scores and the correctness probability to be used with beam search. The beam is then determined only at each *shift*, while search paths produced by other actions are extended exhaustively. Our error states, in contrast, adjust the scores of every action, making the use of best-first search natural.

Non-deterministic oracles for transition-based dependency parsing (Goldberg and Nivre, 2012; Goldberg and Nivre, 2013) are also designed to improve the performance of parsers that use local classification of actions by adding to the amount of information used to train the local classifiers. However, non-deterministic oracles aim to allow a deterministic parser to recover from incorrect actions by including information in the training of the local classifiers based on the notion that there may be several correct actions at a given point, as long as a desired tree remains reachable. In contrast, our local classifier, or oracle, is trained to encode a notion of state quality or approximate global loss that is specifically designed for search. In fact, when used with greedy search, our error states have no positive effect on parsing. This suggests that a combination of the benefits of non-deterministic oracles and error states may be possible.

Our training of local classifiers with error states shares with SEARN (Daumé III et al., 2009) and DAGGER (Ross et al., 2010) the idea of creating a notion of global loss in local scores, but SEARN and DAGGER learn to estimate the quality of search states by iteratively training policies using the entire training set, while we train only one policy, but using explicit information about states outside of the optimal path.

Choi and Palmer (2011) show that the idea of iteratively refining policies in a very similar way as proposed in SEARN and DAGGER can in fact be applied to transition-based dependency parsing to improve accuracy of deterministic parsing. By creating training examples for local classifiers based on parser states that are likely to occur at run time, but would not be generated with the gold-standard derivation, local classification models can be trained to be more robust in recovery from past errors. A key difference is that this provides a way for the parser to do better assuming that a mistake has already been made and is irrevocable, while our error states are designed to improve search, lowering the score of undesirable paths so a different path is chosen.

Our greedy neural network parser is similar to Chen and Manning (2014), who are the first to show the benefits of using feed-forward neural network classifiers in greedy transition-based dependency parsing. Unlike us, they use a single hidden layer of cube activation functions, and more features. We follow the neural network architecture of Vaswani et al. (2013), using two hidden layers of rectified linear units. Chen and Manning (2014) use Adagrad (Duchi et al., 2011) and dropout for optimization, while we use stochastic gradient descent with dropout. Recent work by Weiss et al. (2015) produces the highest published accuracy for English dependency parsing with very similar neural network architectures and similar pre-training of word embeddings. The accuracy of the greedy ver-

sion of their parser is substantially higher than that of our greedy parser, due at least in part to the use of more features. A more interesting difference between their approach and ours is in the way structured prediction is performed. While Weiss et al. add a structured perceptron layer to a network pretrained locally, we train *only locally*, but using error states. Both approaches are effective in producing improvements over the respective greedy baselines, and a direct comparison using the same greedy baseline is left for future work.

7 Conclusion

We presented a new approach for approximate structured inference for transition-based parsing that allows us to obtain high parsing accuracy using neural networks. Using error states, we improved search by producing scores suitable for global scoring using only local models, and showed that our approach is competitive with the structured perceptron in transition-based parsing. Additionally, our approach provides a straightforward way to take advantage of word embeddings in transition-based parsing, which produce high accuracy for transition-based dependency parsing in English, rivaling that of higher-order graph-based parsers. Source code, models and word embeddings for our transition-based dependency parser with error states are available at `http://github.com/sagae/nndep`.

Our approach for using error states to improve search is quite general, and could be applied to other structured problems that can be approximated using local models, such as sequence labeling and transition-based parsing with recurrent neural networks.

An area of future work is the application of error state training in problems where the local classifier has a high number of classes, as is often the case in labeled dependency parsing. A straightforward application of our approach roughly multiplies the number of training examples for the local classifier by the number of possible classes. For example, in labeled dependency parsing, where dependency labels are typically concatenated to actions, the number of classes is often greater than 30, which would increase the number of training examples more than 30-fold. Preventing such an increase in the number of training examples may be possible by factoring the problem in such a way that structure-building decisions are treated separately from labeling decisions (in labeled dependency parsing this would amount to training an arc labeling classifier separately), or perhaps more generally, by sampling from the possible error states.

Acknowledgments

We thank the anonymous reviewers for their numerous insightful suggestions. The work described here has been sponsored by the U.S. Army under contract number W911NF-14-D-0005 and ARO grant W911NF-10-1-0533. Any opinions, content or information presented does not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

References

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 2000. BLLIP 1987-89 WSJ corpus release 1. *Linguistic Data Consortium, Philadelphia*, 36.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of Empirical Methods in Natural Language Processing.*, pages 740–750.

Jinho D. Choi and Martha Palmer. 2011. Getting the most out of transition-based dependency parsing. In *Proceedings of the Association for Computational Linguistics.*, HLT '11, pages 687–692, Stroudsburg, PA, USA. Proceeding of the Association for Computational Linguistics.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the Association for Computational Linguistics.*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of the Empirical Methods in Natural Language Processing*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Journal of Machine Learning Research.*, 75(3):297–325, June.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, August. Coling 2008 Organizing Committee.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the International Conference on Computational Linguistics.*, pages 959–976. The COLING 2012 Organizing Committee.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association of Computational Linguistics*, pages 403–414.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental joint approach to word segmentation, POS tagging, and dependency parsing in Chinese. In *Proceedings of the Association for Computational Linguistics*, ACL '12, pages 1045–1053, Stroudsburg, PA, USA. Association for Computational Linguistics.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Matthew Honnibal, 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*, pages 163–172. Proceedings of the Association for Computational Linguistics.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the Association for Computational Linguistics.*, ACL '10, pages 1077–1086, Stroudsburg, PA, USA. Association for Computational Linguistics.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics.*, NAACL HLT '12, pages 142–151, Stroudsburg, PA, USA. Association for Computational Linguistics.

Richard Johansson and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 206–210, Stroudsburg, PA, USA. Association for Computational Linguistics.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the Association for Computational Linguistics*, pages 595–603. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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

André F. T. Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceeding of the Association for Computational Linguistics.*, pages 617–622.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. 2000. Maximum Entropy Markov Models for information extraction and segmentation. In *Proceedings of the International Conference in Machine Learning.*, volume 17, pages 591–598.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the International Conference on Machine Learning.*, pages 807–814.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of the International Conference on Computational Linguistics.*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic. Association for Computational Linguistics.

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

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2010. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research – Proceedings Track*, 15.

Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the CoNLL 2007 Shared Task in the Joint Conferences on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1044–1050.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proceedings of Empirical Methods in Natural Language Processing.*, pages 1387–1392. Citeseer.

David Weiss, 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, China, July. Association for Computational Linguistics.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with Support Vector Machines. In *The 8th International Workshop of Parsing Technologies (IWPT2003)*.

Majid Yazdani 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, China, July. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 562–571, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the Association for Computational Linguistics*, pages 656–661. Association for Computational Linguistics.

Yue Zhang 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, Oregon. Association for Computational Linguistics.

Kai Zhao, James Cross, and Liang Huang. 2013. Optimal incremental parsing via best-first dynamic programming. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 758–768.