Incremental Recurrent Neural Network Dependency Parser with Search-based Discriminative Training

Majid Yazdani
Computer Science Department
University of Geneva
majid.yazdani@unige.ch

James Henderson
Xerox Research Center Europe
james.henderson@xrce.xerox.com

Abstract

We propose a discriminatively trained recurrent neural network (RNN) that predicts the actions for a fast and accurate shift-reduce dependency parser. The RNN uses its output-dependent model structure to compute hidden vectors that encode the preceding partial parse, and uses them to estimate probabilities of parser actions. Unlike a similar previous generative model (Henderson and Titov, 2010), the RNN is trained discriminatively to optimize a fast beam search. This beam search prunes after each shift action, so we add a *correctness probability* to each shift action and train this score to discriminate between correct and incorrect sequences of parser actions. We also speed up parsing time by caching computations for frequent feature combinations, including during training, giving us both faster training and a form of backoff smoothing. The resulting parser is over 35 times faster than its generative counterpart with nearly the same accuracy, producing state-of-art dependency parsing results while requiring minimal feature engineering.

1 Introduction and Motivation

There has been significant interest recently in machine learning and natural language processing community in models that learn hidden multilayer representations to solve various tasks. Neural networks have been popular in this area as a powerful and yet efficient models. For example, feed forward neural networks were used in language modeling (Bengio et al., 2003; Collobert and Weston, 2008), and recurrent neural networks (RNNs) have yielded state-of-art results in language modeling (Mikolov et al., 2010), language generation (Sutskever et al., 2011) and language understanding (Yao et al., 2013).

1.1 Neural Network Parsing

Neural networks have also been popular in parsing. These models can be divided into those whose design are motivated mostly by inducing useful vector representations (e.g. (Socher et al., 2011; Socher et al., 2013; Collobert, 2011)), and those whose design are motivated mostly by efficient inference and decoding (e.g. (Henderson, 2003; Henderson and Titov, 2010; Henderson et al., 2013; Chen and Manning, 2014)).

The first group of neural network parsers are all deep models, such as RNNs, which gives them the power to induce vector representations for complex linguistic structures without extensive feature engineering. However, decoding in these models can only be done accurately if they are used to rerank the best parse trees of another parser (Socher et al., 2013).

The second group of parsers use a shift-reduce parsing architecture so that they can use search based decoding algorithms with effective pruning strategies. The more accurate parsers also use a RNN architecture (see Section 6), and use generative models to allow beam search. These models are accurate but are relatively slow, and accuracy degrades when you choose decoding settings to optimize speed. Because they are generative, they need to predict the words as the parse proceeds through the sentence, which requires normalization over all the vocabulary of words. Also, the beam search must maintain many candidates in the beam in order to check how well each one predicts future words. Recently (Chen and Manning, 2014) propose a discriminative neural network shift-reduce parser, which is very fast but less accurate (see Section 6). However, this parser uses a feed-forward neural network with a large set of hand-coded features, making it of limited inter-

Proceedings of the 19th Conference on Computational Language Learning, pages 142–152,
Beijing, China, July 30-31, 2015.

est for inducing vector representations of complex linguistic structures.

1.2 Incremental Recurrent Neural Network Architecture

In both approaches to neural network parsing, RNN models have the advantage that they need minimal feature engineering and therefore they can be used with little effort for a variety of languages and applications. As with other deep neural network architectures, RNNs induce complex features automatically by passing induced (hidden) features as input to other induced features in a recursive structure. This is a particular advantage for domain adaptation, multi-task learning, transfer learning, and semi-supervised learning, where hand-crafted feature engineering is often particularly difficult. The information that is transferred from one task to another is embedded in the induced feature vectors in a shared latent space, which is input to another hidden layer for the target model (Henderson et al., 2013; Raina et al., 2007; Collobert et al., 2011; Glorot et al., 2011). This transferred information has proven to be particularly useful when it comes from very large datasets, such as web-scale text corpora, but learning and inference on such datasets is only practical with efficient algorithms.

In this work, we propose a fast discriminative RNN model of shift-reduce dependency parsing. We choose a left-to-right shift-reduce dependency parsing architecture to benefit from efficient decoding. It also easily supports incremental interpretation in dialogue systems, or incremental language modeling for speech recognition. We choose a RNN architecture to benefit from the automatic induction of informative vector representations of complex linguistic structures and the resulting reduction in the required feature engineering. This hidden vector representation is trained to encode the partial parse tree that has been built by the preceding parse, and is used to predict the next parser action conditioned on this history.

As our RNN architecture, we use the neural network approximation of ISBNs (Henderson and Titov, 2010), which we refer to as an Incremental Neural Network (INN). INNs are a kind of RNN where the model structure is built incrementally as a function of the values of previous output variables. In our case, the hidden vector used to make the current parser decision is connected to the hidden vector from previous decisions based on the partial parse structure that has been built by the previous decisions. So any information about the unbounded parse history can potentially be passed to the current decision through a chain of hidden vectors that reflects locality in the parse tree, and not just locality in the derivation sequence (Henderson, 2003; Titov and Henderson, 2007b). As in all deep neural network architectures, this chaining of nonlinear vector computations gives the model a very powerful mechanism to induce complex features from combinations of features in the history, which is difficult to replicate with hand-coded features.

1.3 Search-based Discriminative Training

We propose a discriminative model because it allows us to use lookahead instead of word prediction. As mentioned above, generative word prediction is costly, both to compute and because it requires larger beams to be effective. With lookahead, it is possible to condition on words that are farther ahead in the string, and thereby avoid hypothesizing parses that are incompatible with those future words. This allows the parser to prune much more aggressively without losing accuracy. Discriminative learning further improves this aggressive pruning, because it can optimize for the discrete choice of whether to prune or not (Huang et al., 2012; Zhang and Clark, 2011).

Our proposed model primarily differs from previous discriminative models of shift-reduce dependency parsing in the nature of the discriminative choices that are made and the way these decisions are modeled and learned. Rather than learning to make pruning decisions at each parse action, we learn to choose between sequences of actions that occur in between two shift actions. This way of grouping action sequences into chunks associated with each word has been used previously for efficient pruning strategies in generative parsing (Henderson, 2003), and for synchronizing syntactic parsing and semantic role labeling in a joint model (Henderson et al., 2013). We show empirically that making discriminative parsing decisions at the scale of these chunks also provides a good balance between grouping decisions so that more context can be used to make accurate parsing decisions and dividing decisions so that the space of alternatives for each decision can be considered quickly (see Figure 4 below).

In line with this pruning strategy, we define a score called the *correctness probability* for every shift action. This score is trained discriminatively to indicate whether the entire parse prefix is correct or not. This gives us a score function that is trained to optimize the pruning decisions during search (c.f. (Daumé III and Marcu, 2005)). By combining the scores for all the shift actions in each candidate parse, we can also discriminate between multiple parses in a beam of parses, thereby giving us the option of using beam search to improve accuracy in cases where bounded lookahead does not provide enough information to make a deterministic decision. The *correctness probability* is estimated by only looking at the hidden vector at its shift action, which encourages the hidden units to encode any information about the parse history that is relevant to deciding whether this is a good or bad parse, including long distance features.

1.4 Feature Decomposition and Caching

Another popular method of previous neural network models that we use and extend in this paper is the decomposition of input feature parameters using vector-matrix multiplication (Bengio et al., 2003; Collobert et al., 2011; Collobert and Weston, 2008). As the previous work shows, this decomposition overcomes the features sparsity common in NLP tasks and also enables us to use unlabeled data effectively. For example, the parameter vector for the feature *word-on-top-of-stack* is decomposed into the multiplication of a parameter vector representing the word and a parameter matrix representing *top-of-stack*. But sparsity is not always a problem, since the frequency of such features follows a power law distribution, so there are some very frequent feature combinations. Previous work has noticed that the vector-matrix multiplication of these frequent feature combinations takes most of the computation time during testing, so they cache these computations (Bengio et al., 2003; Devlin et al., 2014; Chen and Manning, 2014).

We note that these computations also take most of the computation time during training, and that the abundance of data for these feature combinations removes the statistical motivation for decomposing them. We propose to treat the cached vectors for high frequency feature combinations as parameters in their own right, using them both during training and during testing.

In summary, this paper makes several contributions to neural network parsing by considering different scales in the parse sequence and in the parametrization. We propose a discriminative recurrent neural network model of dependency parsing that is trained to optimize an efficient form of beam search that prunes based on the sub-sequences of parser actions between two shifts, rather than pruning after each parser action. We cache high frequency parameter computations during both testing and training, and train the cached vectors as separate parameters. As shown in section 6, these improvements significantly reduce both training and testing times while preserving accuracy.

2 History Based Neural Network Parsing

In this section we briefly specify the action sequences that we model and the neural network architecture that we use to model them.

2.1 The Parsing Model

In shift-reduce dependency parsing, at each step of the parse, the configuration of the parser consists of a stack S of words, the queue Q of words and the partial labeled dependency trees constructed by the previous history of parser actions. The parser starts with an empty stack S and all the input words in the queue Q, and terminates when it reaches a configuration with an empty queue Q. We use an arc-eager algorithm, which has 4 actions that all manipulate the word s on top of the stack S and the word q on the front of the queue Q: The decision *Left-Arc*$_r$ adds a dependency arc from q to s labeled r. Word s is then popped from the stack. The decision *Right-Arc*$_r$ adds an arc from s to q labeled r. The decision *Reduce* pops s from the stack. The decision *Shift* shifts q from the queue to the stack. For more details we refer the reader to (Nivre et al., 2004). In this paper we chose the exact definition of the parse actions that are used in (Titov and Henderson, 2007b).

At each step of the parse, the parser needs to choose between the set of possible next actions. To train a classifier to choose the best actions, previous work has proposed memory-based classifiers (Nivre et al., 2004), SVMs (Nivre et al., 2006), structured perceptron (Huang et al., 2012; Zhang and Clark, 2011), two-layer neural networks (Chen and Manning, 2014), and Incremental Sigmoid Belief Networks (ISBN) (Titov and

Henderson, 2007b), amongst other approaches.

We take a history based approach to model these sequences of parser actions, which decomposes the conditional probability of the parse using the chain rule:

$$\begin{aligned} P(T|S) &= P(D^1 \cdots D^m|S) \\ &= \prod_t P(D^t|D^1 \cdots D^{t-1}, S) \end{aligned}$$

where T is the parse tree, $D^1 \cdots D^m$ is its equivalent sequence of shift-reduce parser actions and S is the input sentence. The probability of *Left-Arc*$_r$ and *Right-Arc*$_r$ include both the probability of the attachment decision and the chosen label r. But unlike in (Titov and Henderson, 2007b), the probability of *Shift* does not include a probability predicting the next word, since all the words S are included in the conditioning.

2.2 Estimating Action Probabilities

To estimate each $P(D^t|D^1 \cdots D^{t-1}, S)$, we need to handle the unbounded nature of both $D^1 \cdots D^{t-1}$ and S. We can divide S into the words that have already been shifted, which are handled as part of our encoding of $D^1 \cdots D^{t-1}$, and the words on the queue. To condition on the words in the queue, we use a bounded lookahead:

$$P(T|S) \approx \prod_t P(D^t|D^1 \cdots D^{t-1}, w^t_{a_1} \cdots w^t_{a_k})$$

where $w^t_{a_1} \cdots w^t_{a_k}$ is the first k words on the front of the queue at time t. At every *Shift* action the lookahead changes, moving one word onto the stack and adding a new word from the input.

To estimate the probability of a decision at time t conditioned on the history of actions $D^1 \cdots D^{t-1}$, we overcome the problem of conditioning on an unbounded amount of information by using a neural network to induce hidden representations of the parse history sequence. The relevant information about the whole parse history at time t is encoded in its hidden representation, denoted by the vector h^t of size d.

$$\prod_t P(D^t|D^1 \cdots D^{t-1}, w^t_{a_1} \cdots w^t_{a_k}) = \prod_t P(D^t|h^t)$$

The hidden representation at time t is induced from hidden representations of the relevant previous states, plus pre-defined features $\mathcal{F}$ computed from the previous decision and the current queue and stack:

$$h^t = \sigma(\sum_{c \in \mathcal{C}} h^{t_c} W^c_{HH} + \sum_{f \in \mathcal{F}} W_{IH}(f,:))$$

In which $\mathcal{C}$ is the set of link types for the previous relevant hidden representations, h^{t_c} is the hidden representation of time $t_c < t$ that is relevant to h^t by the relation c, W^c_{HH} is the hidden to hidden transition weights for the link type c, and W_{IH} is the weights from features $\mathcal{F}$ to hidden representations. σ is the sigmoid function and $W(i,:)$ shows row i of matrix W. $\mathcal{F}$ and $\mathcal{C}$ are the only hand-coded parts of the model.

The decomposition of features has attracted a lot of attention in NLP tasks, because it overcomes feature sparsity. There is transfer learning from the same word (or POS tag, Dependency label, etc.) in different positions or to similar words. Also unsupervised training of word embeddings can be used effectively within decomposed features. The use of unsupervised word embeddings in various natural language processing tasks has received much attention (Bengio et al., 2003; Collobert and Weston, 2008; Collobert, 2011). Word embeddings are real-valued feature vectors that are induced from large corpora of unlabeled text data. Using word embeddings with a large dictionary improves domain adaptation, and in the case of a small training set can improve the performance of the model.

Given these advantages, we use feature decompositions to define the input-to-hidden weights W_{IH}.

$$W_{IH}(f,:) = W_{emb.}(val(f),:)W^f_{HH}$$

Every row in $W_{emb.}$ is an embedding for a feature value, which may be a word, lemma, pos tag, or dependency relation. $val(f)$ is the index of the value for feature type f, for example the particular word that is at the front of the queue. Matrix W^f_{HH} is the transition matrix from the feature value embeddings to the hidden vector, for the given feature type f. For simplicity, we assume here that the size of the embeddings and the size of the hidden representations of the INN are the same.

In this way, the parameters of the embedding matrix $W_{emb.}$ is shared among various feature input link types f, which can improve the model in the case of sparse features. It also allows the use of word embeddings that are available on the web to improve coverage of sparse features, but we leave

this investigation to future work since it is orthogonal to the contributions of this paper.

Finally, the probability of each decision is normalized across other alternative decisions, and only conditioned on the hidden representation (softmax layer):

$$P(D^t{=}d|h^t) = \frac{e^{h^t W_{HO}(:,d)}}{\sum_{d'} e^{h^t W_{HO}(:,d')}}$$

where W_{HO} is the weight matrix from hidden representations to the outputs.

3 Discrimination of Partial Parses

Unlike in a generative model, the above formulas for computing the probability of a tree make independence assumptions in that words to the right of $w^t_{a_k}$ are assumed to be independent of D^t. And even for words in the lookahead it can be difficult to learn dependencies with the unstructured lookahead string. The generative model first constructs a structure and then uses word prediction to test how well that matches the next word. If a discriminative model uses normalized estimates for decisions, then once a wrong decision is made there is no way for the estimates to express that this decision has lead to a structure that is incompatible with the current or future lookahead string (see (Lafferty et al., 2001) for more discussion). More generally, any discriminative model that is trained to predict individual actions has this problem. In this section we discuss how to overcome this issue.

3.1 Discriminating Correct Parse Chunks

Due to this problem, discriminative parsers typically make irrevocable choices for each individual action in the parse. We propose a method for training a discriminative parser which addresses this problem in two ways. First, we train the model to discriminate between larger sub-sequences of actions, namely the actions between two *Shift* actions, which we call *chunks*. This allows the parser to delay choosing between actions that occur early in a chunk until all the structure associated with that chunk has been built. Second, the model's score can be used to discriminate between two parses long after they have diverged, making it appropriate for a beam search.

We employ a search strategy where we prune at each shift action, but in between shift actions we consider all possible sequences of actions, similarly to the generative parser in (Henderson, 2003). The INN model is discriminatively trained to choose between these chunks of sequences of actions.

The most straightforward way to model these chunk decisions would be to use unnormalized scores for the decisions in a chunk and sum these scores to make the decision, as would be done for a structured perceptron or conditional random field. Preliminary experiments applying this approach to our INN parser did not work as well as having local normalization of action decisions. We hypothesize that the main reason for this result is that updating on entire chunks does not provide a sufficiently focused training signal. With a locally normalized action score, such as softmax, increasing the score of the correct chunk has the effect of decreasing the score of all the incorrect actions at each individual action decision. This update is an approximation to a discriminative update on all incorrect parses that continue from an incorrect decision (Henderson, 2004). Another possible reason is that local normalization prevents one action's score from dominating the score of the whole parse, as can happen with high frequency decisions. In general, this problem can not be solved just by using norm regularization on weights.

The places in a parse where the generative update and the discriminative update differ substantially are at word predictions, where the generative model considers that all words are possible but a discriminative model already knows what word comes next and so does not need to predict anything. We discriminatively train the INN to choose between chunks of actions by adding a new score at these places in the parse. After each shift action, we introduce a *correctness probability* that is trained to discriminate between cases where the chunk of actions since the previous shift is correct and those where this chunk is incorrect. Thus, the search strategy chooses between all possible sequences of actions between two shifts using a combination of the normalized scores for each action and the *correctness probability*.

In addition to discriminative training at the chunk level, the *correctness probability* allows us to search using a beam of parses. If a correct decision can not be disambiguated, because of the independence assumptions with words beyond the lookahead or because of the difficulty of inferring from an unstructured lookahead, the *correctness*

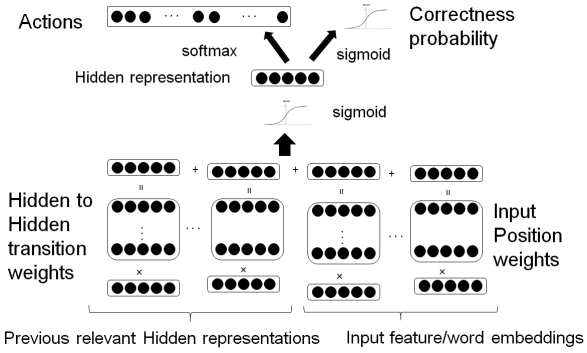


Figure 1: INN computations for one decision

probability score will drop whenever the mistake becomes evident. This means that we can not only compare two partial parses that differ in the most recent chunk, but we can also compare two partial parses that differ in earlier chunks. This allows us to use beam search decoding. Instead of deterministically choosing a single partial parse at each shift action, we can maintain a small beam of alternatives and choose between them based on how compatible they are with future lookahead strings by comparing their respective correctness probabilities for those future shifts.

We combine this *correctness probability* with the action probabilities by simply multiplying:

$$P(T|S) \approx \prod_t P(D^t|h^t)P(\textit{Correct}|h^t)$$

where we train $P(\textit{Correct}|h^t)$ to be the *correctness probability* for the cases where D^t is a shift action, and define $P(\textit{Correct}|h^t) = 1$ otherwise. For the shift actions, $P(\textit{Correct}|h^t)$ is defined using the sigmoid function:

$$P(\textit{Correct}|h^t) = \sigma(h^t W_{Cor})$$

In this way, the hidden representations are trained not only to choose the right action, but also to encode correctness of the partial parses. Figure 1 shows the model schematically.

3.2 Training the Parameters

We want to train the *correctness probabilities* to discriminate correct parses from incorrect parses. The correct parses can be extracted from the training treebank by converting each dependency tree into its equivalent sequence of arc-eager shift-reduce parser actions (Nivre et al., 2004; Titov and Henderson, 2007b). These sequences of actions provide us with positive examples. For discriminative training, we also need incorrect parses to act as the negative examples. In particular, we want negative examples that will allow us to optimize the pruning decisions made by the parser.

To optimize the pruning decisions made by the parsing model, we use the parsing model itself to generate the negative examples (Collins and Roark, 2004). Using the current parameters of the model, we apply our search-based decoding strategy to find our current approximation to the highest scoring complete parse, which is the output of our current parsing model. If this parse differs from the correct one, then we train the parameters of all the *correctness probabilities* in each parse so as to increase the score of the correct parse and decrease the score of the incorrect output parse. By repeatedly decreasing the score of the incorrect best parse as the model parameters are learned, training will efficiently decreases the score of all incorrect parses.

As discussed above, we train the scores of individual parser actions to optimize the locally-normalized conditional probability of the correct action. Putting this together with the above training of the *correctness probabilities* $P(\textit{Correct}|h^t)$, we get the following objective function:

$$\begin{aligned}&\textit{argmax}_\phi \\ &\sum_{T \in T_{pos}} \sum_{t \in T} logP(d^t|h^t) + logP(\textit{Correct}|h^t) \\ &- \sum_{T \in T^\phi_{neg}} \sum_{t \in T} logP(\textit{Correct}|h^t)\end{aligned}$$

where ϕ is the set of all parameters of the model (namely W_{HH}, $W_{emb.}$, W_{HO}, and W_{Cor}), T_{pos} is the set of correct parses, and T^ϕ_{neg} is the set of incorrect parses which the model ϕ scores higher than their corresponding correct parses. The derivative of this objective function is the error signal that the neural network learns to minimize. This error signal is illustrated schematically in Figure 2.

We optimize the above objective using stochastic gradient descent. For each parameter update, a positive tree is chosen randomly and a negative tree is built using the above strategy. The resulting error signals are backpropagated through the INN to compute the derivatives for gradient descent.

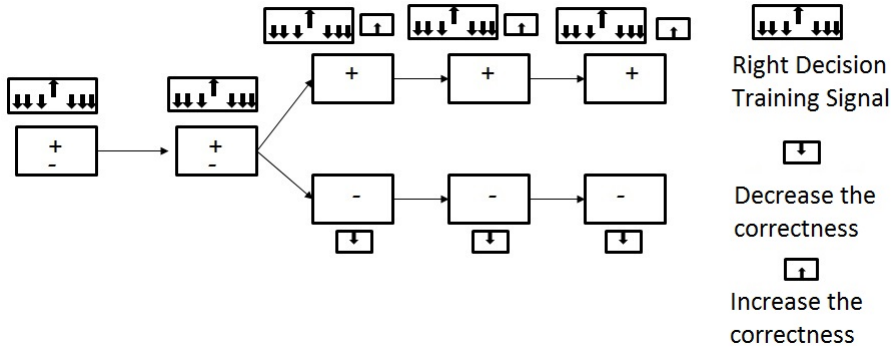
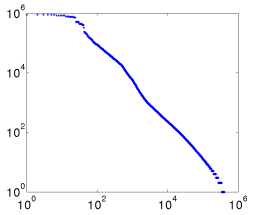
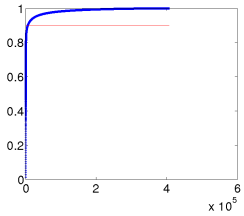


Figure 2: Positive and negative derivation branches and their training signals



(a) Log-Log frequency by rank of features



(b) Cumulative fraction of features ranked by frequency

Figure 3: Feature frequency distributions

4 Caching Frequent Features

Decomposing the parametrization of input features using vector-matrix multiplication, as was described in section 2, overcomes the features sparsity common in NLP tasks and also makes it possible to use unlabeled data effectively. But it adds a huge amount of computation to both the training and decoding, since for every input feature a vector-matrix multiplication is needed. This problem is even more severe in our algorithm because at every training iteration we search for the best incorrect parse.

The frequency of features follows a power law distribution; there are a few very frequent features, and a long tail of infrequent features. Previous work has noticed that the vector-matrix multiplication of the frequent features takes most of the computation time during testing, so they cache these computations (Bengio et al., 2003; Devlin et al., 2014; Chen and Manning, 2014). For example in the Figure 3(b), only 2100 features are responsible of 90% of the computations among ∼400k features, so cashing these computations can have a huge impact on speed. We note that these computations also take most of the computation time during training. First, this computation is the dominant part of the forward and backward error computations. Second, at every iteration we need to decode to find the highest scoring incorrect parse.

We propose to treat the cached vectors for high frequency features as parameters in their own right, using them both during training and during testing. This speeds up training because it is no longer necessary to do the high-frequency vector-matrix multiplications, neither to do the forward error computations nor to do the backpropagation through the vector-matrix multiplications. Also, the cached vectors used in decoding do not need to be recomputed every time the parameters change, since the vectors are updated directly by the parameter updates.

Another possible motivation for treating the cached vectors as parameters is that it results in a kind of backoff smoothing; high frequency features are given specific parameters, and for low frequency features we back off to the decomposed model. Results from smoothing methods for symbolic statistical models indicate that it is better to smooth low frequency features with other low frequency features, and treat high frequency features individually. In this paper we do not systematically investigate this potential advantage, leaving this for future work.

In our experiments we cache features that make up to 90% of the feature frequencies. This gives us about a 20 times speed up during training and about a 7 times speed up during testing, while performance is preserved.

5 Parsing Complexity

If the length of the latent vectors is d, for a sentence of length L, and beam B, the decoding com-

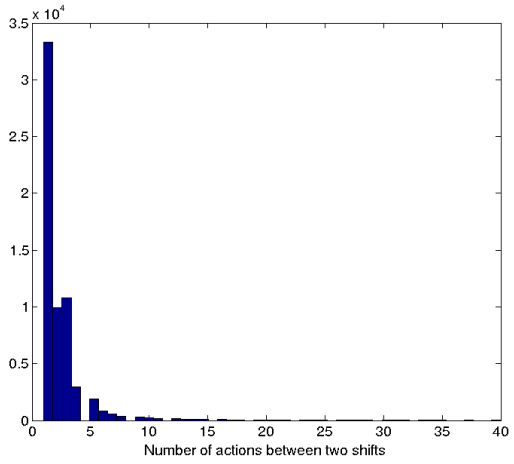


Figure 4: Histogram of number of candidate actions between shifts

plexity is $O(L \times M \times B \times (|\mathcal{F}| + |\mathcal{C}|) \times d^2)$.[1] If we choose the best partial parse tree at every shift, then $B = 1$. M is the total number of actions in the candidate chunks that are generated between two shifts, so $L \times M$ is the total number of candidate actions in a parse. For shift-reduce dependency parsing, the total number of *chosen* actions is necessarily linear in L, but because we are applying best-first search in between shifts, M is not necessarily independent of L. To investigate the impact of M on the speed of the parser, we empirically measure the number of candidate actions generated by the parser between 33368 different shifts. The resulting distribution is plotted in Figure 4. We observe that it forms a power law distribution. Most of the time the number of actions is very small (2 or 3), with the maximum number being 40, and the average being 2.25. We conclude from this that the value of M is not a major factor in the parser's speed.

Remember that $|\mathcal{F}|$ is the number of input feature types. Caching 90% of the input feature computations allows us to reasonably neglect this term. Because $|\mathcal{C}|$ is the number of hidden-to-hidden connection types, we cannot apply caching to reduce this term. However, $|\mathcal{C}|$ is much smaller than $|\mathcal{F}|$ (here $|\mathcal{C}|$=3). This is why caching input feature computations has such a large impact on parser speed.

[1]For this analysis, we assume that the output computation is negligable compared to the hidden representation computation, because the output computation grows with d while the hidden computation grows with d^2.

6 Experimental Results

We used syntactic dependencies from the English section of the CoNLL 2009 shared task dataset (Hajič et al., 2009). Standard splits of training, development and test sets were used. We compare our model to the generative INN model (Titov and Henderson, 2007b), MALT parser, MST parser, and the feed-forward neural network parser of (Chen and Manning, 2014) ("C&M"). All these models and our own models are trained only on the CoNLL 2009 syntactic data; they use no external word embeddings or other unsupervised training on additional data. This is one reason for choosing these models for comparison. In addition, the generative model ("Generative INN, large beam" in Table 1) was compared extensively to state-of-art parsers on various languages and tasks in previous work (Titov and Henderson, 2007b; Titov and Henderson, 2007a; Henderson et al., 2008). Therefore, here our objective is not repeating an extensive comparison to the available parsers.

Table 1 shows the labeled and unlabeled accuracy of attachments for these models. The MALT and MST parser scores come from (Surdeanu and Manning, 2010), which compared different parsing models using CoNLL 2008 shared task dataset, which is the same as CoNLL 2009 for English syntactic parsing. The results for the generative INN with a large beam were taken from (Henderson and Titov, 2010), which uses an architecture with 80 hidden units. We replicate this setting for the other generative INN results and our discriminative INN results. The parser of (Chen and Manning, 2014) was run with their architecture of 200 hidden units with dropout training ("C&M"). All parsing speeds were computed using the latest downloadable versions of the parsers, on a single 3.4GHz CPU.

Our model with beam 1 (i.e. deterministic choices of chunks) ("DINN, beam 1") produces state-of-the-art results while it is over 35 times faster than the generative model with beam size 10. Moreover, we are able to achieve higher accuracies using larger beams ("DINN, beam 10"). The discriminative training of the *correctness probabilities* to optimize search is crucial to these levels of accuracy, as indicated by the relatively poor performance of our model when this training is removed ("Discriminative INN, no search training"). Previous deterministic shift-reduce parsers ("$MALT_{AE}$" and "C&M") are around twice as fast

Model	LAA	UAA	wrd/sec
$MALT_{AE}$	85.96	88.64	7549
C&M	86.49	89.17	9589
MST	87.07	89.95	290
MALT-MST	87.45	90.22	NA
Generative INN,			
beam 1	77.83	81.49	1122
beam 10	87.67	90.61	107
large beam	88.65	91.44	NA
Discriminative INN,			
no search training	85.28	88.98	4012
DINN, beam 1	87.26	90.13	4035
DINN, beam 10	88.14	90.75	433

Table 1: Labelled and unlabelled attachment accuracies and speeds on the test set.

Model	German		Spanish		Czech	
	LAA	UAA	LAA	UAA	LAA	UAA
C&M	82.5	86.1	81.5	85.4	58.6	70.6
MALT	80.7	83.1	82.4	86.6	67.3	77.4
MST	84.1	87.6	82.7	87.3	73.4	81.7
DINN	86.0	89.6	85.4	88.3	77.5	85.2

Table 2: Labelled and unlabelled attachment accuracies on the test set of CoNLL 2009.

as our beam 1 model, but at the cost of significant reductions in accuracy.

To evaluate our RNN model's ability to induce informative features automatically, we trained our deterministic model, MALT, MST and C&M on three diverse languages from CoNLL 2009, using the same features as used in the above experiments on English (model "DINN, beam 1"). We did no language-specific feature engineering for any of these parsers. Table 2 shows that our RNN model generalizes substantially better than all these models to new languages, demonstrating the power of this model's feature induction.

7 Conclusion

We propose an efficient and accurate recurrent neural network dependency parser that uses neural network hidden representations to encode arbitrarily large partial parses for predicting the next parser action. This parser uses a search strategy that prunes to a deterministic choice at each shift action, so we add a *correctness probability* to each shift operation, and train this score to discriminate between correct and incorrect sequences of parser actions. All other probability estimates are trained to optimize the conditional probability of the parse given the sentence. We also speed up both parsing and training times by only decomposing infrequent features, giving us both a form of backoff smoothing and twenty times faster training.

The discriminative training for this pruning strategy allows high accuracy to be preserved while greatly speeding up parsing time. The recurrent neural network architecture provides powerful automatic feature induction, resulting in high accuracy on diverse languages without tuning.

Acknowledgments

The research leading to this work was funded by the EC FP7 programme FP7/2011-14 under grant agreement no. 287615 (PARLANCE).

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March.

Danqi Chen 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. Association for Computational Linguistics, October.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04. Association for Computational Linguistics.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167. ACM.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November.

Ronan Collobert. 2011. Deep learning for efficient discriminative parsing. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 224–232. Journal of Machine Learning Research - Workshop and Conference Proceedings.

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

of the 22nd International Conference on Machine Learning, pages 169–176.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 513–520. ACM, June.

Jan Hajič, 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: Shared Task*, CoNLL '09, pages 1–18. Association for Computational Linguistics.

James Henderson and Ivan Titov. 2010. Incremental sigmoid belief networks for grammar learning. *J. Mach. Learn. Res.*, 11:3541–3570, December.

James Henderson, Paola Merlo, Gabriele Musillo, and Ivan Titov. 2008. A latent variable model of synchronous parsing for syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pages 178–182. Association for Computational Linguistics.

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Comput. Linguist.*, 39(4):949–998, December.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 24–31. Association for Computational Linguistics.

James Henderson. 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, July.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics, June.

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 Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289. Morgan Kaufmann Publishers Inc.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, volume 2010, pages 1045–1048. International Speech Communication Association.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In Hwee Tou Ng and Ellen Riloff, editors, *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 49–56. Association for Computational Linguistics, May 6 - May 7.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 221–225. Association for Computational Linguistics.

Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. 2007. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 759–766. ACM.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Ng, and Chris Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 129–136. ACM.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 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. Association for Computational Linguistics, August.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 649–652. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1017–1024. ACM, June.

Ivan Titov and James Henderson. 2007a. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 947–951. Association for Computational Linguistics, June.

Ivan Titov and James Henderson. 2007b. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, IWPT '07, pages 144–155. Association for Computational Linguistics.

Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. 2013. Recurrent neural networks for language understanding. In *INTERSPEECH*, pages 2524–2528.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Comput. Linguist.*, 37(1):105–151, March.