Recurrent Greedy Parsing with Neural Networks

Joël Legrand*+ and Ronan Collobert+

*Idiap Research Institute,
Rue Marconi 19, Martigny, Suisse

+Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland

joel.legrand@idiap.ch
ronan@collobert.com

Abstract. In this paper, we propose a bottom-up greedy and purely discriminative syntactic parsing approach that relies only on a few simple features. The core of the architecture is a simple neural network architecture, trained with an objective function similar to that of a Conditional Random Field. This parser leverages continuous word vector representations to model the conditional distributions of context-aware syntactic rules. The learned distribution rules are naturally smoothed, thanks to the continuous nature of the input features and the model. Generalization accuracy compares favorably to existing generative or discriminative (non-reranking) parsers (despite the greedy nature of our approach), while the prediction speed is very fast.

Keywords: Syntactic Parsing, Natural Language Processing, Neural Networks

1 Introduction

While discriminative methods are at the core of most state-of-the-art approaches in Natural Language Processing (NLP), historically the task of syntactic parsing has been mainly solved with generative approaches. A major contribution in the parsing field is certainly probabilistic context-free grammar (PCFGs)-based parsers [1–3]. These types of parsers model the syntactic grammar by computing statistics of simple grammar rules occurring in a training corpus. Inference is then achieved with a simple bottom-up chart parser. These methods face a classical learning dilemma: on one hand PCFG rules have to be refined enough to avoid any ambiguities in the prediction. On the other hand, too much refinement in these rules implies lower occurrences in the training set, and thus a possible generalization issue. PCFGs-based parsers are thus judiciously composing with carefully chosen PCFG rules and clever regularization tricks.

Given the success of discriminative methods for various NLP tasks, similar methods have been attempted for the syntactic parsing task. One of the first successful discriminative parsers [4] was based on MaxEnt classifiers (trained over a large number of different features) and a greedy shift-reduce strategy. However, it did not perform on par with the best generative parsers of the time. Costa *et al.* [5] introduced a left-to-right

incremental parser which used a recursive neural network to re-rank possible phrase attachments. They showed that RNN was able to capture enough information to make correct parsing decisions. Their system was, however, only tested on a subset of 2000 sentences. One had to wait a few more years before discriminative parsers could match Collins' parser performance. To this extent , Taskar *et al.* [6] proposed an approach which discriminates the entire space of parse trees, with a max margin criterion applied to Context Free Grammars. Other discriminative approaches [7, 8] also outperformed standard PCFG-based generative parsers, but only by discriminatively re-ranking the K-best predicted trees coming from a generative parser.

Turian and Melamed [9] later proposed a bottom-up greedy algorithm to construct the parse tree, using a feature boosting approach. The parsing is performed following a left-to-right or a right-to left strategy. The greedy decisions regarding the tree construction are made using decision tree classifiers. However, both of these parsers were limited to sentences of less than 15 words, due to a training time growing exponentially with the size of the input.

McClosky *et al.* [10] successfully leveraged unlabeled data to train a parser using a self-training technique. In this approach, a re-ranker is trained over a generative model. The re-ranker is used to generate "labels" over a large unlabeled corpus. These "labels" are then used to retrain the original generative model. This work is currently considered the state-of-the-art in syntactic parsing.

Most recent discriminative parsers [11, 12] rely on Conditional Random Fields (CRFs) with PCFG-like features. Carreras *et al.* [13] used a global-linear model (instead of a CRF) with PCFGs and various new advanced features.

While PCFG-based parsers are widely used, other approaches do exist. In [14], the proposed parser relies on continuous word vector representations, and a discriminative model to predict "levels" of the syntactic tree. Socher *et al.* [15] also relies on continuous word vector representations, which are "compressed" in a pairwise manner to form higher level chunk representations. Their approach is used as a re-ranker of the Stanford Parser [16].

Finally, it is worth noting that generative parsers are still evolving. PCFGs with latent-variables [17] have been used in various ways to improve the performance of classical PCFG as in [18].

In this paper, we propose a greedy and purely discriminative parsing approach. In contrast with most existing methods, it relies on a few simple features. The core of our architecture is a simple neural network which is fed with continuous word vector representations (as in [19, 15]). It models the conditional distributions of *context-aware* syntactic rules. The learned distribution rules are naturally smoothed, thanks to the continuous nature of the input features.

Section 2 introduces our algorithm and relates it to PCFG-based parsers. Section 3 describes the classification model at the core of our architecture. Section 4 reports experimental comparisons with existing approaches. We conclude in Section 5.

2 A greedy discriminative parser

2.1 Smoothed Context Rule Learning

PCFG-based parsers rely on the statistical modeling of rules of the form $A \rightarrow B,\ C$, where A, B and C are tree nodes. The context-free grammar is always normalized in the Chomsky Normal Form (CNF) to make the global tree inference practical (with a dynamic programming like CYK or similar). In general a tree node is represented as several features, including for example its own parsing tags and head word (for non-terminal nodes) or word and Part Of Speech (POS) tag (for terminal nodes) [2]. State-of-the-art parsers rely on a judicious blending of carefully chosen features and regularization: adding features in PCFG rules might resolve some ambiguities, but at the cost of sparser occurrences of those rules. In that respect, the learned distributions must be carefully smoothed so that the model can generalize on unseen data. Some parsers also leverage other types of features (such as bigram or trigram dependencies between words [13]) to capture additional regularities in the data.

In contrast, our system models non-CNF rules of the form $A \rightarrow B_1, ..., B_K$. The score of each rule is determined by looking at a large context of tree nodes. More formally, we learn a classifier of the form:

$$f(C_{left}, B_1, ..., B_K, C_{right}) = (s_1, ..., s_{|\mathcal{T}|}) \tag{1}$$

where the B_k are either terminal or non-terminal nodes, K is the size of the right part of the rule, C_{left} and C_{right} are context terminals or non-terminals and s_t is the score for the parsing tag $t \in \mathcal{T}$. Each possible rule $A_i \rightarrow B_1, ..., B_K$ is thus assigned a score s_i by the classifier (with $A_i \in \mathcal{T}$). These scores can be interpreted as probabilities by performing a softmax operation. We used a Multi Layer Perceptron (MLP) as classifier. Formal details will be given in Section 3.2.

The only tree node features considered in our system are parsing tags (or POS tags for terminals), as well as the headword (or words for terminals). We overcome the problem of data sparsity which occurs in most classical parsers by leveraging *continuous vector representations* for all features associated to each tree node. In particular, word (or headword) representations are derived from recent distributed representations computed on large unlabeled corpora (such as [19, 20]). Thanks to this approach, our system can naturally generalize a rule like $NP \rightarrow a,\ clever,\ guy$ to a possibly unseen rule like $NP \rightarrow a,\ smart,\ guy$, as the vector representation of $smart$ and $clever$ are close to each other, given that they are semantically and syntactically related.

Several works leveraging continuous vector representations have been previously proposed for syntactic parsing. [14] introduced a neural network-based approach, iteratively tagging "levels" of the parse tree where the full sentence was seen at each level. A complex pooling approach was introduced to capture long-range dependencies, and performance only matched early lexicalized parsers. [21] introduced a recursive approach, where representations are "compressed" two by two to form higher-level representations. However, the system was limited to bracketing, and did not produce parsing tags. The authors later proposed an improved version in [15], where their approach was used to re-rank the output of the Stanford Parser, approximately reaching state-of-the-art

performance. In contrast, our approach does not rely on CNF grammars and does not re-rank an external generative parser.

2.2 Greedy Recurrent Algorithm

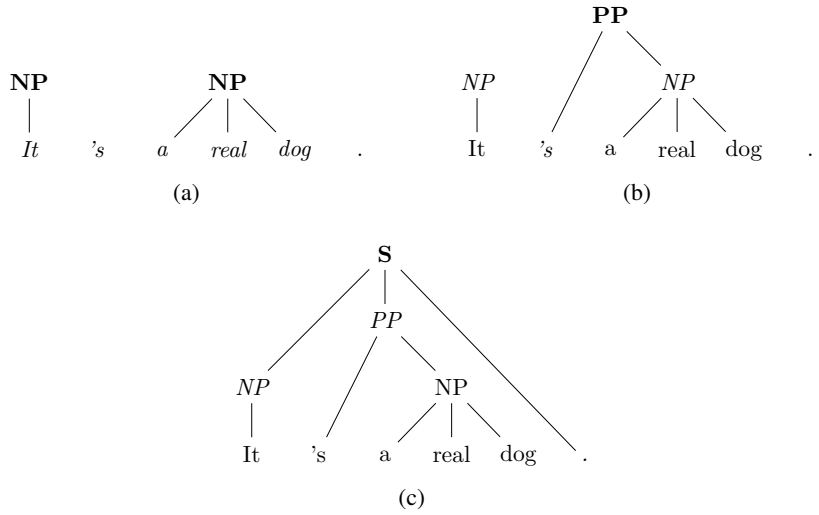

Fig. 1: Illustration of our greedy algorithm: at each iteration (a)→(b)→(c), the classifier sees only the previous tree heads (ancestors), shown here in italics. It predicts new nodes (here in bold). New tree heads become the ancestors at the next iteration. All other previously discovered tree nodes (shown in regular black here) will remain unchanged and ignored in subsequent iterations.

Our parser follows a bottom-up iterative approach: the tree is constructed starting from the terminal nodes (sentence words). Assuming that a part of the tree has been already predicted (see Figure 1), the next iteration of our algorithm looks for all possible new tree nodes which combine ancestors (i.e., heads of the trees predicted so far). New nodes are found by maximizing the score of our context-rule classifier (1), constrained in such a way so that two new nodes cannot overlap, thanks to a dynamic programming approach. The system is recurrent, in the sense that new predicted parsing labels are used in the next iteration of our algorithm.

For each iteration, assuming N ancestors

$$X = [X_1, ..., X_N] \,,$$

finding all possible new nodes with K ancestors would require to apply

$$f(C_{left}, B_1, ..., B_K, C_{right})$$

over all possible windows of K ancestors in X. One would also have to vary K from 1 to N, to discover new nodes of all possible sizes. Obviously, this could quickly become time consuming for large sentence sizes. This problem of finding nodes with a various number of ancestors can be viewed as the classical NLP problem of finding "chunks" of various sizes. This problem is typically transformed into a tagging task: finding the chunk with label A in the rule $A \rightarrow X_i, X_{i+1}, \ldots, X_j$ can simply be viewed as tagging the ancestors with B-A, I-A, $\ldots E$-A, where we use the standard BIOES label prefixing (B*egin*, I*ntermediate*, O*ther*, E*nd*, S*ingle*). See Table 1 for a concrete example. The classifier outputs the "O*ther*" tag, when the considered ancestors do not correspond to any possible rule.

In the end, our approach can be summarized as the following iterative algorithm:

1. Apply a sliding window over the current ancestors: the neural network classifier (1) is applied over all K consecutive ancestors $X_1, ..., X_N$, where K has to be carefully tuned.
2. Aggregate BIOES tags into chunks: a dynamic program (based on a CRF, as detailed in Section 3.3) finds the most likely sequence of BIOES parsing tags. The new nodes are then constructed by simply aggregating BIES tags

$$B\text{-}A, I\text{-}A, \ldots E\text{-}A$$

   into A (for any label A).
3. Ancestors tagged as O, as well as newly found tree nodes are passed as ancestors to the next iteration.

The tree construction ends when there is only one ancestor remaining, or when the classifier did not find any new possible rule (everything is tagged as O).

Table 1: A simple example of a grammar rule extracted from the sentence "*It 's a real dog .*", and its corresponding BIOES grammar. In both cases, we include a left and right context of size 1. The middle column shows the required classifier evaluations. The right column shows the type of scores produced by the classifier.

GRAMMAR	CLASSIFIER EVALUATIONS	SCORES
NP $\rightarrow$ 'S **A REAL DOG** .	f('S, **A**, **REAL**, **DOG**, .)	$s_{NP}, ..., s_{VP}, s_O$
B-NP $\rightarrow$ 'S **A REAL**	f('S, **A**, **REAL**)	$s_{B\text{-}NP}, ..., s_{E\text{-}VP}, s_O$
I-NP $\rightarrow$ **A REAL DOG**	f(**A**, **REAL**, **DOG**)	
E-NP $\rightarrow$ **REAL DOG .**	f(**REAL**, **DOG**, .)	

3 Architecture

In this section, we formally introduce the classification architecture used to find new tree nodes at each iteration of our greedy recurrent approach. A simple two-layer neural

network is at the core of the system. It leverages continuous vector word representations. In this respect, the network is clearly inspired by the work of [22] in the context of language modeling, and later re-introduced in [23] for various NLP tagging tasks.

Given an input sequence of N tree node ancestors $X_1, ..., X_N$ (as defined in Section 2.2), our model outputs a BIOES-prefixed parsing tag for each ancestor X_i, by applying a sliding window approach. These scores are then fed as input to a properly constrained graph on which we apply the Viterbi algorithm to infer the best sequence of parsing tags. The whole architecture (including transition scores in the graph) is trained in an end-to-end manner by maximizing the graph likelihood. The end-to-end neural network training approach was first introduced in [24]. The system can be also viewed as a particular Graph Transformer Network [25], or a particular *non-linear* Conditional Random Field (CRF) for sequences [26]. Each layer of the architecture is presented in detail in the following paragraphs. The objective function will be introduced in Section 3.4.

3.1 Words Embeddings

Our system relies on *raw words*, following the idea of [19]. Each word is mapped into a continuous vector space. For efficiency, words are fed into our architecture as indices taken from a finite dictionary $\mathcal{W}$. Word vector representations, as other network parameters, are trained by back-propagation.

More formally, given a sentence of N words, $w_1, w_2, ..., w_N$, each word $w_n \in \mathcal{W}$ is first embedded in a D-dimensional vector space by applying a lookup-table operation:

$$LT_W(w_n) = W_{w_n} ,$$

where the matrix $W \in \mathbb{R}^{D \times |\mathcal{W}|}$ represents the parameters to be trained in this lookup layer. Each column $W_n \in \mathbb{R}^D$ corresponds to the vector embedding of the n^{th} word in our dictionary $\mathcal{W}$.

These types of architectures allow us to take advantage of word vector representations trained on large unlabeled corpora, by simply initializing the word lookup table with a pre-trained representation [19]. In this paper, we chose to use the representations from [27], obtained by a simple PCA on a matrix of word co-occurrences. As shown in [14] for various NLP tasks, we will see that these representations can provide a great boost in parsing performance.

In practice, it is common to give several features (for each tree node) as input to the network. This can be easily done by adding a different lookup table for each discrete feature. The input becomes the concatenation of the outputs of all these lookup-tables:

$$\begin{aligned} LT_{W_1,...,W_K}(w_n) =& (LT_{W_1}(w_n))^T, \\ & ..., \\ & (LT_{W_{|\mathcal{F}|}}(w_n))^T \end{aligned}$$

where $|\mathcal{F}|$ is the number of features. For simplicity, we consider only one lookup-table in the rest of the architecture description.

3.2 Sliding Window BIOES Tagger

The second module of our architecture is a simple neural network which applies a sliding window over the output of the lookup tables, as shown in Figure 2. The n^{th} window is defined as

$$u_n = [LT(X_{n-\frac{K-1}{2}}), ..., LT(X_n), ..., LT(X_{n+\frac{K-1}{2}})]\,,$$

where K is the size of window. The module outputs a vector of scores $s(u_n) = [s_1, ..., s_{|\mathcal{T}|}]$ (where s_t is the score of the BIOES-prefixed parsing tag $t \in \mathcal{T}$ for the ancestor X_n). The ancestors with indices exceeding the input boundaries ($n - (K-1)/2 < 1$ or $n + (K-1)/2 > N$) are mapped to a special padding vector (which is also learned). As any classical two-layer neural network, our architecture performs several matrix-vector operations on its inputs, interleaved with some non-linear transfer function $h(\cdot)$,

$$s(u_n) = M_2\, h(M_1\, u_n)\,,$$

where the matrices $M_1 \in \mathbb{R}^{H \times K|D|}$ and $M_1 \in \mathbb{R}^{|\mathcal{T}| \times H}$ are the trained parameters of the network. The number of hidden units H is a hyper-parameter to be tuned.

As transfer function, we chose in our experiments a (fast) "hard" version of the hyperbolic tangent:

$$h(x) = \begin{cases} -1 \text{ if } & x < -1 \\ x \text{ if } & -1 <= x <= 1 \\ 1 \text{ if } & x > 1 \end{cases} \tag{2}$$

3.3 Aggregating BIOES Predictions

The scores obtained from the previous module of our architecture are in BIOES format. The next module in our system aggregates these tags and finds the new tree nodes at each iteration of our greedy recurrent approach. We introduce a graph G of scores as shown in Figure 3: each node of the graph corresponds to a BIOES score produced for each ancestor by the neural network module. This graph is constrained in such a way that only feasible sequences of tags are possible (e.g. B-A tags can only be followed by I-A tags, for any parsing label A). Our graph also includes a duration model: on each edge, we add a transition score $A_{tt'}$ for jumping from tag $t \in \mathcal{T}$ to $t' \in \mathcal{T}$.

A score for a sequence of tags $[t]_1^N$ in the lattice G is obtained as the sum of scores along $[t]_1^N$ in G:

$$S([t]_1^N, [u]_1^N, \theta) = \sum_{n=1}^{N} \left(A_{t_{n-1}t_n} + s(u_n)_{t_n}\right),$$

where θ represents all the trainable parameters of the complete architecture. The sequence of tags $[t^*]_1^N$ for the input sequence of tree node ancestors $X_1, \ldots, X_N$ is then inferred by finding the path which leads to the maximal score:

$$[t^*]_1^N = \underset{[t]_1^N \in \mathcal{T}^N}{\operatorname{argmax}}\, S([t]_1^N, [u]_1^N, \theta)$$

The Viterbi algorithm is the natural choice for this inference. From this optimal BIOES tag sequence, we extract sub-sequences B-A, $\ldots$, E-A and S-A as new nodes for the tree. O tags are simply ignored. See Section 2.2 for more details.

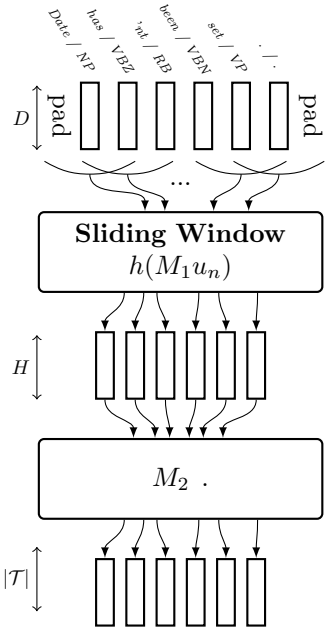

Fig. 2: Sliding window tagger. Given the concatenated output of lookup tables (here the ancestor words/headwords and ancestor tags), the tagger outputs a BIOES-prefixed parsing tag for each ancestor node. The neural network itself is a standard two-layer neural network.

3.4 Training Likelihood

Our architecture sees sequences of ancestor tree nodes, and outputs new possible syntactic tree nodes only from this history. Technically speaking, the training set can be prepared by iterating over each tree in the training corpus, removing all possible leaves in an iterative process so that all training rules are uncovered (see Figure 4).

The neural network is trained by maximizing a likelihood over the training data, using stochastic gradient ascent. The score for a path can be interpreted as a conditional probability over this path by exponentiating score (thus making it positive) and normalizing it with respect to all possible paths. We define $\mathcal{P}$ as the set of possible tag paths in the constrained graph G, as shown in Figure 3. The log-probability of a sequence of tags $[t]_1^N$ given the lookup table representations $[u]_1^N$ is given by:

$$\begin{aligned} \log P([t]_1^N | [u]_1^N, \theta) = & S([t]_1^N, [u]_1^N, \theta) \\ & - \underset{\forall [t']_1^N \in \mathcal{P}}{\text{logadd}}\ S([t']_1^N, [u]_1^N, \theta)) \end{aligned} \tag{3}$$

where we adopt the notation $\text{logadd}_{z_n} = \log\left(\sum_i e^{z_i}\right)$, as in [23].

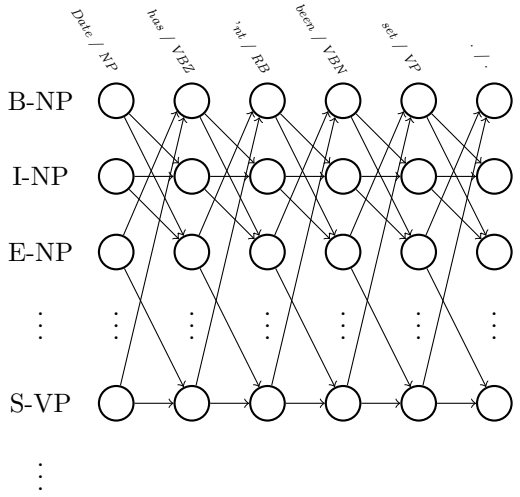

Fig. 3: Constrained graph for tag inference. Only feasible sequences of tags are considered. The nodes of the graph are assigned a score from the tagger shown in Figure 2. Edges of the graph are assigned a transition score which is learned similarly to other parameters in the architecture.

Computing the log-likelihood efficiently is not straightforward, as the number of terms in the logadd grows exponentially with the length of the sentence. Fortunately, it can be computed in linear time with the Forward algorithm, which derives a recursion similar to the Viterbi algorithm (see [28]). The complete architecture is trained by simply backpropagating through this recursion, up to the lookup layers (for further details, see [14]). Note that the likelihood (3) corresponds to a standard CRF model for sequences. The only difference here is that the underlying model is non-linear, while CRFs are often considered as linear models.

4 Experiments

4.1 Corpus

Experiments were performed using the standard English Penn Treebank data set (Marcus et al., 1993). We used the classical parsing setup, with sections 02-21 used to train our model, section 22 used as validation for choosing all our hyper-parameters, and section 23 used for testing. We applied only a small subset of the typical pre-processing set over the data: (1) functional labels, traces were removed, (2) the PRT label was replaced as ADVP [1].

The Penn Treebank data set contains non-terminal tree nodes which only have one non-terminal child, as shown in Figure 5. To avoid possible looping issues in our parsing

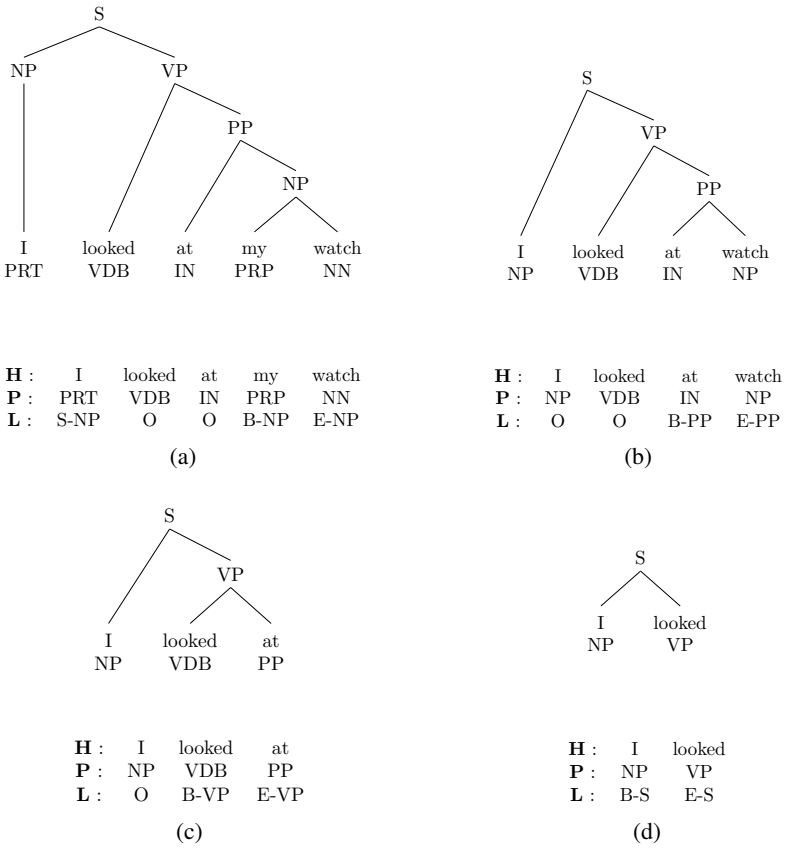

Fig. 4: Iterative procedure (a)→(b)→(c)→(d) to generate the training data, which involves cutting out all tree leaves at each step. The data fed to our network architecture is then easily uncovered (H: ancestor headwords/words, P: ancestor POS/parsing tags, L: parsing labels to be predicted).

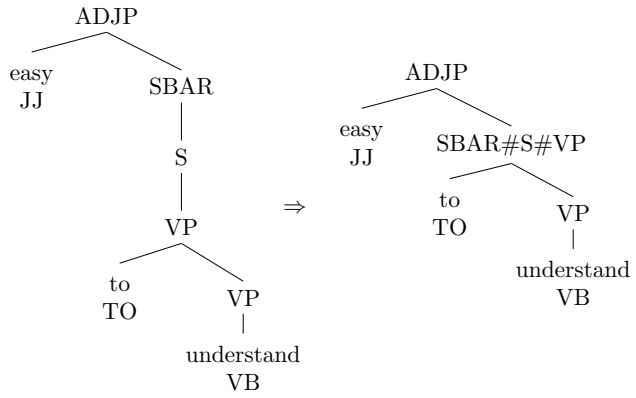

Fig. 5: Training corpus pre-processing. Original Penn Treebank trees containing non-terminal nodes with only one non-terminal node (left), and after concatenating those nodes (right).

algorithm (e.g. a node being repetitively tagged with two different tags in our iterative process), we transformed the *training* corpus so that non-terminal nodes having only one non-terminal child were merged together, and take as tag the concatenation of all merged node tags (see Figure 5). This way, the system learns that a node must contain at least two ancestors. The iterative process is thus guaranteed to converge. We kept only concatenated labels which occurred at least 30 times (corresponding to the lowest number of occurrences of the less common original parsing tag), leading to 11 additional parsing tags. Added to the original 26 parsing tags, this resulted in 161 tags produced by our parser. At test time, the inverse operation is performed: concatenated tag nodes are simply expanded into their original form.

4.2 Features

We consider the following features to train our architecture:

- Words and headwords:
  - For terminal nodes, the word itself, in low caps[1]. As in [2], words occurring 5 times or less were mapped to an "UNKNOWN" word.
  - For non-terminal nodes: headwords, following the procedure described in [2].
- POS tags (for terminals) or parsing tags of the node's ancestors (for non-terminals). POS tags were produced with SENNA [23].
- POS tags of headwords.

[1] Adding a capital feature had no impact on the performance of our parser. Note that POS tags were generated with the original caps in the sentence.

4.3 Results

We train the network using stochastic gradient descent over the available training data, until convergence on the validation set. We chose the following hyper-parameters according to the validation. Lookup-table sizes for the words and tags (part-of-speech and parsing) are 100 and 20, respectively. The window size for the tagger is $K = 7$ (3 neighbors from each side). The size of the tagger's hidden layer is $H = 500$. We used the word embeddings obtained from [27] to initialize the word lookup-table. These embeddings were then fine-tuned during the training process. Finally, we fixed the learning rate to $\lambda = 0.025$ during the stochastic gradient procedure. The only "trick" used during training was to divide the learning rate by the input size of each linear layer [29].

Table 2 shows the importance of the different features we used. Even though the training procedure is non-convex, the variance of the F1 score over 20 different runs (for the architecture Word + POS + hw + wi) was only 0.01.

Table 2: Influence of different features. Results are given in terms of F1-score. POS = part-of-speech, hw = head-word, wi = word initialization from [27].

FEATURE	F1
WORD + POS	85.1
WORD + POS + HW	86.9
WORD + POS + WI	86.2
WORD + POS + HW + WI	88.3

Since our architecture performs the decoding very quickly, we additionally performed a voting procedure using several models learned from different random initializations. We averaged all neural network classifiers (ignoring their own respective CRF decoding part) and trained a new CRF on top of it (without fine-tuning any of the neural network classifiers). The scores obtained with 10 classifiers are shown in Table 3.

Results in Table 3 are reported in terms of recall (R), precision (P) and F1 score. Scores were obtained using the Evalb implementation[2]. We compare our system with several other parsers. We chose to report the scores of the three main generative parsers, as well as those of known re-ranking parsers. We also considered two major purely discriminative parsers.

[2] Available at http://nlp.cs.nyu.edu/evalb/

Table 3: Results in terms of Precision (P), Recall (R), and F1 score. The reported time is the time to parse the full WSJ test corpus.

	MODEL	(R)	(P)	F1	(R)	(P)	F1	TIME
GENERATIVE	MAGERMAN (1995)	84.6	84.9	84.8				
	COLLINS (1999)	88.5	88.7	88.6	88.1	88.3	88.2	1247
	CHARNIAK (2000)	90.1	90.1	90.1	89.6	89.5	89.6	
GENERATIVE WITH RE-RANKING	HENDERSON (2004)				89.8	90.4	90.1	
	CHARNIAK AND JOHNSON (2005)			92.0			91.1	
	SOCHER ET AL (2013)			91.1			92.1	390
	MCCLOSKY ET AL (2006)						92.1	
PURELY DISCRIMINATIVE	PETROV AND KLEIN (2008)			90.0			89.4	
	CARRERAS ET AL. (2008)				90.7	91.4	91.1	
	OUR MODEL	88.4	89.0	88.7	88.0	88.6	88.3	110
	OUR MODEL (VOTING)	90.0	90.1	90.1	89.6	89.7	89.6	

4.4 Rule Prediction Analysis

Figure 6 shows the output of the classifier (applied on every possible window of size 7) for the sentence "When the little guy gets frightened, the big guys hurt badly.". For this sentence, the expected rule are the following:

$$
\begin{aligned}
\text{WHADVP} &\rightarrow \text{When} \\
\text{NP} &\rightarrow \text{the little guy} \\
\text{ADJP} &\rightarrow \text{frightened} \\
\text{NP} &\rightarrow \text{the big guys} \\
\text{ADVP} &\rightarrow \text{badly}
\end{aligned}
$$

It is interesting to see that the network alone is able to predict all the rules of the sentence. The CRF is however essential to produce a consistent output, by aggregating BIES prefixed chunks.

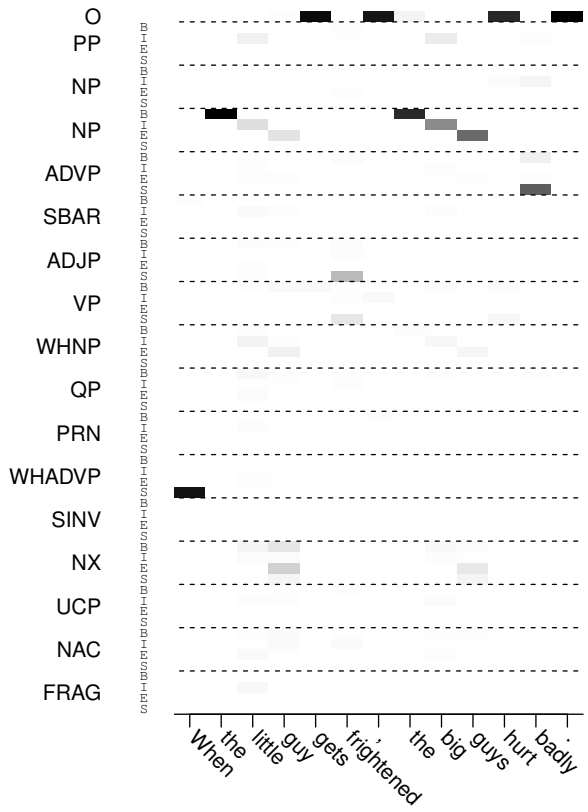

Fig. 6: Normalized scores from the network classifier (black means high score) for the sentence ”When the little guy gets frightened, the big guys hurt badly.”. Each tag is in BIOES form (y axis). Each ancestor in the input is on the x axis.

5 Conclusion

We presented a very simple model that is able to learn syntactic grammar rules surprisingly well, considering the simple features employed. This parser achieves performance very close to state-of-the-art re-ranking systems and is almost the best among the purely generative parsers. Due to its simplicity, there are many possibilities for further improvement. In particular, the head-word procedure from Collins could be revisited, e.g. by learning a higher-level chunk representation in the same spirit as [15]. We could also investigate re-ranking approaches, as well as the use of unlabeled corpora.

Acknowledgments. This work was supported by NEC Laboratories America. We would like to thank Leonidas Lefakis and Pedro Oliveira Pinheiro for proofreading this paper and Dimitri Palaz for his contribution on figures 2 and 3.

References

1. D. M. Magerman, "Statistical decision-tree models for parsing," in *In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995.
2. M. Collins, "Head-driven statistical models for natural language parsing," *Comput. Linguist.*, 2003.
3. E. Charniak, "A maximum-entropy-inspired parser," in *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, 2000.
4. A. Ratnaparkhi, "Learning to parse natural language with maximum entropy models," *Mach. Learn.*, Feb. 1999.
5. F. Costa, P. Frasconi, V. Lombardo, and G. Soda, "Towards incremental parsing of natural language using recursive neural networks," 2002.
6. B. Taskar, D. Klein, M. Collins, D. Koller, and C. D. Manning, "Max-margin parsing," in *In Proceedings of EMNLP*, 2004.
7. J. Henderson, "Discriminative training of a neural network statistical parser," in *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, 2004.
8. E. Charniak and M. Johnson, "Coarse-to-fine N-best parsing and MaxEnt discriminative reranking," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005.
9. J. Turian and I. D. Melamed, "Advances in discriminative parsing," in *In Proceedings of the Joint International Conference on Computational Linguistics and Association of Computational Linguistics (COLING/ACL*, 2006.
10. D. McClosky, E. Charniak, and M. Johnson, "Effective self-training for parsing," in *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, 2006.
11. J. R. Finkel, A. Kleeman, and C. D. Manning, "Efficient, feature-based, conditional random field parsing," in *In Proc. ACL/HLT*, 2008.
12. S. Petrov and D. Klein, "Sparse multi-scale grammars for discriminative latent variable parsing," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.
13. X. Carreras, M. Collins, and T. Koo, "TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing," in *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, 2008.

14. R. Collobert, "Deep learning for efficient discriminative parsing," in *AISTATS*, 2011.
15. R. Socher, J. Bauer, C. Manning, and A. Ng, "Parsing With Compositional Vector Grammars," in *ACL*, 2013.
16. D. Klein and C. Manning, "Accurate unlexicalized parsing," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, 2003.
17. T. Matsuzaki, Y. Miyao, and J. Tsujii, "Probabilistic CFG with latent annotations," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005.
18. S. B. Cohen and M. Collins, "Tensor decomposition for fast parsing with latent-variable PCFGs," in *Proceedings of NIPS*, 2012.
19. R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *International Conference on Machine Learning, ICML*, 2008.
20. P. S. Dhillon, D. Foster, and L. Ungar, "Multi-view learning of word embeddings via cca," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.
21. R. Socher, C. Lin, A. Y. Ng, and C. D. Manning, "Parsing natural scenes and natural language with recursive neural networks.," in *ICML*, 2011.
22. Y. Bengio and R. Ducharme, "A neural probabilistic language model," in *NIPS 13*, 2001.
23. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, 2011.
24. J. S. Denker and C. J. C. Burges, "Image segmentation and recognition," in *In The Mathematics of Induction*, 1995.
25. L. Bottou, Y. LeCun, and Y. Bengio, "Global training of document processing systems using graph transformer networks," in *Proc. of Computer Vision and Pattern Recognition*, 1997.
26. J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Eighteenth International Conference on Machine Learning, ICML*, 2001.
27. R. Lebret and R. Collobert, "Word embeddings through hellinger PCA," in *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 2014.
28. L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proceedings of the IEEE*, pp. 257–286, 1989.
29. D. C. Plaut and G. E. Hinton, "Learning sets of filters using back-propagation," *Computer Speech and Language*, 1987.