N-gram Parsing for Jointly Training a Discriminative Constituency Parser

Arda Çelebi and Arzucan Özgür

***Abstract*—Syntactic parsers are designed to detect the complete syntactic structure of grammatically correct sentences. In this paper, we introduce the concept of n-gram parsing, which corresponds to generating the constituency parse tree of n consecutive words in a sentence. We create a stand-alone n-gram parser derived from a baseline full discriminative constituency parser and analyze the characteristics of the generated n-gram trees for various values of n. Since the produced n-gram trees are in general smaller and less complex compared to full parse trees, it is likely that n-gram parsers are more robust compared to full parsers. Therefore, we use n-gram parsing to boost the accuracy of a full discriminative constituency parser in a hierarchical joint learning setup. Our results show that the full parser jointly trained with an n-gram parser performs statistically significantly better than our baseline full parser on the English Penn Treebank test corpus.**

***Index Terms*—Constituency parsing, n-gram parsing, discriminative learning, hierarchical joint learning.**

I. Introduction

PARSING a natural language sentence is a process of characterizing the syntactic description of that sentence based on the syntax of its language. Over the last half-century, there have been many techniques developed to improve parsing accuracy. Some of the studies have targeted the model that the parser relies on, such as by replacing rule-based approaches [1], [2] with statistical models like generative [3], [4] and discriminative ones [5], [6]. Others introduced external ways of boosting the parser, such as by using a reranker [7], [8], by bootstrapping it with itself in a self-training setup [9], or by using partial parsing in a co-training setup [10]. Another recent thread of research is about a more specialized form of the co-training approach, where multiple models from different domains are jointly trained together and help each other to do better. One example is [11], where they introduce the Hierarchical Joint Learning (HJL) approach to jointly train a parser and a named entity recognizer. Their HJL model achieved substantial improvement in parsing and named entity recognition compared to the non-jointly trained models.

In this paper, we aim to improve the accuracy of a discriminative constituency parser by training it together with another parser in the HJL setup. While our actual parser works on complete sentences, its accompanying parser tackles the parsing task in a less complex way, that is, by parsing n-grams instead of complete sentences. To the best of our knowledge, this is the first study that introduces the concept of n-gram parsing. Even though syntactic parsers expect grammatically correct and complete sentences, an n-gram parser is designed to parse only n consecutive words in a sentence. An outputted n-gram tree is still a complete parse tree, but it covers only n words instead of the whole sentence. We derive our n-gram parser from a discriminative parser which was implemented based on [12]. After analyzing the characteristics of n-gram parsing, we train the full parser together with the n-gram parser. Our underlying hypothesis is that the n-gram parser will help the full parser at cases where the n-gram parser does better. We performed experiments with different n-gram sizes on the English Penn treebank corpus [13] and obtained a statistically significant increase in the accuracy of the jointly trained full parser over the original (non-jointly trained) full parser.

This paper continues with the related studies. Following that, in Section III, we introduce the concept of n-gram parsing and the characteristics of the n-gram trees. In Sections IV and V, we describe how we perform discriminative constituency parsing and how we use the HJL approach, respectively. Before discussing the experiments, we introduce the data and the evaluation methods that we used in Section VI. We present the experimental results obtained with the n-gram parser alone and the jointly trained parser. We conclude and outline future directions for research in the last section.

II. Related Work

In this paper we tackle the problem of improving the performance of a discriminative constituency parser by training it with an n-gram parser using hierarchical joint learning. Although generative models [3], [4] still dominate the constituency parsing area due to their faster training times, a number of discriminative parsing approaches have been proposed in the recent years motivated by the success of discriminative learning algorithms for several NLP tasks such as part-of-speech tagging and relation extraction. An advantage of discriminative models is their ability to incorporate better feature rich representations. There are three different approaches for applying discriminative models to the parsing task. The first and perhaps the most successful one

Manuscript received on December 7, 2012; accepted for publication on January 11, 2013.

Arda Çelebi and Arzucan Özgür are with Department of Computer Engineering, Boğaziçi University, Bebek, 34342 İstanbul, Turkey (e-mail: {arda.celebi, arzucan.ozgur}@boun.edu.tr).

is to use a discriminative reranker to rerank the n-best list of a generative parser [5], [7], [6]. To our knowledge, the forest-based reranker in [8] is the best performing reranker that helps its accompanying parser to achieve an F_1 score of 91.7%[1]. The second approach considers parsing as a sequence of independent discriminative decisions [14], [15]. By discriminative training of a neural network based statistical parser, an F_1 score of 90.1% is obtained in [15]. The third approach, which we adapted for this paper from [12], is to do joint inference by using dynamic programming algorithms in order to train models and use these to predict the globally best parse tree. With this, an F_1 score of 90.9% is achieved in [12]. Due to their notoriously slow training times, however, most discriminative parsers run on short sentences. That is why we use sentences that have no more than 15 words for full sentence parsing in this study.

One of the aspects of our research is that it involves working with multiple parsers at the same time, that is the n-gram parser and the full sentence parser. There have been a couple of studies that experimented with multiple parsers in the literature. One example is [16] which extended the concept of rerankers by combining k-best parse outputs from multiple parsers and achieved an F_1 score of 92.6%. In [17], Fossum and Knight worked on multiple constituency parsers by proposing a method of parse hybridization that recombines context-free productions instead of constituents in order to preserve the structure of the output of the individual parsers to a greater extent. With this approach, their resulting parser achieved an F_1 score of 91.5%, which is 1 point better than the best individual parser in their setup.

Another thread of research related to ours is jointly training multiple models. This evolved from the concept of multi-task learning, which is basically explained as solving multiple problems simultaneously. In the language processing literature, there have been a couple of studies where this concept is adapted for multi-domain learning [18], [19], [20]. In these studies, they make use of labelled data from multiple domains in order to improve the performances on all of them. In [11], for example, a joint discriminative constituency parser alongside a named-entity recognizer is used and substantial gains over the original models is reported. Like most of the prior research, a derivative of the hierarchical model, namely the Hierarchical Joint Learning (HJL) approach is used. In this paper, we adapted this approach by replacing the named-entity recognizer with an n-gram parser, which to our knowledge hasn't been attempted before in the literature.

The most distinguishing contribution of our research is the introduction of n-gram parsing. To the best of our knowledge, n-gram parsing has never been considered as a stand-alone parsing task in the literature before. One reason might be that n-gram trees have no particular use on their own. However, they have been used as features for statistical models in either lexicalized or unlexicalized forms. For example, in [9] they are used to train the reranking model of the self-training pipeline. There are only a couple of studies in the literature comparable to our notion of n-gram trees. One of them is the stochastic tree substitution grammars (TSG) used in Data Oriented Parsing (DOP) models in [21]. However, unlike TSG trees, our n-gram trees always have words at the terminal nodes. Another related concept is the Tree Adjunct Grammar (TAG) and the concept of local trees proposed in [22]. As in the case of TSG trees, TAG local trees also differ from our n-gram trees by not having words at all terminal nodes but one. Another related study was performed in [23], where significant error reductions in parsing are achieved by using n-gram word sequences obtained from the Web.

In the literature, the concept of n-grams is used in a number of contexts to represent a group of n consecutive items. This can be, for instance, characters in a string or words in a sentence. In our research, we consider an n-gram as n consecutive words selected from a sentence. n-gram parsing, then, refers to the process of predicting the syntactic structure that covers these n words. We call this structure an n-gram tree. In this paper, we study the parsing of 3- to 9-grams in order to observe how n-gram parsing differs with length and at which lengths the n-gram parser helps the full parser more. A sample 4-gram tree which is extracted from a complete parse tree is shown in Figure 1. Compared to complete parse trees, n-gram trees are smaller parse trees with one distinction. That is, they may include constituents that are trimmed from one or both sides in order to fit the constituent lengthwise within the borders of the n-gram. We call such constituents incomplete, and denote them with the -INC functional tag. n-gram parsing is fundamentally no different than the conventional parsing of a complete sentence. However, n-grams, especially the short ones, may have no meanings on their own and/or can be ambiguous due to the absence of the surrounding context. Even though the relatively smaller size of n-gram trees makes it easier and faster to train on them, their incomplete and ambiguous nature makes the n-gram parsing task difficult. Despite all, n-gram parsing can still be useful for the actual full sentence parser, just like the partial parsing of a sentence used for bootstrapping [10]. In this paper, we train the full parser together with an n-gram parser and let the n-gram parser help the full parser at areas where the n-gram parser is better than the full one.

A. *N-gram Tree Extraction Algorithm*

In order to generate a training set for the n-gram parser, we extract n-gram trees out of the complete parse trees of the Penn Treebank, which is the standard parse tree corpus used in the literature [13]. Since we use sentences with no more than 15 words (WSJ15) for complete sentence parsing, we use the rest of the corpus (WSJOver15) for n-gram tree extraction.

The pseudocode of our n-gram tree extraction algorithm is given in Fig. 2. It takes a complete parse tree as

[1]The performance scores reported in this section are for Section 23 of the English Penn Treebank.

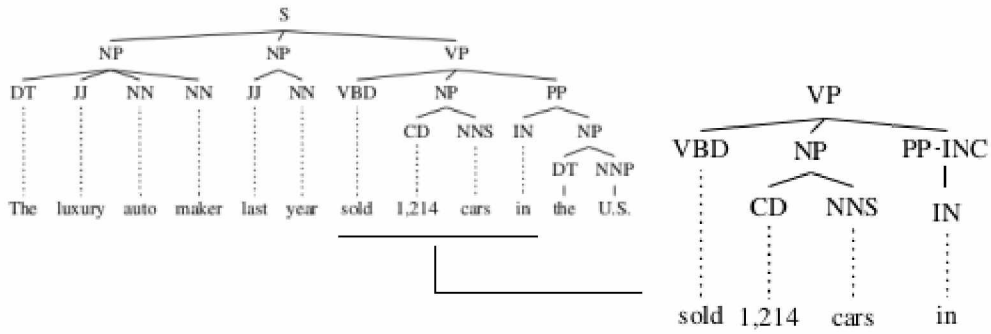

Fig. 1. Sample 4-gram tree extracted from a complete parse tree

Require: n, width of the n-gram trees
Require: $tree$, parse tree of a sentence
  $len \leftarrow$ length of the given sentence
  $i \leftarrow 0$
  while $i < len - n$ **do**
    $subtree \leftarrow$ get subtree that covers $[i, i+n]$ span
    $trimmed \leftarrow$ trim $subtree$'s constituents outside the $[i, i+n]$ span, if any
    if $trimmed$ has any const. with no head child **then**
      i++ and continue
    end if
    $markedtree \leftarrow$ mark all trimmed consts. as incomplete
    $filtered \leftarrow$ filter out incomplete unary rule chain from the ROOT, if any
    save $filtered$ tree as n-gram tree
    i++
  end while

Fig. 2. Extracting and storing n-gram trees from a parse tree

input and returns all the extracted valid n-gram trees. It starts by traversing the sentence from the first word and preserves the minimum subtree that covers n consecutive words. While doing that, it may trim the tree from one or both sides in order to fit the constituents lengthwise within the borders of the n-gram. Hence, the extracted n-gram trees may contain incomplete and thus ungrammatical constituents, which is not something that a conventional parser expects as input. Nevertheless, we assume that not all of the trimmed constituents are ungrammatical according to the concept of generatively accurate constituents that we introduce in this paper. This concept stems from the concept of the head-driven constituency production process [3] where a constituent is theoretically generated starting from its head child and continuing towards the left and right until all children are generated. If the head child is the origin of the production, then it is safe to say that it defines the constituent. Therefore, our n-gram tree extraction process makes sure that the head child is still included in the trimmed constituents. Otherwise, the whole n-gram tree is considered generatively inaccurate and is thus discarded. If all the heads are preserved, the algorithm marks the trimmed constituents with the -INC functional tag. For example, the PP constituent of the 4-gram tree in Figure 1 is trimmed from right hand side and since the head child "IN" is still included in the constituent, it is considered generatively accurate. The corresponding constituent is marked with an -INC tag and the extracted tree is stored as a valid 4-gram tree. However, if we try to extract the next 4-gram in the same sentence, it would have failed due to not being able to keep the head of the rightmost NP. The -INC tags are later used in the features in order to let the n-gram parser better predict such incomplete cases. Following these steps, the extraction process filters out the incomplete chains of unary rules that can be reached from the $ROOT$ constituent. The algorithm also keeps the parent information of the $ROOT$ constituent of the n-gram tree as an additional context information.

B. Characteristics of n-gram Trees

As we apply the extraction algorithm on the WSJOver15 portion of the Penn Treebank, we get hundreds of thousands of n-gram trees for each value of n in $\{3..9\}$. The analysis of these data sets reveals interesting points about the characteristics of such n-gram trees. Table I gives the percentages of the most common constituents in each n-gram training set along with the corresponding numbers obtained from the complete parse trees of the WSJ15 portion, which we use for training our full parser. The comparison indicates that the percentages of the noun (NP), verb (VP), and prepositional (PP) phrases in the n-gram trees are higher than the ones in the complete parse trees. On the other hand, the percentages of long-range constituents like S are lower for the n-gram trees, which is expected as the extraction process disfavors such constituents. Nonetheless, we see higher percentage in case of another long-range constituent SBAR, which exemplifies how the extraction process may still favor some long-range constituents. Based on this analysis, we may postulate that the increasing percentage of the NPs, VPs, and PPs per parse tree may help the n-gram parser do a better job in addition to the fact that they are smaller, thus less complex phrases.

TABLE I
PERCENTAGE OF THE MOST COMMON NON-TERMINAL CONSTITUENTS IN TRAINING SETS

Model	NP	VP	PP	ADJP	ADVP	S	SBAR	QP
3-gram	21.20	10.82	6.25	1.04	1.03	4.68	1.43	0.96
4-gram	20.69	10.87	6.44	1.05	1.07	4.93	1.64	0.85
5-gram	20.39	10.87	6.45	1.06	1.10	5.11	1.75	0.80
6-gram	20.11	10.81	6.49	1.04	1.10	5.21	1.84	0.79
7-gram	19.86	10.78	6.49	1.02	1.11	5.29	1.92	0.78
8-gram	19.68	10.74	6.50	1.01	1.11	5.35	1.99	0.77
9-gram	19.54	10.68	6.50	1.00	1.10	5.39	2.04	0.76
Full (WSJ15)	18.74	9.51	4.21	1.05	1.69	6.91	1.00	0.60

III. DISCRIMINATIVE CONSTITUENCY PARSING

In order to parse the n-grams and the complete sentences, we implemented a feature-rich discriminative constituency parser based on the work in [12]. It employs a discriminative model with the Conditional Random Field (CRF) based approach. Discriminative models for parsing maximize the conditional likelihood of the parse tree given the sentence. The conditional probability of the parse tree is calculated as in Equation 1, where $\mathcal{Z}_s$ is the global normalization function and $\phi(r|s;\theta)$ is the local clique potential:

$$P(t|s;\theta) = \frac{1}{\mathcal{Z}_s} \prod_{r \in t} \phi(r|s;\theta), \tag{1}$$

where

$$\mathcal{Z}_s = \sum_{t' \in \tau(s)} \prod_{r \in t'} \phi(r|s;\theta), \tag{2}$$

$$\phi(r|s;\theta) = exp \sum_i \theta_i f_i(r,s). \tag{3}$$

The probability of the parse tree t given the sentence s is the product of the local clique potentials for each one-level subtree r of a parse tree t which is normalized by the total local clique potential of all possible parse trees defined by $\tau(s)$. Note that the clique potentials are not probabilities. They are computed by taking the exponent of the summation of the parameter values θ for the features that are present for a given subtree r. The function $f_i(r,s)$ returns 1 or 0 depending on the presence or absence of feature i in r, respectively. Given a set of training examples, the goal is to choose the parameter values θ such that the conditional log likelihood of these examples, i.e., the objective function $\mathcal{L}$ given in Equation 5, is minimized:

$$\begin{aligned} \mathcal{L}(\mathcal{D};\theta) &= \sum_{(t,s) \in \mathcal{D}} \left(\sum_{r \in t} \langle f(r,s), \theta \rangle - logZ_{s,\theta} \right) \\ &- \sum_i \frac{\theta_i^2}{2\sigma^2}. \end{aligned} \tag{4}$$

When the partial derivative of our objective function with respect to the model parameters is taken, the resulting gradient in Equation 5 is basically the difference between the empirical counts and the model expectations, along with the derivative of the L_2 regularization term to prevent over-fitting. These partial derivatives which are calculated with the inside-outside algorithm by traversing all possible parse trees for a given sentence are then used to update the parameter values at each iteration. As in [12], we use stochastic gradient descent (SGD), which updates parameters with a batch of training instances instead of all in order to converge to the optimum parameter values faster:

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{(t,s) \in \mathcal{D}} \left(\sum_{r \in t} f_i(r,s) - E_\theta[f_i|s] \right) - \frac{\theta_i}{\sigma^2}. \tag{5}$$

We use the same feature templates of [12] and the same tool [24] to calculate the distributional similarity clusters which are used in the feature definitions. However, we use a different combination of corpora to calculate these clusters. We gathered an unlabelled data set of over 280 million words by combining Reuters RCV1 corpus [25], Penn treebank, and a large set of newswire articles downloaded over the Internet. Despite the difference, we tried to keep the size and the types of contents comparable to [12]. We use the default parameter settings for the tool provided by [24] and set the number of clusters to 200. In order to handle out-of-vocabulary (OOV) words better, we also introduce a new lexicon feature template $\langle$*prefix, suffix, base(tag)*$\rangle$, which makes use of the most common English prefixes and suffixes. A feature is created by putting together the prefix and suffix of a word, if available, along with the base tag of that word. If it does not have any prefixes or suffixes, NA is used, instead. As for n-gram parsing, we did not include or exclude any features. Like in [12], we also implemented chart caching and parallelization in order to save time.

IV. HIERARCHICAL JOINT LEARNING

In this section, we show how we jointly train the n-gram parser and the full parser. We use an instance of the multi-task learning setup called the Hierarchical Joint Learning (HJL) approach introduced in [11]. HJL enables multiple models to learn more about their tasks due to the commonality among the tasks. By using HJL, we expect the n-gram parser to help the full parser in cases where the n-gram parser is better.

As described in [11], the HJL setup connects all the base models with a top prior, which is set to zero-mean Gaussian in our experiments. The only requirement for HJL is that the base models need to have some common features in addition to the set of features specific to each task. As both parsers employ the same set of feature templates, they have common features

through which HJL can operate. All the shared parameters between base models are connected to each other through this prior. It keeps the values of the shared parameters from each base model close to each other by keeping them close to itself.

The parameter values for the shared features are updated by incorporating the top model feature into the parameter update function as in Equation 6. While the first term is calculated by using the update value from Equation 5, the second term ensures that the base model m is not getting apart from the top model by taking the difference between the top model and the corresponding shared parameter value. The variance σ_d^2 is a parameter to tune this relation:

$$\frac{\partial \mathcal{L}_{hier}(\mathcal{D};\theta)}{\partial \theta_{m,i}} = \frac{\partial \mathcal{L}_{hier}(\mathcal{D}_m,\theta_m)}{\partial \theta_{m,i}} - \frac{\theta_{m,i} - \theta_{*,i}}{\sigma_d^2}. \quad (6)$$

As shown in Equation 7, the updates for the top model parameter values are calculated by summing the parameter value differences divided by the base model variance σ_m^2, and then by subtracting the regularization term to prevent overfitting:

$$\frac{\partial \mathcal{L}_{hier}(\mathcal{D};\theta)}{\partial \theta_{*,i}} = \left(\sum_{m \epsilon \mathcal{M}} \frac{\theta_{*,i} - \theta_{m,i}}{\sigma_m^2} \right) - \frac{\theta_{*,i}}{\sigma_*^2}. \quad (7)$$

As in the case of the discriminative parser described in the previous section, SGD is used for faster parameter optimization. At each epoch of SGD, a batch of training instances is selected uniformly from each model in the setup. The number of training instances coming from each set, hence, depends on the relative sizes of the training sets.

V. Experimental Setup

A. Data

We evaluate our models by using the English Penn treebank corpus [13]. Like previous studies in this field, we use sections 02–21 for training, 22 for development, and 23 for testing. For complete sentence parsing, we use only the sentences that have no more than 15 words, that is WSJ15. To train the n-gram parsers, on the other hand, we use the rest of the Penn treebank, which we call WSJOver15. To test the n-gram parsers, we use the n-gram trees extracted from the development and the test sets of the WSJ15 in order to make the results more comparable with the full parser. By using our n-gram tree extraction algorithm, we extract n-gram trees for $n = [3, 9]$. Table II gives the number of parse trees in the training, development and test sets of each parser.

B. Evaluation

We use the *evalb* script[2] to get labelled precision, recall, and F_1 scores. These are calculated based on the number of nonterminals in the parser's output that match those in the standard/golden parse trees. We also report the percentage of completely correct trees, the average number of brackets crossing with the actual correct spans, and the percent of guessed trees that have no crossing brackets with respect to the corresponding gold tree. In order to better understand the n-gram parser and the jointly trained parser, we also evaluate how accurately these parsers handle different types of constituents.

[2]The *evalb* script is available on http://nlp.cs.nyu.edu/evalb.

TABLE II
Number of parse trees for each parser

Model	Training Set	Dev. Set	Test Set
3-gram	384,699	1,742	2,495
4-gram	318,819	1,341	1,916
5-gram	267,155	1,050	1,506
6-gram	227,505	807	1,158
7-gram	195,229	635	891
8-gram	168,075	486	667
9-gram	145,040	349	491
Full	9,753	421	603

VI. Experiments and Discussion

A. Baseline Parser

Our baseline parser is a discriminative constituency parser that runs on complete sentences. In order to make it run at its best, we set the learning factor η to 0.1 and the variance σ^2 to 0.1. We do 20 passes over the training set and use the batch size of 15 for the purpose of SGD. Table III shows the results we obtained with these settings on the development and test sets of WSJ15 portion of the Penn treebank. Our baseline parser achieves an F_1 score of 87.8% on the development set and 86.4% on the test set. Compared to the results obtained in [12], our implemented version's performance is a couple of points behind. The difference might be caused by small implementation details as well as by the different corpus that we used to calculate the distributional similarity clusters as discussed in Section IV.

TABLE III
Results on the Penn treebank

Dataset	Precision	Recall	F_1 score
Dev. Set	87.5	88.1	87.8
Test Set	86.4	86.4	86.4

B. N-gram Parser

Before training the full parser with the n-gram parser using HJL, we test the stand-alone n-gram parsers in order to understand where they are good at or where they fail, especially with respect to the full parser. We experimented with seven different n-gram sizes, i.e., with $n = [3, 9]$. Even though there are hundreds of thousands of training instances for each parser available from the WSJOver15 portion of the Penn treebank, we train our models with $20,000$ instances due to time and computational constraints. For a statistically more reliable evaluation, we report the averages of the scores obtained from five randomly selected versions of each training set.

TABLE IV
RESULTS OF THE n-GRAM PARSERS FOR THE DEVELOPMENT SET

Model	Precision	Recall	F_1 score	Exact	Avg CB	No CB	TagAcc
3-gram	86.37	86.60	86.49	73.13	0.02	98.15	86.80
4-gram	85.55	85.89	85.72	66.50	0.08	94.86	87.88
5-gram	86.91	86.29	86.60	61.68	0.10	93.02	89.95
6-gram	86.68	86.41	86.54	54.76	0.16	89.33	90.67
7-gram	87.49	87.00	87.24	51.29	0.21	86.52	92.17
8-gram	87.12	86.77	86.94	49.48	0.28	83.05	92.63
9-gram	87.69	86.96	87.33	46.68	0.35	79.41	92.91

TABLE V
F_1 SCORES ON THE MOST COMMON CONSTITUENTS FROM THE DEVELOPMENT SET

Model	NP	VP	PP	S	SBAR	ADVP	ADJP	QP
3-gram	89.26	89.86	92.04	84.69	56.31	73.19	46.50	75.27
4-gram	88.02	89.79	90.72	83.52	61.43	73.23	48.23	75.77
5-gram	88.35	90.42	89.81	85.77	72.39	76.31	52.22	72.96
6-gram	87.65	91.04	89.03	86.11	71.49	75.16	52.00	74.34
7-gram	87.80	91.73	88.40	87.39	77.91	76.87	50.09	84.45
8-gram	87.22	92.29	87.81	87.18	76.43	77.84	49.05	86.73
9-gram	87.79	91.76	87.33	87.46	76.37	72.21	53.66	86.43
Full	88.77	89.81	88.79	90.91	80.56	79.42	59.31	94.21

We use the same experimental setup of the baseline full parser. However, we optimize the parameters specifically for n-gram parsing. To make the n-gram parser run at its best, we set the learning factor η to 0.05 and the variance σ^2 to 0.1. Instead of doing 20 iterations like we did for the full parser, we observe that 10 iterations are enough. We choose a batch size of 30, instead of 15 for the SGD. Both decisions are related to the fact that the n-gram trees are relatively smaller compared to the complete parse trees. Thus, an n-gram parser requires a larger batch of training instances, but takes fewer iterations to get to its best performance.

Table IV shows the averaged F_1 scores obtained with all seven n-gram parsers on the development set. The comparison of our n-gram parsers with each other reveals a couple of interesting points. Firstly, using bigger n-gram trees in general leads to slightly higher F_1 scores, but the increase in precision is more apparent. Secondly, the fact that the 3-gram parser achieves an F_1 score of 86.5% by guessing 73.13% of the parse trees exactly, suggests that finding the exact n-gram tree is mostly an easy job, yet a small set of 3-gram trees contain most of the errors. This observation does not hold for larger n values, since the parsing task becomes more difficult for bigger trees.

In order to do further analysis, we investigate how accurately the n-gram parsers handle the different types of constituents. Table V shows the average F_1 scores of each n-gram model for the most common constituents. The first thing to notice is the degrading performance of handling noun (NP) and prepositional (PP) phrases, and the improving performance of handling verb phrases (VP) and declarative clauses (S) as n increases. When n increases, longer as well as more complex NPs and PPs are introduced. This results in degrading performance for such phrases.

On the other hand, as the sizes of the n-gram trees increase, it becomes easier to handle long-range constituents like VPs and Ss, since the parser sees more of them in the training set. The same argument holds for the remaining types of constituents in Table V. Another interesting point is the significantly lower accuracies of the n-gram parsers on QPs, especially with smaller n-gram trees.

TABLE VI
ACCURACY ON THE INCOMPLETE CONSTITUENTS IN THE DEVELOPMENT SET

Model	% of Incomplete Constituents in Golden Trees	Incomplete Constituent Accuracy	% of Unidentified Incomplete Const. w.r.t. All Unidentifieds
3-gram	22.0	86.65	26.2
4-gram	17.4	85.78	21.1
5-gram	14.5	84.20	16.7
6-gram	12.3	83.41	15.2
7-gram	10.8	83.32	13.9
8-gram	9.5	83.92	11.5
9-gram	8.7	84.94	10.4

Table VI shows the performances of the n-gram parsers on the incomplete constituents, as well as the percentages of constituents that are incomplete and the percentages of unidentified constituents from the golden trees that are incomplete. In most cases, as n increases, the accuracies on the incomplete constituents decreases. The contribution of the incomplete constituents to the number of unidentified constituents decreases as well.

However, this is more attributed to the fact that their percentage with respect to all constituents drops as n increases. Another point to notice is that despite its high performance, more than a quarter of the constituents unidentified by the 3-gram parser are incomplete. Considering that the 3-gram parser predicts 73.13% of the parse trees completely, it is highly likely that the performance of the 3-gram parser is affected by such constituents.

TABLE VII
AVERAGED F_1 SCORES OF THE BASELINE FULL PARSER (B) JOINTLY TRAINED WITH EACH n-GRAM MODEL

	Results for Dev. Set of the WSJ15				Results for Test Set of the WSJ15			
Model(s)	1K	2K	5K	10K	1K	2K	5K	10K
B+3-gram	87.60	87.69	87.97	87.91	86.37	86.07	86.23	86.21
B+4-gram	87.93	87.98	**87.99***	87.70	86.52	**86.42*****	86.44	86.54
B+5-gram	87.72	87.67	88.00	87.72	86.36	85.84	86.35	86.33
B+6-gram	87.88	87.73	**88.12****	87.66	86.55	85.95	86.24	86.31
B+7-gram	87.83	87.94	88.05	87.72	86.58	86.16	86.24	86.42
B+8-gram	87.93	87.91	87.96	87.78	**86.57****	86.45	86.16	86.43
B+9-gram	**88.19***	87.89	87.89	87.86	86.46	86.42	86.44	**86.59*****

TABLE VIII
F_1 SCORES OF THE JOINTLY TRAINED PARSER ON THE MOST COMMON CONSTITUENTS IN THE DEVELOPMENT SET

Model(s)	NP	VP	PP	S	SBAR	ADVP	ADJP	QP
B+3-gram	88.91	89.87	89.39	90.20	80.09	80.05	64.51	92.44
B+4-gram	88.82	90.30	89.43	90.26	81.00	79.51	61.65	92.14
B+5-gram	89.10	90.21	89.23	90.22	79.44	79.56	61.23	92.68
B+6-gram	89.19	90.15	89.30	90.26	80.55	79.75	63.18	93.07
B+7-gram	89.04	90.19	89.15	90.35	80.73	79.31	62.93	93.03
B+8-gram	88.96	90.17	88.90	90.27	80.91	79.26	61.11	92.48
B+9-gram	88.86	90.14	89.06	90.12	79.45	79.07	62.38	92.74
Baseline (B)	88.77	89.81	88.79	90.91	80.56	79.42	59.31	94.21

C. Jointly Trained Parser

In order to boost the accuracy of the full parser, we train it along with each n-gram parser. For the full and n-gram models, we use the previously used variance settings, that is 0.1. We set the top model variance σ_*^2 to 0.1 as well. We set the learning factors for the n-gram models and the top model to 0.1, whereas we use 0.05 for the full parsing model. With a lower learning rate, we make sure that the full parsing model starts to learn at a slower pace than usual so that it doesn't directly get into the effect of the accompanying n-gram model. As in the case of the baseline full parser, we do 20 passes over the training set and at each iteration, we update the parameters with a batch of 40 training instances gathered from all training sets in the setup.

In order to evaluate the effect of the training set size for each n-gram model, we use training sets of four different sizes for the n-gram parsers. We execute each experiment three times with randomly selected training sets. Table VII shows the averaged F_1 scores obtained by the jointly trained full parser on the development and test sets of the WSJ15. The rows indicate which models are trained together, whereas each column corresponds to a training set of different size for the n-gram model. In case of the full parser, we use the standard training set of the Penn treebank, which contains $9,753$ instances.

Scores in bold in Table VII indicate that the value is significantly[3] better than the baseline value according to the t-test. When we compare the results with the baseline F_1 score of 87.8% on the development set and 86.4% on the test set, we observe slight improvement at some of the configurations. In general, the jointly trained full parser outperforms the baseline parser when it is trained alongside an n-gram parser that uses a relatively smaller training set, like 5,000 instances for the development set and 1,000 instances for the test set. The best results though, are obtained by jointly training the baseline parser with the 9-gram parser. These results are statistically significantly better than the ones of the non-jointly trained full parser both for the development and test sets. In addition to these comparisons, we also observed that within 20 iterations, the jointly trained full parser reaches its best performance faster with respect to the baseline parser, which shows the push of the n-gram parser over the full parser.

We also analyze how accurately the jointly trained parser[4] handles different constituent types. Table VIII shows the averaged F_1 scores for the most common constituent types in the development set. The results indicate a couple of interesting reasons behind the slight improvement of the jointly trained full parser over the baseline. The first one is the slight improvement on NPs as the n-grams are getting bigger, which is especially visible with the best performing configuration among them, that is the one with the 6-gram model. PPs and VPs are also better processed with almost all jointly trained models. The biggest improvement is seen with the adjective phrases (ADJPs), especially when smaller n-grams are used. Even though the impact of ADJPs to the overall result is small compared to the other phrase types like NPs and PPs, this improvement is still worth mentioning. It is interesting to note that the same analysis on the stand-alone n-gram parsers reveals that they are not that good with ADJPs. Another thing to notice is the degrading performance over the QPs, as well as SBAR and S type constituents due to the fact that the n-gram parsers perform relatively worse on them (see Table V).

[3]The superscript * adjacent to the F_1 scores indicates that its significance is $p < 0.01$. In case of ** and ***, it is $p < 0.005$ and $p < 0.001$, respectively.

[4]Each accompanying n-gram parser in the HJL setup uses 5,000 training instances.

VII. Conclusion and Future Work

In this paper, we introduced n-gram parsing and analyzed how it is different than full sentence parsing. We observed that the bigger n-grams we use, the better accuracies we get, mostly due to increasing context information. We showed that the n-gram parsers are better than the full parser at parsing NPs, VPs, and PPs, but worse at parsing Ss and SBARs. After analyzing the stand-alone n-gram parsers, we used them for jointly training a full discriminative parser in the HJL setup in order to boost the accuracy of the full parser. We achieved statistically significant improvement over the baseline scores. The analysis of the results obtained with the jointly trained parser revealed that the resulting parser is better at processing NPs, VPs, PPs, and surprisingly ADJPs. However, it is negatively influenced by the performance of the n-gram parser over constituents like S and SBAR. Furthermore, it achieves its best performance faster than the baseline parser, indicating yet another benefit of training alongside an n-gram parser.

As future work, we plan to improve our baseline parser in order to make the jointly trained parser more competitive with respect to its peers in the literature. We will explore new approaches for selecting better n-gram trees to improve the quality of the training data. We also plan to use multiple n-gram parsers in joint training instead of just one. In addition, we will use the n-gram trees and the HJL setup to build a self-trained parser by expanding the n-gram parser's training data with n-gram trees extracted from the output of the full sentence parser. This will enable the full sentence parser to be indirectly trained with its own output.

Acknowledgments

We thank Brian Roark and Suzan Üskudarlı for their invaluable feedback. This work was supported by the Boğaziçi University Research Fund 12A01P6.

References

[1] T. Kasami, "An efficient recognition and syntax-analysis algorithm for context-free languages," *Technical report, Air Force Cambridge Research Lab*, 1965.

[2] J. Earley, "An effcient context-free parsing algorithm," *Communications of the ACM*, vol. 13(2), pp. 94–102, 1970.

[3] M. Collins, "Head-driven statistical models for natural language parsing," Ph.D. dissertation, Department of Computer and Information Science, University of Pennsylvania, 1999.

[4] E. Charniak, "Statistical parsing with a context-free grammar and word statistics," *Proceedings of AAAI-97*, pp. 598–603, 1997.

[5] A. Ratnaparkhi, "Learning to parse natural language with maximum entropy models," *Machine Learning*, vol. 34(1–3), pp. 151–175, 1999.

[6] E. Charniak, "A maximum-entropy-inspired parser," *Proceedings of the North American Association of Computational Linguistics*, 2000.

[7] M. Collins, "Discriminative reranking for natural language parsing," *Proceedings of ICML-2000*, pp. 175–182, 2000.

[8] L. Huang, "Forest reranking: Discriminative parsing with non-local features," *Proceedings of Ninth International Workshop on Parsing Technology*, pp. 53–64, 2005.

[9] D. McClosky, E. Charniak, and M. Johnson, "Effective self-training for parsing," *Proceedings of HLT-NAACL*, 2006.

[10] S. Abney, "Part-of-speech tagging and partial parsing," *Corpus-Based Methods in Language and Speech Processing, Kluwer Academic Publishers, Dordrecht*, 1999.

[11] J. R. Finkel and C. D. Manning, "Hierarchical joint learning: Improving joint parsing and named entity recognition with non-jointly labeled data," *Proceedings of ACL 2010*, 2010.

[12] J. R. Finkel, A. Kleeman, and C. D. Manning, "Efficient, feature-based conditional random field parsing," *Proceedings of ACL/HLT-2008*, 2008.

[13] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19(2), pp. 313–330, 1993.

[14] A. Ratnaparkhi, "A linear observed time statistical parser based on maximum entropy models," *Proceedings of EMNLP*, pp. 1–10, 1997.

[15] J. Henderson, "Discriminative training of a neural network statistical parser," *42nd ACL*, pp. 96–103, 2004.

[16] H. Zhang, M. Zhang, C. L. Tan, and H. Li, "K-best combination of syntactic parsers," *Proceedings of EMNLP 2009*, pp. 1552–1560, 2009.

[17] V. Fossum and K. Knight, "Combining constituent parsers," *Proceedings of NAACL 2009*, pp. 253–256, 2009.

[18] H. Daume III and D. Marcu, "Domain adaptation for statistical classifiers," *Journal of Artificial Intelligence Research*, 2006.

[19] J. R. Finkel and C. D. Manning, "Nested named entity recognition," *Proceedings of EMNLP 2009*, 2009.

[20] ——, "Joint parsing and named entity recognition," *Proceedings of the North American Association of Computational Linguistics*, 2009.

[21] R. Bod, R. Scha, and K. Sima'an, "Data oriented parsing," *CSLI Publications, Stanford University*, 2003.

[22] A. Joshi, L. Levy, and M. Takahashi, "Tree adjunct grammars," *Journal of Computer and System Sciences*, vol. 10:1, pp. 136–163, 1975.

[23] M. Bansal and D. Klein, "Web-scale features for full-scale parsing," *Proceedings of 49th Annual Meeting of ACL: HLT*, pp. 693–702, 2011.

[24] A. Clark, "Combining distributiona and morphological information for part of speech induction," *Proceedings of the tenth Annual Meeting of the European Association for Computational Linguistics (EACL)*, pp. 59–66, 2003.

[25] T. Rose, M. Stevenson, and M. Whitehead, "The Reuters corpus volume 1 - from yesterday's news to tomorrow's language resources," *Proceedings of the 3rd international conference on language resources and evaluation.*, 2002.