Online Learning for Inexact Hypergraph Search

Hao Zhang
Google
haozhang@google.com

Liang Huang **Kai Zhao**
City University of New York
{lhuang@cs.qc,kzhao@gc}.cuny.edu

Ryan McDonald
Google
ryanmcd@google.com

Abstract

Online learning algorithms like the perceptron are widely used for structured prediction tasks. For sequential search problems, like left-to-right tagging and parsing, beam search has been successfully combined with perceptron variants that accommodate search errors (Collins and Roark, 2004; Huang et al., 2012). However, perceptron training with inexact search is less studied for bottom-up parsing and, more generally, inference over hypergraphs. In this paper, we generalize the violation-fixing perceptron of Huang et al. (2012) to hypergraphs and apply it to the cube-pruning parser of Zhang and McDonald (2012). This results in the highest reported scores on WSJ evaluation set (UAS 93.50% and LAS 92.41% respectively) without the aid of additional resources.

1 Introduction

Structured prediction problems generally deal with exponentially many outputs, often making exact search infeasible. For sequential search problems, such as tagging and incremental parsing, beam search coupled with perceptron algorithms that account for potential search errors have been shown to be a powerful combination (Collins and Roark, 2004; Daumé and Marcu, 2005; Zhang and Clark, 2008; Huang et al., 2012). However, sequential search algorithms, and in particular left-to-right beam search (Collins and Roark, 2004; Zhang and Clark, 2008), squeeze inference into a very narrow space. To address this, Huang (2008) formulated constituency parsing as approximate bottom-up inference in order to compactly represent an exponential number of outputs while scoring features of arbitrary scope. This idea was adapted to graph-based dependency parsers by Zhang and McDonald (2012) and shown to outperform left-to-right beam search.

Both these examples, bottom-up approximate dependency and constituency parsing, can be viewed as specific instances of inexact hypergraph search. Typically, the approximation is accomplished by cube-pruning throughout the hypergraph (Chiang, 2007). Unfortunately, as the scope of features at each node increases, the inexactness of search and its negative impact on learning can potentially be exacerbated. Unlike sequential search, the impact on learning of approximate hypergraph search – as well as methods to mitigate any ill effects – has not been studied. Motivated by this, we develop online learning algorithms for inexact hypergraph search by generalizing the violation-fixing percepron of Huang et al. (2012). We empirically validate the benefit of this approach within the cube-pruning dependency parser of Zhang and McDonald (2012).

2 Structured Perceptron for Inexact Hypergraph Search

The structured perceptron algorithm (Collins, 2002) is a general learning algorithm. Given training instances $(x, \hat{y})$, the algorithm first solves the decoding problem $y' = \mathbf{argmax}_{y \in \mathcal{Y}(x)} \mathbf{w} \cdot \mathbf{f}(x, y)$ given the weight vector $\mathbf{w}$ for the high-dimensional feature representation $\mathbf{f}$ of the mapping (x, y), where y' is the prediction under the current model, $\hat{y}$ is the gold output and $\mathcal{Y}(x)$ is the space of all valid outputs for input x. The perceptron update rule is simply: $\mathbf{w}' = \mathbf{w} + \mathbf{f}(x, \hat{y}) - \mathbf{f}(x, y')$.

The convergence of original perceptron algorithm relies on the $\mathbf{argmax}$ function being exact so that the condition $\mathbf{w} \cdot \mathbf{f}(x, y') > \mathbf{w} \cdot \mathbf{f}(x, \hat{y})$ (modulo ties) always holds. This condition is called a *violation* because the prediction y' scores higher than the correct label $\hat{y}$. Each perceptron update moves weights

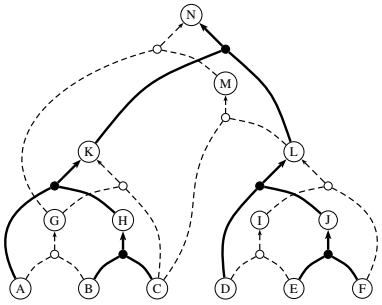

Figure 1: A hypergraph showing the union of the gold and Viterbi subtrees. The hyperedges in bold and dashed are from the gold and Viterbi trees, respectively.

away from y' and towards $\hat{y}$ to fix such violations. But when search is inexact, y' could be suboptimal so that sometimes $\mathbf{w} \cdot \mathbf{f}(x, y') < \mathbf{w} \cdot \mathbf{f}(x, \hat{y})$. Huang et al. (2012) named such instances *non-violations* and showed that perceptron model updates for non-violations nullify guarantees of convergence. To account for this, they generalized the original update rule to select an output y' within the pruned search space that scores higher than $\hat{y}$, but is not necessarily the highest among all possibilities, which represents a true violation of the model on that training instance. This *violation fixing perceptron* thus relaxes the $\mathbf{argmax}$ function to accommodate inexact search and becomes provably convergent as a result.

In the sequential cases where $\hat{y}$ has a linear structure such as tagging and incremental parsing, the violation fixing perceptron boils down to finding and updating along a certain prefix of $\hat{y}$. Collins and Roark (2004) locate the earliest position in a chain structure where $\hat{y}_{\text{pref}}$ is worse than y'_{pref} by a margin large enough to cause $\hat{y}$ to be dropped from the beam. Huang et al. (2012) locate the position where the violation is largest among all prefixes of $\hat{y}$, where size of a violation is defined as $\mathbf{w} \cdot \mathbf{f}(x, y'_{\text{pref}}) - \mathbf{w} \cdot \mathbf{f}(x, \hat{y}_{\text{pref}})$.

For hypergraphs, the notion of prefix must be generalized to subtrees. Figure 1 shows the packed-forest representation of the union of gold subtrees and highest-scoring (Viterbi) subtrees at every gold node for an input. At each gold node, there are two incoming hyperedges: one for the gold subtree and the other for the Viterbi subtree. After bottom-up parsing, we can compute the scores for the gold subtrees as well as extract the corresponding Viterbi subtrees by following backpointers. These Viterbi subtrees need not necessarily to belong to the full Viterbi path (i.e., the Viterbi tree rooted at node (N)). An update strategy must choose a subtree or a set of subtrees at gold nodes. This is to ensure that the model is updating its weights relative to the intersection of the search space and the gold path.

Our first update strategy is called *single-node max-violation (s-max)*. Given a gold tree $\hat{y}$, it traverses the gold tree and finds the node n on which the violation between the Viterbi subtree and the gold subtree is the largest over all gold nodes. The violation is guaranteed to be greater than or equal to zero because the lower bound for the max-violation on any hypergraph is 0 which happens at the leaf nodes. Then we choose the subtree pair $(\hat{y}_n, y'_n)$ and do the update similar to the prefix update for the sequential case. For example, in Figure 1, suppose the max-violation happens at node (K), which covers the left half of the input x, then the perceptron update would move parameters to the subtree represented by nodes (B), (C), (H) and (K) and away from (A), (B), (G) and (K).

Our second update strategy is called *parallel max-violation (p-max)*. It is based on the observation that violations on non-overlapping nodes can be fixed in parallel. We define a set of *frontiers* as a set of nodes that are non-overlapping and the union of which covers the entire input string x. The frontier set can include up to $|x|$ nodes, in the case where the frontier is equivalent to the set of leaves. We traverse $\hat{y}$ bottom-up to compute the set of frontiers such that each has the max-violation in the span it covers. Concretely, for each node n, the max-violation frontier set can be defined recursively,

$$\text{ft}(n) = \begin{cases} n, & \text{if } n = \text{maxv}(n) \\ \bigcup_{n_i \in \text{children}(n)} \text{ft}(n_i), & \text{otherwise} \end{cases}$$

where $\text{maxv}(n)$ is the function that returns the node with the absolute maximum violation in the subtree rooted at n and can easily be computed recursively over the hypergraph. To make a perceptron update, we generate the max-violation frontier set for the entire hypergraph and use it to choose subtree pairs $\bigcup_{n \in \text{ft}(\text{root}(x))} (\hat{y}_n, y'_n)$, where $\text{root}(x)$ is the root of the hypergraph for input x. For example, in Figure 1, if the union of (K) and (L) satisfies the definition of ft, then the perceptron update would move feature

weights away from the union of the two Viterbi subtrees and towards their gold counterparts.

In our experiments, we compare the performance of the two violation-fixing update strategies against two baselines. The first baseline is the *standard* update, where updates always happen at the root node of a gold tree, even if the Viterbi tree at the root node leads to a non-violation update. The second baseline is the *skip* update, which also always updates at the root nodes but skips any non-violations. This is the strategy used by Zhang and McDonald (2012).

3 Experiments

We ran a number of experiments on the cube-pruning dependency parser of Zhang and McDonald (2012), whose search space can be represented as a hypergraph in which the nodes are the complete and incomplete states and the hyperedges are the instantiations of the two parsing rules in the Eisner algorithm (Eisner, 1996).

The feature templates we used are a superset of Zhang and McDonald (2012). These features include first-, second-, and third-order features and their labeled counterparts, as well as valency features. In addition, we also included a feature template from Bohnet and Kuhn (2012). This template examines the leftmost child and the rightmost child of a modifier simultaneously. All other high-order features of Zhang and McDonald (2012) only look at arcs on the same side of their head. We trained the parser with hamming-loss-augmented MIRA (Crammer et al., 2006), following Martins et al. (2010). Based on results on the English validation data, in all the experiments, we trained MIRA with 8 epochs and used a beam of size 6 per node.

To speed up the parser, we used an unlabeled first-order model to prune unlikely dependency arcs at both training and testing time (Koo and Collins, 2010; Martins et al., 2013). We followed Rush and Petrov (2012) to train the first-order model to minimize filter loss with respect to max-marginal filtering. On the English validation corpus, the filtering model pruned 80% of arcs while keeping the oracle unlabeled attachment score above 99.50%. During training only, we insert the gold tree into the hypergraph if it was mistakenly pruned. This ensures that the gold nodes are always available, which is required for model updates.

3.1 English and Chinese Results

We report dependency parsing results on the Penn WSJ Treebank and the Chinese CTB-5 Treebank. Both treebanks are constituency treebanks. We generated two versions of dependency treebanks by applying commonly-used conversion procedures. For the first English version (PTB-YM), we used the Penn2Malt[1] software to apply the head rules of Yamada and Matsumoto and the Malt label set. For the second English version (PTB-S), we used the Stanford dependency framework (De Marneffe et al., 2006) by applying version 2.0.5 of the Stanford parser. We split the data in the standard way: sections 2-21 for training; section 22 for validation; and section 23 for evaluation. We utilized a linear chain CRF tagger which has an accuracy of 96.9% on the validation data and 97.3% on the evaluation data[2]. For Chinese, we use the Chinese Penn Treebank converted to dependencies and split into train/-validation/evaluation according to Zhang and Nivre (2011). We report both unlabeled attachment scores (UAS) and labeled attachment scores (LAS), ignoring punctuations (Buchholz and Marsi, 2006).

Table 1 displays the results. Our improved cube-pruned parser represents a significant improvement over the feature-rich transition-based parser of Zhang and Nivre (2011) with a large beam size. It also improves over the baseline cube-pruning parser without max-violation update strategies (Zhang and McDonald, 2012), showing the importance of update strategies in inexact hypergraph search. The UAS score on Penn-YM is slightly higher than the best result known in the literature which was reported by the fourth-order unlabeled dependency parser of Ma and Zhao (2012), although we did not utilize fourth-order features. The LAS score on Penn-YM is on par with the best reported by Bohnet and Kuhn (2012). On Penn-S, there are not many existing results to compare with, due to the tradition of reporting results on Penn-YM in the past. Nevertheless, our result is higher than the second best by a large margin. Our Chinese parsing scores are the highest reported results.

[1] http://stp.lingfil.uu.se//∼nivre/research/Penn2Malt.html

[2] The data was prepared by André F. T. Martins as was done in Martins et al. (2013).

	Parser	Penn-YM UAS	Penn-YM LAS	Penn-YM Toks/Sec	Penn-S UAS	Penn-S LAS	Penn-S Toks/Sec	CTB-5 UAS	CTB-5 LAS	CTB-5 Toks/Sec
	Zhang and Nivre (2011)	92.9-	91.8-	†680	-	-	-	86.0-	84.4-	-
	Zhang and Nivre (reimpl.) (beam=64)	93.00	91.98	800	92.96	90.74	500	85.93	84.42	700
	Zhang and Nivre (reimpl.) (beam=128)	92.94	91.91	400	93.11	90.84	250	86.05	84.50	360
	Koo and Collins (2010)	93.04	-	-	-	-	-	-	-	-
	Zhang and McDonald (2012)	93.06	91.86	220	-	-	-	86.87	85.19	-
	Rush and Petrov (2012)	-	-	-	92.7-	-	4460	-	-	-
	Martins et al. (2013)	93.07	-	740	92.82	-	600	-	-	-
	Qian and Liu (2013)	93.17	-	180	-	-	-	87.25	-	100
	Bohnet and Kuhn (2012)	93.39	92.38	†120	-	-	-	87.5-	85.9-	-
	Ma and Zhao (2012)	93.4-	-	-	-	-	-	87.4-	-	-
cube-pruning	w/ skip	93.21	92.07	300	92.92	90.35	200	86.95	85.23	200
	w/ s-max	**93.50**	**92.41**	300	93.59	91.17	200	87.78	86.13	200
	w/ p-max	93.44	92.33	300	**93.64**	**91.28**	200	**87.87**	**86.24**	200

Table 1: Parsing results on test sets of the Penn Treebank and CTB-5. UAS and LAS are measured on all tokens except punctuations. We also include the tokens per second numbers for different parsers whenever available, although the numbers from other papers were obtained on different machines. Speed numbers marked with † were converted from sentences per second.

The speed of our parser is around 200-300 tokens per second for English. This is faster than the parser of Bohnet and Kuhn (2012) which has roughly the same level of accuracy, but is slower than the parser of Martins et al. (2013) and Rush and Petrov (2012), both of which only do unlabeled dependency parsing and are less accurate. Given that predicting labels on arcs can slow down a parser by a constant factor proportional to the size of the label set, the speed of our parser is competitive. We also tried to prune away arc labels based on observed labels for each POS tag pair in the training data. By doing so, we could speed up our parser to 500-600 tokens per second with less than a 0.2% drop in both UAS and LAS.

3.2 Importance of Update Strategies

The lower portion of Table 1 compares cube-pruning parsing with different online update strategies in order to show the importance of choosing an update strategy that accommodates search errors. The max-violation update strategies (s-max and p-max) improved results on both versions of the Penn Treebank as well as the CTB-5 Chinese treebank. It made a larger difference on Penn-S relative to Penn-YM, improving as much as 0.93% in LAS against the skip update strategy. Additionally, we measured the percentage of non-violation updates at root nodes. In the last epoch of training, on Penn-YM, there was 24% non-violations if we used the skip update strategy; on Penn-S, there was 36% non-violations. The portion of non-violations indicates the inexactness of the underlying search. Search is harder on Penn-S due to the larger label set. Thus, as expected, max-violation update strategies improve most where the search is the hardest and least exact.

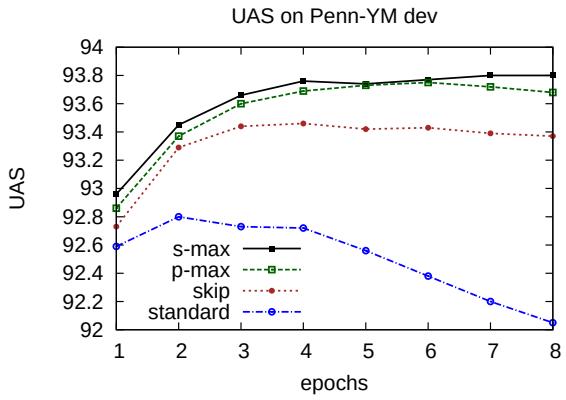

Figure 2: Constrast of different update strategies on the validation data set of Penn-YM. The x-axis is the number of training epochs. The y-axis is the UAS score. s-max stands for single-node max-violation. p-max stands for parallel max-violation.

Figure 2 shows accuracy per training epoch on the validation data. It can be seen that bad update strategies are not simply slow learners. More iterations of training cannot close the gap between strategies. Forcing invalid updates on non-violations (standard update) or simply ignoring them (skip update) produces less accurate models overall.

	ZN 2011 (reimpl.)		skip		s-max		p-max		Best Published†	
Language	*UAS*	*LAS*	*UAS*	*LAS*	*UAS*	*LAS*	*UAS*	*LAS*	*UAS*	*LAS*
SPANISH	86.76	83.81	87.34	84.15	**87.96**	**84.95**	87.68	84.75	*87.48*	*84.05*
CATALAN	94.00	88.65	94.54	89.14	94.58	89.05	**94.98**	**89.56**	*94.07*	*89.09*
JAPANESE	93.10	91.57	**93.40**	91.65	93.26	**91.67**	93.20	91.49	*93.72*	*91.7-*
BULGARIAN	93.08	89.23	93.52	89.25	**94.02**	**89.87**	93.80	89.65	*93.50*	*88.23*
ITALIAN	87.31	82.88	87.75	83.41	87.57	83.22	**87.79**	**83.59**	*87.47*	*83.50*
SWEDISH	90.98	**85.66**	90.64	83.89	**91.62**	85.08	91.62	85.00	*91.44*	*85.42*
ARABIC	78.26	67.09	80.42	69.46	80.48	69.68	**80.60**	**70.12**	*81.12*	*66.9-*
TURKISH	76.62	66.00	76.18	65.90	**76.94**	**66.80**	76.86	66.56	*77.55*	*65.7-*
DANISH	90.84	86.65	91.40	86.59	91.88	86.95	**92.00**	**87.07**	*91.86*	*84.8-*
PORTUGUESE	91.18	87.66	91.69	88.04	92.07	88.30	**92.19**	**88.40**	*93.03*	*87.70*
GREEK	85.63	78.41	86.37	78.29	86.14	78.20	**86.46**	**78.55**	*86.05*	*77.87*
SLOVENE	84.63	76.06	85.01	75.92	**86.01**	**77.14**	85.77	76.62	*86.95*	*73.4-*
CZECH	87.78	82.38	86.92	80.36	88.36	82.16	**88.48**	**82.38**	*90.32*	*80.2-*
BASQUE	**79.65**	71.03	79.57	71.43	79.59	71.52	79.61	**71.65**	*80.23*	*73.18*
HUNGARIAN	84.71	80.16	85.67	80.84	85.85	81.02	**86.49**	**81.67**	*86.81*	*81.86*
GERMAN	91.57	**89.48**	91.23	88.34	**92.03**	89.44	91.79	89.28	*92.41*	*88.42*
DUTCH	82.49	79.71	83.01	79.79	**83.57**	**80.29**	83.35	80.09	*86.19*	*79.2-*
AVG	86.98	81.55	87.33	81.56	87.76	82.08	87.80	82.14		

Table 2: Parsing Results for languages from CoNLL 2006/2007 shared tasks. When a language is in both years, we use the 2006 data set. The best results with † are the maximum in the following papers: Buchholz and Marsi (2006), Nivre et al. (2007), Zhang and McDonald (2012), Bohnet and Kuhn (2012), and Martins et al. (2013), For consistency, we scored the CoNLL 2007 best systems with the CoNLL 2006 evaluation script. ZN 2011 (reimpl.) is our reimplementation of Zhang and Nivre (2011), with a beam of 64. Results in bold are the best among ZN 2011 reimplementation and different update strategies from this paper.

3.3 CoNLL Results

We also report parsing results for 17 languages from the CoNLL 2006/2007 shared-task (Buchholz and Marsi, 2006; Nivre et al., 2007). The parser in our experiments can only produce projective dependency trees as it uses an Eisner algorithm backbone to generate the hypergraph (Eisner, 1996). So, at training time, we convert non-projective trees – of which there are many in the CoNLL data – to projective ones through flattening, i.e., attaching words to the lowest ancestor that results in projective trees. At testing time, our parser can only predict projective trees, though we evaluate on the true non-projective trees.

Table 2 shows the full results. We sort the languages according to the percentage of non-projective trees in increasing order. The Spanish treebank is 98% projective, while the Dutch treebank is only 64% projective. With respect to the Zhang and Nivre (2011) baseline, we improved UAS in 16 languages and LAS in 15 languages. The improvements are stronger for the projective languages in the top rows. We achieved the best published UAS results for 7 languages: Spanish, Catalan, Bulgarain, Italian, Swedish, Danish, and Greek. As these languages are typically from the more projective data sets, we speculate that extending the parser used in this study to handle non-projectivity will lead to state-of-the-art models for the majority of languages.

4 Conclusions

We proposed perceptron update strategies for inexact hypergraph search and experimented with a cube-pruning dependency parser. Both single-node max-violation and parallel max-violation update strategies signficantly improved parsing results over the strategy that ignores any invalid udpates caused by inexactness of search. The update strategies are applicable to any bottom-up parsing problems such as constituent parsing (Huang, 2008) and syntax-based machine translation with online learning (Chiang et al., 2008).

Acknowledgments: We thank André F. T. Martins for the dependency converted Penn Treebank with automatic POS tags from his experiments; the reviewers for their useful suggestions; the NLP team at Google for numerous discussions and comments; Liang Huang and Kai Zhao are supported in part by DARPA FA8750-13-2-0041 (DEFT), PSC-CUNY, and a Google Faculty Research Award.

References

B. Bohnet and J. Kuhn. 2012. The best of bothworlds - a graph-based completion model for transition-based parsers. In *Proc. of EACL*.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of CoNLL*.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proc. of EMNLP*.

D. Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2).

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. of ACL*.

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of ACL*.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

H. Daumé and D. Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proc. of ICML*.

M. De Marneffe, B. MacCartney, and C.D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. of LREC*.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: an exploration. In *Proc. of COLING*.

L. Huang, S. Fayong, and G. Yang. 2012. Structured perceptron with inexact search. In *Proc. of NAACL*.

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL*.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proc. of ACL*.

X. Ma and H. Zhao. 2012. Fourth-order dependency parsing. In *Proc. of COLING*.

A. F. T. Martins, N. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proc. of EMNLP*.

A. F. T. Martins, M. B. Almeida, and N. A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proc. of ACL*.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. of EMNLP-CoNLL*.

X. Qian and Y. Liu. 2013. Branch and bound algorithm for dependency parsing with non-local features. *TACL*, Vol 1.

A. Rush and S. Petrov. 2012. Efficient multi-pass dependency pruning with vine parsing. In *Proc. of NAACL*.

Y. Zhang and S. Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proc. of EMNLP*.

H. Zhang and R. McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proc. of EMNLP*.

Y. Zhang and J. Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of ACL-HLT*, volume 2.