Dependency Parsing

Johnny Tan, Henry Chen
Emory University, Atlanta, Georgia
{johnny.tan,henry.chen}@emory.edu

Abstract

Greedy search transition-based parsers are only able to explore a portion of the whole search space.To tackle this restriction beam search is used at decoding time instead. In this paper we first analyze the performance of the greedy search projective parsing algorithm against our beam search parser. Then we evaluate the use of word embeddings, rich non-local features [Zhang and Nivre, 2011], various learning techniques, and hyper-parameter tuning to improve parser accuracy.

1 Introduction

Transition-based parsing have been shown to achieve state of the art performance in accuracy and speed from a breadth of features and linear time complexity. The standard transition-based approach follows shift-reduce parsing and uses a component called the oracle to predict an optimal sequence. Many shift-reduce systems such as the arc-eager or arc-standard use a static oracle to produce a single sequence of transitions that is followed in its entirety. Because of its greedy nature, such systems fail to address ambiguity and do not consider alternative paths to reaching the gold tree. As a result the static oracle is infeasible to be used in different domains.

In recent years, the improvements of graph-based parsing have demonstrated to beat greedy transition parsing in accuracy by as much as 2%. However as a trade-off for accuracy, graph parsing tends to be up to 10 times slower Choi et al. [2015]. The standard graph-based approach attempts to find the most likely tree from all possible trees using maximum spanning tree algorithms. This exhaustive search guarantees to find the best tree and addresses ambiguity cases at the cost of speed. In attempts to balance accuracy and speed, beam search, a heuristic search algorithm is used for both graph and transition based parsing. The approximation heuristic and update methodology has proven to yield performance gains in parsing tasks Collins and Roark [2004], and can achieve accuracy close to inference. This paper seeks to:

- Evaluate the performance of our beam parser relative to the greedy arc eager parser
- Explore the scoring for k-best trees and our ”late update” to the model
- Evaluate the effectiveness of rich non-local features, parameter tuning, and word clusters on parsing

2 Related Work

Beam search combined with incremental left to right parsing results in competitive performance to the generalized model on parsing the Penn Treebank Collins and Roark [2004] . By using an ”early update” strategy the model is updated whenever the gold action falls off the beam and the rest of the sequences are neglected. The intuition behind this is that the later sequences are directly influenced by the previous sequences and should be neglected once the sequence is wrong. In contrast, the use of a ”late update” strategy still yields improvements despite under performing the ”early update” Huang et al. [2012]. Other related work include finding the k-best tree Huang and Chiang [2005].

Existing parsing systems all employ a similar restricted set of features. Compared to graph-based systems, transition-based systems can easily incorporate arbitrarily complex non-local features. The introduction of even richer feature representations outweighs the effect of combining graph-based models with transition-based models in performance Zhang and Nivre [2011].

3 Background and Approach

3.1 Adaptive Gradient Descent Learning

Our system uses a stochastic adaptive subgradient algorithm called Adagrad to exploit rarely seen features while remaining scalable Duchi et al. [2011]. We train Adagrad in on-line fashion and mini-batches with DAgger aggregation. In addition we use Adadelta which learns per dimension based on first order information over time and results in better computational overhead than Adagrad .

3.2 Greedy Transition-Based Parsing

In the arc-eager system Nivre [2003], a given configuration consist of a stack, an input queue for words, and an output list to hold the arcs. A set of operations are defined to operate on our input words to build the parsed tree until termination. They are defined as the following:

- Shift: Removes an input word from the queue and pushes it to the stack
- Reduce: Pops an item off the stack
- LeftArc: Pops an item off the stack, adds an arc to in relation of input word from queue
- RightArc: Removes an input word from queue, adds an arc to in relation to stack, pushes the input word to the stack

3.3 Beam Search

Beam search has been widely used in decoding for both graph-based and transition-based systems. At each state instead of taking a single best optimal prediction, a beam sized number of predictions based on a heuristic is taken into consideration. By increasing the number of possible transitions to take, the parser has a higher error tolerance in terms of prediction selection, and has the ability to select better scoring trees even if the parser might has made mistakes in the earlier states.

Algorithm 1 Beam Search

Require: $k \geq 0$ (k is beam width)
  function $BeamSearch(x, w, k)$
  $B_0 \leftarrow [\epsilon]$
  for $i \in 1...|X|$ **do**
    $B_i \leftarrow BEST_k(x, B_{i-1}, w)$
  end for
  return $B_{|x|}[0]$ (ret best sequence)

w - model, $BEST_k$ - returns top K extensions

The code above describes how given the current beam we try to find the top K extensions possible at each state and then return the best sequence at the end. Our system follows a "late update" model where we update our model after we have reach the end and have calculated all beams. It has two heuristics in choosing the best tree. It considers both the highest scoring tree and the closest tree to the gold regardless of score. The scores of each prediction is then normalized using softmax.

3.4 Rich Non-Local Features

We compare the accuracy of using all the rich non-local features for transition based parsing suggested by Zhang and Nivre [2011] against our own non local feature template. Table 1 shows the following non-local features we used. We choose to use valency features of all directions instead of specifying left or right. S denotes the stack and N denotes the input queue. lmd represents left most dependent and lns is left nearest sibling followed by the order. Direction is also denoted where a represents all directions, l as left, or r as right.

Valency	Third Order
$S_{0va}S_0w$	$S_{0rmd2}pS_0pN_0p$
$N_{0va}N_0w$	$S_{0lmd2}pS_{0lmd}pS_0p$
-	$N_{0lmd2}pS_{0p}N_0p$
-	$S_{0lns2}dS_{0p}N_0p$
-	$N_{0lmd2}dS_{0p}N_{0p}$
-	$S_{0lmd2d}S_{0p}N0p$
-	$S_{0rmd2}p$
-	$S_{0h2}p$
-	$S_{0rmd2}w$
-	$S_{0h2}w$
-	$S_{0lmd2}d$

Table 1: p - pos, w - word lemma, v - valency, d - dependency label, # - offset

3.5 Word Clusters

Formed from embeddings generated by neural nets, word clusters provide valuable information to improve NLP tasks such as named entity recognition and part of speech tagging Turian et al. [2010], Zhai et al. [2015]. In our experiments we use word clusters created from agglomeratative clustering and Brown clustering on the New York Times corpus. The structure of these word clusters is a binary tree where each word is mapped to a bit string that represents the path to it from the root node. Brown

cluster is an unsupervised learning algorithm that creates "class n-gram" models. The agglomerative clustering that we use is called Greedy Lazy Agglomerative Clustering (Greedo) Stratos et al.. These clusters are formed similarly to Brown by agglomeratively clustering dimensionally reduced word co-occurence matrixes using Ward's algorithm.

3.6 Setup and Evaluation

The English portion of the Penn Treebank is used for experiments following the standard split 02-21 for train, 22 for development, and 23 for test. We run our experiments on both the Standford format and the Malt. For beam search we experiment with beam sizes of 2, 4, 8, 16. In addition to using rich non-local features and word clusters we also tune our hyper parameters for optimal performance. The evaluation score is split between UAS, the number of correct heads, and LAS the number of correct labels and heads.

4 Experiments

4.1 Greedy Search

Greedy Search Result				
—	Malt		Stanford	
Trainer	LAS	UAS	LAS	UAS
Adagrad	88.01	88.87	87.97	89.73
Adagrad Mini Batch	86.62	87.49	87.35	87.49
Adadelta	87.97	88.86	87.71	88.54

4.2 Beam Search

Beam Search Result (adjusted)					
—		Malt		Stanford	
Trainer	Beam size	LAS	UAS	LAS	UAS
Adagrad	Beam 2	87.14	88.13	85.04	86.88
Adagrad	Beam 4	88.22	89.14	83.71	85.52
Adagrad	Beam 8	89.13	89.89	82.16	84.09
Adagrad	Beam 16	89.46	90.23	81.3	82.6

4.3 Rich Non-Local Feature

Zhang and Nivre's Features				
—	Malt		Stanford	
Algorithm	LAS	UAS	LAS	UAS
Adagrad	86.01	86.86	85.81	87.77
Adagrad Mini Batch	83.61	84.51	83.78	85.50
Adadelta	85.27	86.18	86.15	88.00

4.4 Word Clusters

Word Clusters					
—		Malt		Stanford	
Algorithm	Cluster	LAS	UAS	LAS	UAS
Adadelta	-	87.69	88.55	87.57	89.84
Adamini	-	86.16	86.37	87.28	89.04
Adagrad	-	87.90	88.75	87.82	89.58
Adagrad	Brown	88.01	88.87	87.97	89.73
Adagrad	Agglomerative	87.62	88.47	87.48	89.25

5 Results

5.1 Note on Beam Search Results

From our experiments we find that our beam search parser is not performing well on the development set compared to greedy. We begin to question our scoring mechanism and the "late update" that is being applied to our model. We notice that our development score is decreasing as we increase the number of beam sizes. The question of whether the noise in data was affecting our scores dramatically or if we were just picking the wrong tree at decode came to mind. Instead of continuing on with evaluation on the test set, we decide to test our hypothesis that we were just picking the wrong tree at decode time. The LAS and UAS evaluation methodology is modify to check for this. In addition to just counting correct labels or heads, the beam search evaluation now consists of checking if we chose the right beam and that we just didn't choose the gold tree in the last beam.

From this new evaluation method we notice that our scores are now increasing as the beam size increases. This does tells us that we are definitely predicting the right beam with the gold tree in it. It tells us that our scoring mechanism or "late updating" is hurting our model. We attempt to normalize our score using softmax to reduce unfavorable bias. We try feature reduction to reduce noisy information. However we begin to realize that the scoring mechanism and how the model is updated is the bigger factor our parser's erformance. We notice that not only were we scoring the wrong operations during decode, but we should neglect states that were already wrong previously. Hence we realize why "late updating" under performs and "early" updating" is a better option.

5.2 Beam Search versus Greedy

Our modified evaluation on beam search gives us a better understanding on the accuracy of beam search if the scoring mechanism was correct. Relative to the results to greedy, beam search would have only outperform at beam size 16. Taking run

time and the accuracy gains, and the results of our greedy search with non-local features into account, Greedy arc-eager would be a better parser than our beam parser.

5.3 Non-Local Feature

From the results of using rich non-local features suggested by Zhang and Nivre [2011] we did not find any significant improvement in our transition-based parser. We instead suggest our set of non-local features which include less valency and less third order features. We suggest using valency in all directions instead of left or right and using less combinations of higher order features. This gives us a more reduced feature space and better accuracy.

5.4 Word Clusters

In the case of dependency parsing, Brown clusters have the most improvements compared to agglomerative (.3% to .4% more). And compared to baseline with out the use of this information, word clusters create a least a .2% boost in accuracy.

5.5 Additional Findings

Through hyper parameter tuning we found that Adagrad works better with a lower feature cutoff (feature cutoff = 2) and that Adadelta performed just as well as Adagrad with a batch ratio of .1.

6 Conclusion

We analyze our beam search parser with a "late update" strategy relative to the greedy parser and found that "late update" is not necessarily the best strategy. Although the development scores of our parser did not perform as expected, from the adjusted evaluation scores, we were able to conclude that our parser is picking the right beam with an incorrect scoring mechanism. In addition we confirm that the use of word clusters yield significant improvements for dependency parsing. We suggested less non-local features than Zhang and Nivre [2011] and show that it yields improvement. We conclude that greedy parsing is a better choice than beam.

Acknowledgments

We acknowledge Dr. Jinho D. Choi for providing valuable training, development, and test data for our experiments in this paper. Also, we acknowledge the fact that our parser is built on top of the infrastructure of EmoryNLP, founded and maintained by Dr. Jinho D. Choi at Emory University.

References

Jinho Choi, Tetreault Joel, and Stent Amanda. It depends: Dependency parser comparison using a web-based evaluation tool. In *Proceedings of the 56th Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2015.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2004.

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

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

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

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer, 2003.

Karl Stratos, Do-kyum Kim, Michael Collins, and Daniel Hsu. A spectral algorithm for learning class-based n-gram models of natural language.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

Mike Zhai, Johnny Tan, and Jinho Choi. Intrinsic and extrinsic evaluations of word embeddings. 2015.

Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193. Association for Computational Linguistics, 2011.