CS395T Project 2: Shift-Reduce Parsing

Abstract

In this project, we explore transition based models for unlabeled dependency parsing. We first implement an arc standard transition system with a greedy shift-reduce parser. The parser greedily chooses the best decision from available legal decisions at every state. Next, we implement a global parser using beam search for the arc standard system. Finally, we extend our analysis to other transition systems, such as Nivre's arc eager and arc hybrid transition systems.

1 Introduction

A dependency tree is a syntactic tree-structured representation of a sentence. It specifies relations between words in a sentence, denoted by labeled directed arcs. Dependency parsing is the task of inferring these relations between words of a given sentence.

In this project, we focus on identifying the unlabeled dependencies between words. More specifically, for each word in a given sentence, we identify its *head* word, without worrying about the corresponding label. We evaluate the algorithms by measuring the number of unlabeled dependencies identified correctly, as a ratio of the total dependencies in the test set.

1.1 Arc Standard Transition System

The algorithm maintains a configuration of stack, input buffer, and dependencies. They are initialized with the ROOT element, word tokens and an empty list respectively. At each step, the algorithm takes one of the following three actions: Left-arc, Right-arc, or Shift. The former two operations assert a dependency relation between the top elements of the stack, whereas the Shift operation pushes the top element of the buffer into the stack. In order to train the dependency model, we first generate (state-decision) pairs that specify the gold decisions of the arc-standard oracle for the given observed state. A multi-class classifier is then trained over this data. In this project, we implement the greedy and beam search based approaches for training and parsing.

2 Implementation Details

2.1 Part 1: Greedy Shift-Reduce Parsing

The arc standard oracle is used to generate gold (parser state, decision) pairs. We build a multi-class classifier that looks at the current state and predicts the appropriate decision for that state. In the greedy shift-reduce parsing, the system only encounters gold parser states during training.

Decode: For a given test sentence, the parser iteratively evaluates parser states, and takes the greedy action as determined by the multi class classifier. We perform additional checks to ensure that the system does not take illegal actions/decisions even if it the highest scored action.

Figure 1 gives an outline of the UAS score achieved by the greedy shift reduce parser. We compare the performance of logistic and averaged perceptron; both were trained using stochastic gradient descent. We achieve a **UAS score of 79.98** using averaged perceptron, trained over 5 epochs.

2.2 Part 2: Global Beam Search

One of the shortcomings of the greedy shift reduce parser is that it offers no lookahead to the system. At each step, the optimal transition is decided based entirely on the current configuration, without considering past or future configurations that would result due to the current decision. We implement a global beam search parser that addresses this shortcoming.

In global beam search, instead of selecting decisions with the highest score for each state, we ac-

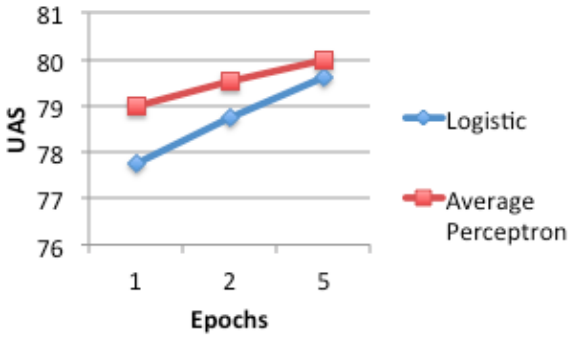

Figure 1: UAS scores for the greedy shift reduce parsers. We compare the performance of multi-class logistic and averaged perceptron, across different epochs. The learning rate for logistic regression is set to 0.01 (determined empirically).

cumulate scores over successive states, and select the state/decision sequence that leads to the highest score. For each sentence, we initialize a beam of size k, containing only the initial state configuration. At every step m, we determine all possible legal successors for the states in the beam at step $m-1$, and retain the top k choices. We follow this process iteratively, till we reach the final state. The states/decision sequences that correspond to the head of the beam is outputted as the predicted dependency parse.

To train the model, the gold decisions and state sequences is determined using the arc standard oracle, similar to the greedy approach. In this implementation, I used the 'early update' strategy; the weight vectors were updated using stochastic gradient descent, as soon as the gold decision (and state) falls off the beam.

Table 1 outlines the results of global beam search for the arc standard transition system. We compare the performance of structured SVM (with loss augmentation, learning rate = 0.1) and averaged perceptron. The results confirm our hypothesis that increasing the beam size leads to higher accuracy scores. However, it also increases the computation time significantly. We are able to achieve a maximum **UAS of 80.45** using averaged perceptron with beam size 7, trained over 5 epochs.

2.3 Part 3: Extension

As part of the extension, I implemented two additional transition based dependency parsers, namely Arc-Eager and Arc-Hybrid systems.

Table 1: UAS scores for global beam search for the arc standard model. The best result is obtained by using averaged perceptron with beam size = 7, trained over 5 epochs.

		beam size (k)		
	epochs	k = 3	k = 5	k = 7
Structured SVM	1	76.49	77.06	76.68
	3	77.73	77.8	79.08
	5	78.61	79.37	79.05
Averaged perceptron	1	78.56	78.98	79.26
	3	79.38	80.22	80.3
	5	80.14	80.19	**80.45**

2.3.1 Arc Eager Transition System

In the arc standard transition system, the oracle waits for a particular word to obtain all its child dependencies, before assigning it a parent word through right arc. This is necessary as the transitions in the arc standard system remove the dependent word from the stack as soon as it is attached to a parent. On the contrary, the arc eager transition system adds a right dependency arc as soon as possible, without removing the dependent word from consideration in future states. Instead, it introduces a separate reduce operator, that pops the topmost element from the stack.

Figure 2 outlines the transitions used by the arc eager oracle to obtain gold decisions. This transition strategy is sound and complete for projective trees. However, for non-projective trees, there may be states where it is not possible to apply any of these operators, while restricting to gold dependencies. In such scenarios, we enforce an *incorrect* Reduce operation. Before performing the reduce operation, we check if the element to be removed has a parent dependency. If such a dependency does not exist, we assign a dependency arc from its left neighbor to the given word.

LEFT-ARC$_l$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma,j\|\beta,A\cup\{(j,l,i)\})$
RIGHT-ARC$_l$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma\|i\|j,\beta,A\cup\{(i,l,j)\})$
REDUCE	$(\sigma\|i,\beta,A) \Rightarrow (\sigma,\beta,A)$
SHIFT	$(\sigma,i\|\beta,A) \Rightarrow (\sigma\|i,\beta,A)$

Figure 2: Transitions for the arc eager transition system. This figure is borrowed from (Goldberg and Nivre, 2012)

2.3.2 Arc Hybrid Transition System

The arc-hybrid transition system combines the left-arc operator of the arc eager system, and the right-arc operator of the arc standard system. This effectively removes the need for an extra reduce operator. Figure 3 gives an overview of the transitions in the arc hybrid system.

$$\begin{array}{ll} \textbf{Shift} & (\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A) \\ \textbf{LArc} & (\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j \rightarrow i)\}) \\ \textbf{RArc} & (\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i \rightarrow j)\}) \end{array}$$

Figure 3: Transitions for the arc hybrid transition system. This figure is borrowed from (Qi and Manning, 2017)

Similar to arc eager, the transition system is sound and complete for projective trees. For non-projective trees, we forcibly enforce a right arc whenever the oracle is unable to add a gold dependency,

2.3.3 Training and Evaluation

Similar to arc standard, we train the arc eager and arc hybrid transition system over the gold state-decision pairs. We train an averaged perceptron model, with greedy parsing. Figure 4 shows the performance of the different transition system on the dev set, across different epochs. As we can see, both the arc eager and arc hybrid transition systems show significant improvement over the arc standard model. This suggests that top-down approach to building a dependency tree works better than bottom up approaches (such as arc-standard). One of the reasons for this could be that in the arc standard system, the algorithm has to perform an additional implicit prediction of whether a given word has already obtained all its child dependencies. An early addition of such an incorrect dependency would deprive all its child words of the right dependency arc. The arc-eager transition system addresses this shortcoming by adding dependency arcs as soon as it encounters them. In our experiments, the arc-hybrid transition system performs the best, achieving a **UAS score of 82.43** over 5 epochs. Results over the blind test set is generated using this model.

An interesting observation from our explorations was that high accuracy values could be achieved with limited number of training samples.

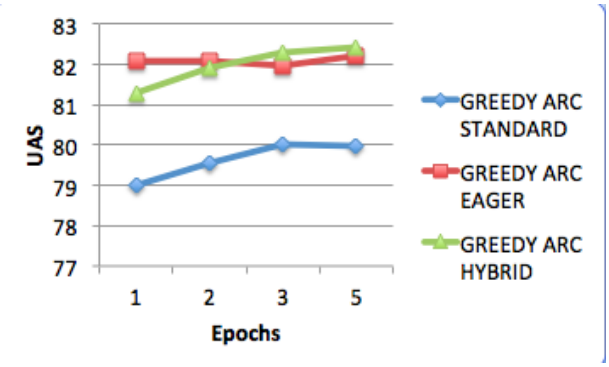

Figure 4: Performance of the greedy shift reduce parser for different transition systems.

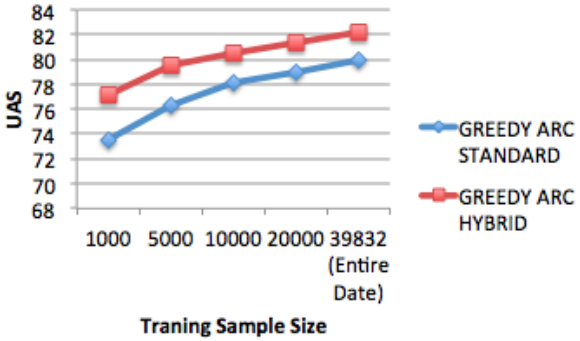

Figure 5: Performance of the Arc Standard and Arc Hybrid systems when trained over different sample sizes.

Figure 5 shows the accuracies of the greedy arc standard and arc hybrid system when trained with different number of training examples. As can be seen, the accuracy gain is very little after training over 10000 samples, even when the training sample size is doubled.

Another observation from Figure 4 and 5 is that the accuracy gain achieved by arc hybrid transition system over the arc standard transition system remains nearly constant across epochs and training sample sizes, between $2.8 - 3.2$. This might suggest that the accuracy gain is entirely due to the difference in the transitions defined, and not because of random parameter initializations. However, it would be helpful to get further insight into this.

References

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing.

Peng Qi and Christopher D Manning. 2017. Arc-swift: A novel transition system for dependency parsing. *arXiv preprint arXiv:1705.04434*.