Assignment Report:

Semantic Dependency Parsing

Name: 汤济之
ID: 1300012124
E-mail: tjz427@sina.cn
School of Electronics Engineering and Computer Science, Peking University

Abstract

The second assignment that I choose is ***Semantic Dependency Parsing***. I trained an arc-eager transition based parser using a basic perceptron model. The parser achieves an F1-measure of over 70% in labeled data, and over 75% in unlabeled data. I also tried kernel method and a state-of-art method called stack-LSTM. But all failed because of the training speed.

1. Assignment Introduction

Semantic Dependency Parsing (SDP) is a central task in natural language processing. It aims to recover sentence-internal predicate-argument relationships for all content words, i.e. the semantic structure constituting the relational core of sentence meaning.

However, there is no standard representations for semantic dependencies. In English, 3 types of representations are broadly used: **DM**, **PAS**, and **PCEDT**. They have distinct annotations, showing different understanding towards the meaning of a sentence. The detailed difference can be found in this paper listed below[1].

Graph-based and transition-based approaches are two most-used methods to semantic dependency parsing. They adopt very different views of the problem, each view having its strengths and limitations. Graph-based models often struggle to find the global optimal, while transition-based models are fast and greedy, working word by word, step by step.

I implemented a transition-based parser with basic perceptron in this assignment. I also did some further exploration——to implement a state-of-art model called stack-LSTM parser, but somehow cannot training properly on my computer.

In section 2, I will introduce the training data and the evaluation approach. In section 3, I will describe my model thoroughly. The result will be shown in section 4. And some further discussion and my failed model will be talked in section 5.

2. Data and Evaluation

Training data and test data are given.

Training data consists 3259 pre-annotated English sentence, in DM representation. The data format is shown in Table 1. Sentences are separated by an empty line. And each sentence starts from a

[1] Oepen S, Kuhlmann M, Miyao Y, et al. SemEval 2014 Task 8: Broad-coverage semantic dependency parsing[C]//Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014). 2014: 63-72.

id	form	lemma	pos	top	pred	arg_1	arg_2
#20200002							
1	Ms.	Ms.	NNP	-	+	_	_
2	Haag	Haag	NNP	-	-	compound	ARG1
3	plays	play	VBZ	+	+	_	_
4	Elianti	Elianti	NNP	-	-	_	ARG2
5	.	.	.	-	-	_	_

Table 1. data format

number with "#", indicates the id of the sentence. Below id, each line describes a word. Column 2 id the original form of a word, while column 3 represents the corresponding lemma and column 4 represents the POS-tag. "top" and "pred" are marked by "+" or "-", where "+" means yes "-" means no. "top" tells whether this word is pointed by the root of the graph. And "pred" indicate whether the word is a predicate, i.e. whether the word is a node that has outgoing dependency edges in the dependency graph. Columns after "pred" indicate the relations between arguments and predicate. For example, in table 1 "compound" means the word "Haag" is a "compound" of the first predicate.

In the test data, the format is the same. The only different is that in test data only column "id", "form", "lemma", and "pos" are given.

As for evaluation, a small tool is provided. The key measures are *labeled* and *unlabeled precision* and *recall* with respect to predicted dependencies and *labeled* and *unlabeled exact match* with respect to complete semantic dependency graphs.

3. Method

3.1 Transition-based Model

Transition based model is a clever transform of this semantic dependency parsing problem. It reduces the complex graph searching problem to a simple classification problem. in this approach, we are not going to predict the exact optimum in the whole graph, but to simulate the process of drawing a dependency graph step by step, from one state to another state. And the transition process between states is a classification problem. by do this greedily, the dependency graph can be constructed.

The key component in transition-based model is a set of *configurations*. A configuration contains a *stack* to store partially parsed dependency graphs, and a *buffer* to store remaining words. I designed stack based structure to store operation history and graph structure to save partially pared dependency graph also. In each transition step, we can select features from current configuration, and predict the operation.

Commonly, there 2 sets of operations are widely used in transition-based model. One is *arc-standard*, which contains 3 kinds of operations: *Shift*, to move one word from buffer to stack; *Left-Arc*, to generate a left-oriented edge in the dependency graph between the word at the top of the stack and the top of the buffer, remove the word at the top of the stack and move the word at the top of the buffer to that place; *Right-Arc* does the reverse as Left-Arc. Another algorithm is arc-eager, which separate edge generating and word removing as two independent operations. Thus, in contrast to arc-standard, arc-eager algorithm has one more operation called *Reduce*, which means remove one word at the top of the stack.

I use the transition-based model with arc-eager algorithm to implement the parser.

3.2 Why Arc-eager

According to the description above, we

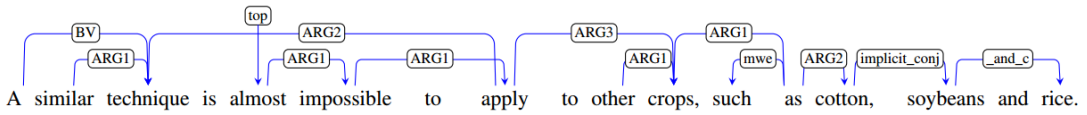

Figure 2. DM dependency graph

should know that there must be a premise when using arc-standard algorithm. Because in arc-standard algorithm, at each edge generating step, the word pointed by the edge must be removed from the configuration, i.e. each word should have at most one head in the dependency graph. So if in a specific kind of dependency graph, there are multiple edges pointed to a single word, arc-standard algorithm may introduce many mistakes into the final result.

Unfortunately, as an example shown in figure 2, DM format is not a format that satisfying that premise. It is clear that there 3 edges point to the word "technique" simultaneously. So in this assignment, arc-standard method is not appropriate, unless do tree-approximation first.

Because of the separation of edge generating and word reduction, arc-eager algorithm call deal with this problem. But the cost is that the oracle operation sequence is about twice as long as that in arc-standard algorithm. It may improve the difficulty and time of training.

3.3 Averaged Perceptron

The averaged perceptron model is a simple and effective model in classification. I have talked about this model in my 1st assignment report so I'm not going to repeat the principle and process of this model here.

Theoretically, perceptron can only deal with linear separable data, and perform little worse than SVM. The reason why I choose this model is only one word——fast. I've tried other models, but the training times are unbearable. I'll discuss this further in the discus section.

3.4 Feature Selection

I referred several papers here[2][3][4].

The features contain 2 parts. One is word information, including one-word features, word-pair features and three-word features. Another part is operation history.

In three-word features, the predicted edge information was used. But due to the number of children or parents is different among words, only the left-most child and parent features are extracted from the graph.

All the feature templates are listed in the table below:

One-word	s1l, s1p, s1lp, s2l, s2p, s2lp, b1l, b1p, b1lp, b2l, b2p, b2lp
Word-pair	s1lps2lp, s1lps2l, s1lps2p, s1ls2lp, s1ps2lp, s1ls2l, s1ps2p, s1lpb1lp, s1lpb1l, s1lpb1p, s1lb1lp, s1pb1lp, s1lb1l, s1pb1p, b1lpb2lp, b1lpb2l, b1lpb2p, b1lb2lp, b1pb2lp, b1lb2l, b1pb2p
Three-word	s2ls1lb1l, s2ps1lb1l, s2ls1pb1l, s2ps1pb1l, s2ls1lb1p, s2ps1lb1p, s2ls1pb1p, s2ps1pb1p, s2ps1plcs1p, s2ps1prcs1p, s2ps1plcs2p, s2ps1prcs2p, s2ps1llcs1p,

[2] Marinov S, Nivre J. A data-driven dependency parser for Bulgarian[C]//Proc. of the 4th Workshop on Treebanks and Linguistic Theories (TLT). 2005: 89-100.

[3] Nivre J. Incrementality in deterministic dependency parsing[C]//Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together. Association for Computational Linguistics, 2004: 50-57.

[4] Chen D, Manning C D. A Fast and Accurate Dependency Parser using Neural Networks[C]//EMNLP. 2014: 740-750.

	s2ps1llcs1p, s2ps1plps1p, s2ps1prps1p, s2ps1plps2p, s2ps1prps2p, s2ps1llps1p, s2ps1llps1p, b1ps1plcs1p, b1ps1prcs1p, b1ps1plcs2p, b1ps1prcs2p, b1ps1llcs1p, b1ps1llcs1p, b1ps1plps1p, b1ps1prps1p, b1ps1plps2p, b1ps1prps2p, b1ps1llps1p, b1ps1llps1p
Operation history	h1, h2, h3, h1h2, h1h2h3

Table 2. feature selection

s1 means the first word in stack (count from top), b1 means the first word in buffer, l means "lemma", p mean "postag", lcs1 means "left-most child of s1", rcs1 means "right-most child of s1", lps1 means "left-most parent of s1", h1 means the first operation in history stack etc.

In the training process, the size of appeared features can reach over 2 million. That's why kernel based model like kernelized perceptron does not work in my case——the dimension of feature space is too big, and result in very low speed.

3.5 Other Configuration

Because of the time limit, I didn't try lots of configurations.

5 iterations seem enough for training. And for every iteration, training sentences are random shuffled. I randomly selected 163 sentences (5% of the whole data) for model evaluation after every iteration. Before every evaluation step, the perceptron weights will be averaged.

4. Result

I will show one of the training result using the configuration I have explained above.

After 5 iterations of training, the F1-measure on labeled data can achieve 71.6%:

```
15717 ### Labeled scores
15718
15719 LP: 73.39
15720 LR: 69.90
15721 LF: 71.60
15722 LM: 9.88
```

And the F1-measure on unlabeled data goes higher——75.95%:

```
15772 ### Unlabeled scores
15773
15774 UP: 77.85
15775 UR: 74.15
15776 UF: 75.95
15777 UM: 10.49
```

I don't know whether this is the limit of this model, but according the evaluation result after every iteration, it's close to that.

5. Discussion

The performance of my model is not that good as I have expected. Actually, the baseline of *SemEval 2014 Task 8* achieved F1-score on labeled data of 54.68%. And the participant can reach over 90%. I think the 3 main reasons caused the performance gap are:

1. The training data is too small. For the formal participant, the training data consists of more than 30,000 sentences, 10-folds as I have.
2. The basic perceptron algorithm aims to linear separable data. But in this case, it obviously is not linear separable.
3. The partially parsed dependency graph is hard to contribute to the later training. Because of the limited power of one-hot representation of features.

5.1 Kernel Method

To deal with the reason 2, I have tried to implement kernel method into my model——change the basic perceptron to kernelized perceptron. In this way, in every classification step, this model will compute the inner product of current features and every features that met before. The one-hot representation format makes the feature

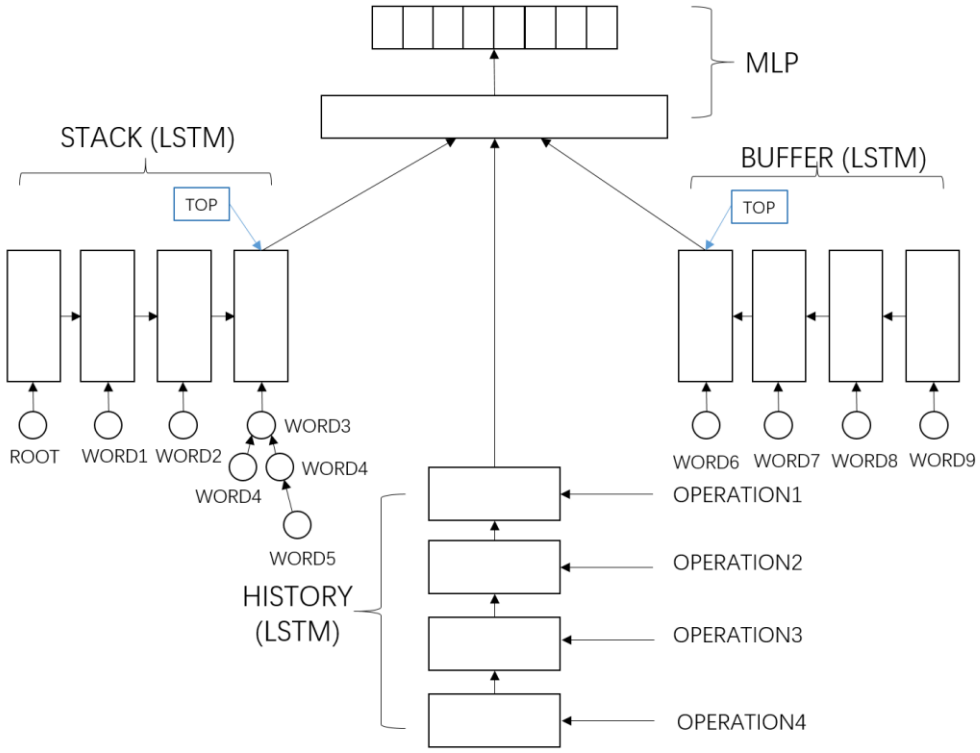

Figure 2. stack-LSTM transition-based parser

space bigger and bigger, and thus the training speed goes slower and slower.

5.2 Stack-LSTM Parser

One of an approaches to solve the "large dimension problem" is to use embedding techniques to get a relative low-dimension distributed representation. And this leads to the neural networks.

Neural network models have already been successfully used in this semantic dependency task. One of an amazing work is called stack-LSTM model[5]. The framework can be seen in figure 2. It is based on transition-based model, but the stack and buffer are replaced by LSTM. "stack-LSTM" means that if we regard a LSTM as a stack, there is a pointer "top" in LSTM always point to "the top of the stack". And by move the "top" pointer backward and forward, we can easily implement "pop" and "push" operations in a standard stack. The partially parsed graph is stored in a multilayer network called "composition tree" that computes the distributed features, which serves as an input of stack-LSTM, of a subgraph. Then stack feature, buffer feature and history feature then concatenate to a large vector and input to the softmax-MLP to get the predicate operations.

Actually in this assignment I implemented this stack-LSTM model from scratch at first (in folder ./SDP_failed). It takes me about 1 week. But without multi-threaded acceleration and GPU acceleration, the speed of training is very very slow. And that time I finally realized the importance of a well-developed deep learning toolbox. And there was not much time left for me because of the final exam, so I just roughly implemented the basic perceptron model. I'm very sorry for that.

[5] Dyer C, Ballesteros M, Ling W, et al. Transition-based dependency parsing with stack long short-term memory[J]. arXiv preprint arXiv:1505.08075, 2015.