HW3: Dependency Parsing

Zhivar Sourati Hassan Zadeh
University of Southern California
souratih@usc.edu

Abstract

Dependency parsing is an important part of Natural Language Processing (NLP) and is used in various fields and applications of this field, both directly and indirectly. On this ground, having accurate and efficient models can help numerous NLP tools. In this study we analyze an accurate and fast dependency parser starting from its building blocks and elaborate on its different components to find the components in which change are influential and can enhance or harm the model. We find that parsers can be developed cheaper with less data with comparable accuracy with the best model, and bigger models cannot always yield better results.

1 Introduction

Dependency Parsing refers to examining the dependencies between the words of a sentence to analyze its grammatical structure. Based on this, a sentence is broken into several components in which there is a direct link between every linguistic unit of a sentence. These links are termed dependencies. Dependency parsing is getting to be highly popular in different applications and domains and one reason for the increasing popularity is the fact that dependency-based syntactic representations seem to be useful in many applications of language technology, such as machine translation and Information Extraction (Wang et al., 2018), thanks to their transparent encoding of predicate-argument structure (Kübler et al., 2009).

There are different approaches of developing such systems, namely, data driven, and grammar based (Kübler et al., 2009). Focus of this study being on the data driven approach by framing the problem in a supervised learning setting, we analyze different aspect of a dependency parser proposed by (Chen and Manning, 2014). Data-driven dependency parsers having two parts of learning and parsing (Kübler et al., 2009), this dependency parser uses information from the words, part of speech (POS), as well as previously seen words' relation to their parents. Parser in this framework is also a shallow neural network that predicts the relation of words to their parents in a transition-based procedure.

In this study we reproduce the model designed by (Chen and Manning, 2014) and further do various ablation studies on different parts of the model to develop a better understanding on the effectiveness of different components in the whole framework. In the following, the Section 2 will discuss the dataset we used for our study, the transition-based model for featurization as well as parser are explained thoroughly in Section 3, and the details of our experiments will be given in Section 4. Finally, the results will be presented and further insights will be given for future works respectively in Section 5 and Section 6.

2 Dataset

The dataset we use is the dependency trees (converted from Penn Tree Bank (Marcus et al., 1993)) in CoNLL format. It provides us with the information on each token of a sentence, namely, the current sentence word number, the word and its lemma, the POS tag of the word, and finally the parent of each word in the sentence alongside its relation to its parent. Since the algorithm proposed by (Chen and Manning, 2014) only works on the projective sentences (Jurafsky and Martin, 2008), as a pre processing step, we remove all the sentences in our dataset which are non-projective utilizing the span-overlap detection algorithms. The number of sentences given in the dataset as well as the number of sentences that are projective are presented in Table 1. As you can see, the non-projective sentences do not account for a significant part of the dataset and by removing them, the change in our vocabularies is negligible.

Table 1: Statistics on the vocabulary, POS tags, and dependency labels of both datasets, with and without the non-projective sentences

Dataset	# Samples	# Vocab	# POS tags	# Relations
All sentences (Train)	39832	44390	46	40
Projective sentences (Train)	39712	44307	46	40
All sentences (Dev)	1700	6841	45	40
Projective sentences (Dev)	1695	6836	45	40

3 Methods

In this section we will explain thoroughly both, the learning as well as parsing section of the transition-based dependency parsing algorithm.

3.1 Learning

The transition-based class of parsing algorithms start from an initial unit that corresponds to the tree without any connections between tokens called a configuration and iteratively process all the tokens in the sentence until they reach the configuration that corresponds to the fully connected dependency tree.

The transition between different configurations is done through arc-standard system and shift-reduce parsing method (Nivre et al., 2006). In this system a configuration unit has three main parts: (1) the stack s, (2) the buffer b, and (3) the tree generated so far t. We start from the initial configuration $c = (s, b, t)$ that has $s = [ROOT], b = [w_1, w_2, ..., w_n], t = \emptyset$.

Note that $ROOT$ denotes the token that only has one child which is the word in the sentence that does not have a parent and is regarded as the central word of the sentence. Also w_is are the words in the sentence start from 1 to N for a sentence that has N words. At each step we apply one of the transitions below on the configuration and get a updated configuration (the transition is done only if the transition is possible, e.g., the shift is done only if the buffer is not empty, and root will never get popped as it is the remaining item in the stack in the final configuration):

- Adds the relation $s_1 \rightarrow s_2$ with label l to the t and removes s_2 from stack s, denoted as left_arc (l)
- Adds the relation $s_2 \rightarrow s_1$ with label l to the t and removes s_1 from stack s, denoted as right_arc (l) if all the children of s_1 are already popped from the buffer b
- Adds b_1 from buffer to the stack denoted as shift

Features and target labels are defined and extracted for each configuration regardless of their stack, buffer, or tree status to be fed into our dependency parser. There are in total 48 features, 18 features from the words, 18 features from the POS tags, and 12 features from the dependency labels.

In detail, the 18 word features that are extracted from each configurations comprise: (1) the top three words in the buffer and stack $s_1, s_2, s_3, b_1, b_2, b_3$, (2) the first and second leftmost and rightmost children of the top two words on the stack $(lc_1(s_i), rc_1(s_i)lc_2(s_i), rc_2(s_i), i = 1, 2)$ (3) the leftmost of leftmost and rightmost of the rightmost children of the top two words on the stack $(lc_1(lc_1(s_i)), rc_1(rc_1(s_i)), i = 1, 2)$. The 18 POS tag features are the POS tags of the words extracted from the 18 word features, and the 12 dependency label features are the dependency labels of the last 12 words of the 18 words discussed in the word features, as there are no dependency labels for the words still in the stack or buffer and yet have to be processed to be assigned parent and dependency label (Note that the tree view of the configuration is only based on the dependencies and parents that are assigned so far).

For each configuration, the 48 features discussed would build the input features and the target label is the transition done alongside the dependency label assigned to the created relation (except for when the transition is shift). More concretely we have $1 + 2L$ possible target values for each transition having L distinct dependency labels.

3.2 Parsing

Parsing is done through a simple feed-forward neural network. Having the input features and target labels, parser would simply process the input features and as a classification task will output the transition which is most probable based on features that are essentially a compact view of the stack, buffer, as well as the tree created so far. In the training phase, using teacher forcing (Williams and Zipser, 1989), different configurations from a single sentence are considered to be independent data points and ground truth of each configuration is used as the target label, rather than the predicted target label from the previous configuration. On the other hand, although this will leave room for propagated errors, the parsing is done totally greedily in the testing time. After processing each configuration, the tree, buffer, and stack get updated based on the predicted dependency label, and the view

of the next configuration and its input features are based on this updated view and not on the ground truth.

In the following we explain the details of the feed-forward neural network that is used for our classification task (classifying the target dependency label.) First of all, all inputs features go through an embedding layer to transform them into dense vectors each having $embed\_dim$ dimensions. Input features having three different types of words, POS tags and dependency labels, we use three different embedding components, one for each of the words, POS tags, and dependency labels but all having the same dimensions. The encoded values from the output of this stage will be concatenated and fed to a feed-forward neural network with three hidden layers to get the logits used for dependency label classification. The whole framework is depicted in Fig 1.

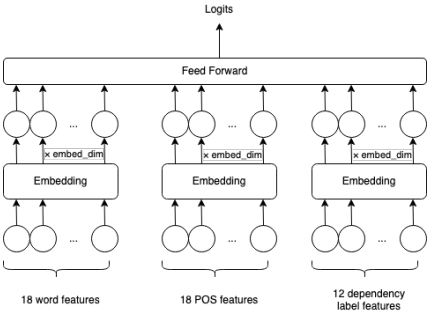

Figure 1: The structure of the dependency parser neural network

4 Experiments

To gain better insights into different parts of the dependency parser neural network, we opted to devote part of our discussion to study the effectiveness of different components and to find the best combinations of them.

Particularly, we used different hidden dimensions for the feed-forward layer as well as the embedding layer, different ratios of the data to train on, as well as analysis on the effect of external word embeddings in our network. Plus, we also compared our results on our curated test split with another blind test split on Vocareum[1] as a sanity check, presented in Table 2. Please note that experiencing similar performance with both test splits and for the sake of doing more and better experiments delving deeper into the subject we did all our experiments and also compared them with each other using our curated test set.

For the evaluation of the models and assessing their performance on dependency parsing task, we use Labeled Attachment Score (LAS) which is a standard evaluation metric in dependency parsing: the percentage of words that are assigned both the correct syntactic head and the correct dependency label. The same score can get computed ignoring labels for each dependency label that is called Unlabeled Attachment Score (UAS). We report both of these metrics as well as the weighted F1 classification metric (dependency label classification task based on the input features discussed in Section 3.1) on all our results and analysis.

Table 2: Performance of the model on the curated test set and blind test set (this does not reflect on the best model and the best performance reported in Section 5)

Dataset	UAS	LAS
Curated Test set	0.858	0.799
Blind Test set	0.849	0.799

For all the experiments the learning rate was set to 0.08 and a l2 regularization with a weight of 0.0001 was applied on our loss terms. Also a batch size of 16 was used in all our experiments with a dropout rate of 0.5. The results and our discussion on the differences between these components are explained further in Section 5. For parsing, a CPU machine was used, while for the training, all the models were trained on a A100-PCIE-40GB GPU.

5 Results

In this section, we present the results of our study alongside with ablation studies we did with different components.

Number of features in the embedding layer can have a strong effect on the amount of information that can be embedded from words, POS tags, and dependency labels and modeled eventually in the dependency parser. As we used GloVe (Pennington et al., 2014) for our word embeddings and it has four different variations with $\{50, 100, 200, 300\}$ dimensions, we chose the same numbers for the other two embedding components as well, and developed models each with different number of features in the embedding layer that its results are shown in Table 3, showing drop in the performance with using more features in the embedding layer. We attribute this observation to overfitting and that increasing the size of the embedding layer solely makes the model capable of fitting the noise better and the information to be captured in the embedding layer is already captured

[1] https://www.vocareum.com/

Table 3: Performance of the model using different hidden dimensions as well as different embedding dimensions for the feed-forward network

# Embed_dim	UAS	LAS	Dev F1
50	0.889	0.832	0.959
100	0.872	0.814	0.953
200	0.860	0.802	0.949
300	0.850	0.792	0.944

# Hidden_dim	UAS	LAS	Dev F1
50	0.874	0.817	0.954
100	0.884	0.827	0.958
300	0.889	0.832	0.959
500	0.878	0.821	0.954

Table 4: Performance of the model using external word embedding as well as our own word embedding trained from scratch

Embedding type	UAS	LAS	Dev F1
External	0.889	0.832	0.959
From scratch	0.888	0.832	0.960

even in the smallest embedding layer we analyzed.

Hidden dimension of the feed-forward network as a hyper parameter was analyzed with different sizes to check whether the size of the model can contribute to the performance of the model or not. We used hidden dimensions from the set $\{50, 100, 300, 500\}$. Table 3 depicts the performance of the model having different hidden dimensions. It can be seen in Table 3 that the larger feed-forward networks boost the performance of the model until a particular point and from there a drop can be observed. This shows that increasing the size of the model cannot always improve the model and more subtle change in the architecture and features are needed.

Different proportions of the data can be used for training and developing models with limited access to the training data to assess the amount of data that is needed to develop sufficiently accurate dependency parsers. In our experiments we tried to train models with $\{0.1, 0.4, 0.7, 1.0\}$ of the training data. The results of our analysis illustrated in Fig 2, show that even with smaller datasets, we can develop models that can reach 0.8 in UAS and LAS. In fact, keeping everything else in the model the same, the performance of the model with 0.7 of the data is almost as same as all the training data.

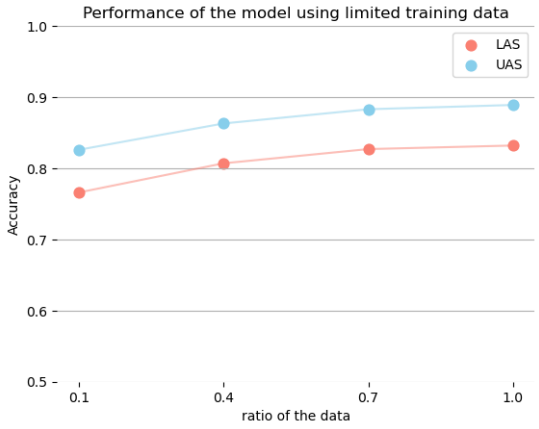

Figure 2: The performance of the model using limited training data

External word embeddings can assist the model with the semantic information on the words of the sentences that are trained on large amount of text and already capture useful semantic attributes between similar or related words. On the other hand, having large enough corpus at hand to train models alongside their embedding layers from scratch, we compared both external word embeddings and our own word embedding that has to be trained from scratch. This comparison is shown in Table 4. We can see that both models are performing on par with each other and no clear strong conclusion can be made on this ground. Nevertheless, the fact that external word embedding are trained on large corpora and the only way we are developing our word embeddings from scratch is through dependency parsing, can tell us that the especial attributes captured in external word embeddings are not essential for having accurate dependency parsers.

6 Conclusion

Dependency parsing plays an important role in various NLP tasks and applications, and having models that can efficiently and accurately generate the dependency tree from a sentence is crucial. Trying to develop a better broad intuition over the overall framework from the featurization until the parsing, we used a transition-based dependency parser proposed by (Chen and Manning, 2014) and analyzed various parts of the parser. Our results show that bigger models especially in the embedding layer do not help the model. Also we argue that the extent to which the task is relied on semantic information is not as much to benefit from external word embeddings. Finally we claim that even with smaller proportions of the dataset, we can develop accurate enough models. While this work tries to have a broad overview of design choices and hyper-parameters more intuitively, further work can be done to focus on the nuances of the combination of different architectures and hyper-parameters with more theoretical groundings.

References

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Dan Jurafsky and James Martin. 2008. *Speech and Language Processing*, 2 edition. Pearson, Upper Saddle River, NJ.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Springer International Publishing.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Yu-Siang Wang, Chenxi Liu, Xiaohui Zeng, and Alan Yuille. 2018. Scene graph parsing as dependency parsing.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280.