OCLSP at SemEval-2016 Task 9: Multilayered LSTM as a Neural Semantic Dependency Parser

Lifeng Jin **Manjuan Duan** **William Schuler**
Department of Linguistics
The Ohio State University
{jin,duan,schuler}@ling.osu.edu

Abstract

Semantic dependency parsing aims at extracting arcs and semantic role labels for all words in a sentence. In this paper, we propose a semantic dependency parser which is based on Long Short-term Memory and makes heavy use of embeddings of words and POS tags. We describe in detail the implementation of the neural parser, including preprocessing, postprocessing and various input features, and show that the neural parser performs close to the top system in the shared task and is very good at capturing non-local dependencies. We also discuss some issues related to the parser and how to improve it.

1 Introduction

Semantic dependency parsing is a form of semantic analysis where semantic roles for all words in a sentence are analyzed and specific semantic relations are assigned to each word pair (Che et al., 2012). Chinese semantic dependency parsing is especially of interest as there is a remarkable difference between syntactic and semantic dependencies in Chinese (Che et al., 2012). In the SemEval 2016 Task 9 Chinese Semantic Dependency Parsing shared task, semantic dependency parsing for two text genres, TEXT, which includes sentences from conversations and primary school textbooks, and NEWS, which contains newswire text, is explored. We propose a neural parser with LSTM as the basic units and make heavy use of vectorial representations of basic linguistic units like words and POS tags. In this paper, we give a detailed description of the neural parser and discuss a few issues related to the current implementation.

2 Long Short-term Memory

Recurrent Neural Networks, or RNNs, are good for sequential prediction, but the problem of exploding or vanishing gradients makes learning long distance dependencies very difficult for them (Hochreiter, 1998). The LSTM architecture is proposed to address this problem (Hochreiter and Schmidhuber, 1997). In this paper, we follow the version defined in Graves et al. (2013b).

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (2)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \quad (4)$$

$$h_t = o_t \tanh(c_t) \quad (5)$$

where σ is the sigmoid function, and i, f, o and c are the input gate, forget gate, output gate and the cell respectively. x_t is the input at time step t, and h_t is the hidden state or the output of the LSTM unit at time step t. With various gates and a cell, the LSTM unit may learn to store and release information inside the cell over many time steps. LSTMs have recently been used for various NLP tasks such as machine translation (Bahdanau et al., 2014; Sutskever et al., 2014), syntactic parsing (Vinyals et al., 2015) and semantic relatedness prediction and sentiment classification (Tai et al., 2015). In this paper we apply the LSTM architecture to semantic dependency parsing and show that to use LSTM as a semantic dependency parser is both very powerful and very promising.

3 Overview of the System

3.1 Parsing Typology

For a sentence $S = \langle w_1, w_2, ...w_i, ...w_n \rangle$, the proposed parser traverses the whole parse chart linearly through all possible pairwise combinations of the elements in the sequence to predict the dependency labels. For example, the parser will first consider the word pair $\langle w_1, w_1 \rangle$ for dependency label prediction, treating the first in the pair to be the dependent and the second the head in a dependency relation, and then the word pair $\langle w_1, w_2 \rangle$, and so on. When it gets to $\langle w_i, w_n \rangle$, it proceeds to parse $\langle w_{i+1}, w_1 \rangle$, until the final pair $\langle w_n, w_n \rangle$ is parsed. For cases like $\langle w_i, w_i \rangle$, because there is no dependency arc that connects the same word in the same position, we use this for predicting the root of the sentence. Therefore the actual input to the parser is quadratic on the length of the input sequence, making the complexity of the parser $O(n^2)$.

3.2 Architecture of the Neural Parser

The neural parser is shown in figure 1. It consists of two main parts: the lower level which has a bidirectional LSTM as its main component and the upper level which has an LSTM and an output layer as its main component.

Lower Level: The main component of the lower level system is a bidirectional LSTM (Graves et al., 2013a). The bidirectional LSTM takes the same input sequence with n time-steps, and runs both forward and backward in time to generate hidden states for each time-step, and concatenates the two hidden states for a single time step to form a single hidden state of the whole system. For example, for a time sequence $X = \langle x_1, x_2, ...x_t, ...x_n \rangle$, the input to the bidirectional LSTM is x_t, and the output from it is the concatenation of the hidden state from the forward LSTM h_t and the hidden state of the backward LSTM h'_t. This bidirectional feature, which is also exploited by Bahdanau et al. (2014) for neural machine translation, provides information for the current time step with information gathered both from the beginning of the sentence and the end of the sentence in terms of cell states and the hidden states.

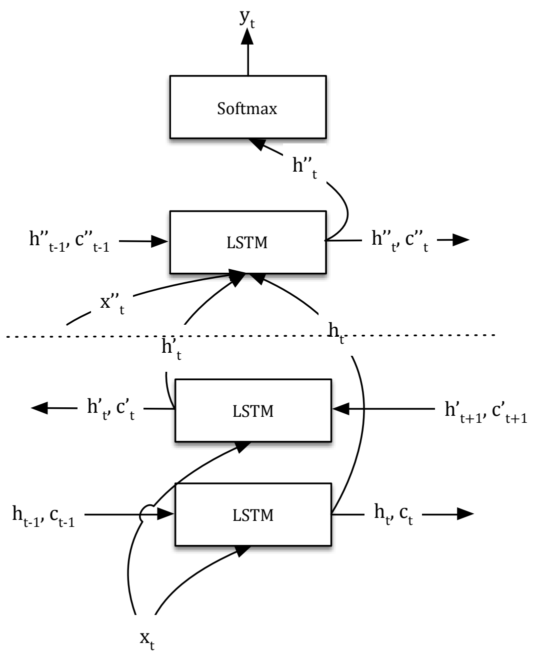

Figure 1: The OCLSP neural parser. x, h, c and y are the model input, hidden state of a LSTM, cell state of a LSTM and the model output respectively.

Upper Level: The main component of the upper level system is an LSTM. The LSTM takes in the output from the lower level (h_t, h'_t) as well as a secondary input x''_t the system and generates a hidden state h''_t which is then fed into a softmax layer to transform into a probability distribution. If $x_t = f(w_i, w_j)$ where w is a word in a sentence, f is the function from words to input vectors and i and j are the indexes of the words, then the final output of the upper level of the system can be viewed as $p(label_{i,j}|f(w_1, w_1), ..., f(w_i, w_j), ..., f(w_n, w_n), \theta)$, where θ is the model parameters.

3.3 Input Features

The input to the parser consists of two parts: the primary input x_t which is the input fed to the lower level system and the secondary input x''_t which is fed to the upper level system. In development, different kinds of input features are added to the model incrementally to help feature selection. The f function mentioned in the model description translates word pairs to concatenation of different feature vectors.

The system makes heavy use of vectorial representations trained with *word2vec* (Mikolov et al., 2013). There are two kinds of embeddings used by the parser: word embeddings and Part-Of-Speech embeddings. The word embeddings are all trained with the full Chinese Wikipedia, [1] the Chinese Gigaword (Graff and Chen, 2003) as well as the training and development datasets from both genres in this task. The POS embeddings are trained with the POS tag sequences from training and development datasets from both genres in this task. The embeddings are fixed during training and test.

Basic features: The basic features for the lower level system are 300-dimension word embeddings for the current word pair $\langle w_i, w_j \rangle$. The basic features for the upper level system are 50-dimension word embeddings for the context words of the current word pair with a window size of 4, i.e. $\langle w_{i-3}, w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}, w_{i+3}, w_{j-3}, w_{j-2}, w_{j-1}, w_{j+1}, w_{j+2}, w_{j+3} \rangle$. The 50 dimensional word embeddings are used because we want the word embeddings at the upper level of the same length as the word embeddings in the lower level, and also we want to keep the number of parameters of the upper level inside a practical range for training. Whenever there is no word at a certain position of the window, an all-zero vector is used to indicate that there is no information from that position for the system to use.

POS features: The POS features for the lower level system are 50 dimensional POS embeddings for the current word pair, and for the upper level system, the 50 dimensional POS embeddings are also used for the context words of the current word pair. The all-zero vector is also used whenever a position inside the window has no word.

[1] `https://zh.wikipedia.org`

Model	Labeled F	Unlabeled F
TEXT		
basic	61.26%	75.31%
basic+pos	65.58%	78.98%
basic+pos+gcg	67.68%	81.12%
NEWS		
basic+pos	57.94%	74.42%
basic+pos+gcg	58.95%	75.44%

Table 1: Results of models with different feature combinations on development dataset. The performance of the model with only basic features on NEWS is not reported here because it became evident that it never outperformed the other two models with more features.

Predicate-argument features: In previous work, non-local dependencies expressed using a generalized categorial grammar (Bach, 1981; Nguyen et al., 2012) have been used to train an off-the-shelf PCFG parser (Petrov and Klein, 2007) to accurately parse Chinese (Duan and Schuler, 2015). Predicate-argument dependency features were extracted from parsed training sentences using this representation and the trained parser, which gives directed dependencies with 16 different labels, like *arg-1* and *Nmod*, for some pairs of the words. These dependencies are transformed first into undirected dependencies with the original labels. This transform mitigates some problems with arbitrary arc direction discrepancies between two dependency schemes. Then the labels of these dependencies are converted into one hot vectors for the upper level system to use. For the word pairs without a predicate-argument dependency, an all-zero vector of length 16 is used.

Table 1 shows the labeled and unlabeled F scores for models trained with different feature sets on the development dataset. The POS embeddings are good indicators of semantic dependencies as it improves the scores on TEXT by about 4 percent, and the predicate-argument features further improve the F score by another 2 percent on TEXT, and 1 percent on NEWS. The submitted systems use all three feature categories to train and test.

3.4 Preprocessing and postprocessing

There are a few preprocessing and postprocessing steps that are adopted in the system to generate better results.

Word Segmentation: The datasets provided in the task come pre-segmented, but the training data used by *word2vec* need to be segmented first to produce embeddings. We use NLPIR 2015 (Zhang et al., 2003) for segmentation with all words in training and development datasets put into the user dictionary for segmentation. This guarantees that there is no out-of-vocabulary word in development, which also increases parser performance on development data.

OOV Replacement: The OOV words are processed in the following way. Suppose there is a large dataset D_1 for training word embeddings. The vocabulary of D_1 is the set V_1. Suppose there is a new dataset D_2 which has the vocabulary set V_2. The OOV words from D_2 are $V_{oov} = V_2 - V_1$. We first train an embedding model W_1 for D_1 and W_2 for D_2, and then for each word v_{oov} in V_{oov}, we find the word v in V_1 which has the highest similarity score by cosine to v_{oov} in W_2, and replace the v_{oov} with v in V_2. This procedure essentially replaces all the OOV words with the most similar words in the current semantic space of the word embeddings. By doing this, the models do not have to be retrained every time the semantic space of all word embeddings is shifted from W_1 to W_2 with addition of new embeddings. Also the replaced embeddings have more semantic information of the OOV words than a general UNK symbol.

Graph Generation: The proposed parser has no guarantee to produce a dependency graph with all nodes connected to the root. In practice, the raw output of the parser is often not a graph. The parser is generally over-conservative, so many words do not have a dependency arc at all. Therefore the Chu-Liu/Edmond's algorithm[2] is applied to the probability matrix for a complete parse chart of a sentence with the reciprocals of the probability values as costs of edges between nodes, and the minimal spanning directed graph is generated for the sentence. This guarantees that all final outputs are directed graphs.

The Chu-Liu/Edmond's algorithm outputs a directed graph with no reentrance, but semantic dependency graphs in the task have reentrant nodes. In order to produce reentrant dependencies, we set a threshold of σ on the probabilities for the arcs of a head word. When the parser decides there are multiple heads for a dependent, and the probabilities of the extra arcs are higher than σ, these arcs will be incorporated into the dependency graph. Two submitted systems use this method to generate final results. One system uses the direct output from the parser and generates an arc for any word without a head by adding an arc with the highest probability between the word and any candidate head.

3.5 Training

The loss function of the parser is set to be Negative Log-likelihood function and the training objective is to maximize the log-likelihood of the dependency labels given all word pairs. We use mini-batch update and Adagrad (Duchi et al., 2011) to optimize the parameter learning. Each sentence is a mini-batch. The models used for different datasets are trained separately. We used the F score on the development dataset as the metric to measure convergence. For TEXT, there are in total 10754 sentences in training and for NEWS there are 8301 sentences in training. There are in theory 171 possible dependency labels and 1 no-dependency label in training, but in practice we only observe 158 different semantic dependency labels and 1 no-dependency label in training and development. The models for TEXT are trained for 12 epochs, and the models for NEWS are trained for 8 epochs.

4 Task Evaluation Results

Three systems with slightly different configurations for the Graph Generation process were submitted for test. The results are shown in Table 2. The *lbpg* system does not use the Chu-Liu/Edmond's algorithm for graph generation. Instead it just generates an arc for a word without a head by adding an arc with the highest

[2] `https://github.com/mlbright/edmonds`

Model	**LF**	**NLF**	**UF**	**NUF**
TEXT				
lbpg	65.54%	**57.51%**	79.39%	63.21%
lbpgs	66.21%	47.79%	79.85%	55.51%
lbpg75	66.38%	**57.51%**	79.91%	63.87%
TOP	**68.59%**	50.57%	**82.41%**	**64.58%**
NEWS				
lbpg	57.22%	45.57%	74.93%	58.03%
lbpgs	57.81%	41.56%	75.54%	54.34%
lbpg75	57.78%	**48.89%**	75.40%	58.28%
TOP	**59.06%**	40.84%	**77.64%**	**60.20%**

Table 2: Results of the submitted models on test dataset. LF, NLF, UF and NUF are respectively Labeled F, Non-local Labeled F, Unlabeled F and Non-local Unlabeled F scores.

probability between the word and any candidate head. It is the worst performing model among the three in terms of F scores. The *lbpgs* and *lbpg75* are both models with the graph generation algorithm. The difference between them is that *lbpgs* uses a σ value of 100, which means that all words have only one head in its output, whereas *lbpg75* uses a σ value of 75 meaning that the extra arcs of a word must have a probability over 0.75.

For the *lbpgs* system, it is interesting to see that only allowing one head per word does not seem to have a huge impact on the F scores. It actually performs better on the NEWS dataset in terms of LF and UF. This indicates that the systems do not perform very well on the extra arcs, either predicting the wrong labels for the arcs or predicting the wrong arcs altogether. However, the non-local scores see a big drop.

For the *lbpg75* system, it is clear that by choosing a high threshold of extra arc probability, in TEXT at least, we get slightly higher F scores. In both datasets, the non-local scores are the highest with this model, meaning that for the arcs with high confidence, the non-local dependency arcs within them are relatively accurate.

5 Discussion and Future Work

Negative Training Cases: One of the most obvious drawbacks of systems like this is that there is a disproportional amount of negative training cases in the training data. In training, the whole parse chart needs to be filled with labels. However, most of the time the number of arcs in a sentence does not exceed $2n$. Therefore the number of word pairs with the no-dependency label, or the number of negative cases, is $n^2 - 2n$. When n gets big, for example for some sentences in NEWS n can be over 200, the number of negative cases gets very large compared to the number of positive cases. It is interesting to see that the parser still captures fairly well the information from positive cases, which are few compared to the negative ones. However, but it may be even better if the number of negative cases can be greatly reduced, the information from positive cases can be even more accessible and the training time can be shorter.

OOV Words: The parser relies on word embeddings and POS embeddings to make dependency predictions. Due to different segmentation schemes used by the automatic segmenter and the annotators, about 10% of the vocabulary in test is OOV words even when the word embeddings are trained with Chinese Wikipedia and Chinese Gigaword combined. The OOV strategy adopted in this paper is shown to be better than just replacing all OOVs with a uniform vector like all-ones or all-zeros, in which case the parser just treats all OOVs as punctuation. However the relation between the OOV words and the replacement words do not seem to correspond to any linguistic or world knowledge for most of the replacement-OOV word pairs. For example, the most similar word for replacement of the OOV word “马来语” (Malay language) is “910129”, and the most similar word for “税改” (tax reform) is “面谈成绩” (interview performance). OOVs are a challenge to all parsers, but it is especially important to parsers like the one in this paper, where its performance depends crucially on the quality of the embeddings.

Parsing Typology: The parsing typology used in this paper alleviates problems like reentrant nodes and crossing arcs, but at a cost of efficiency and accuracy. Only about 3% of all the dependency arcs are extra arcs, i.e. they are second or third dependency heads for a given dependent. Such dependencies are very hard to predict, and the typology used here, although able to generate such dependencies naturally, is less than ideal in different ways described above. More research needs to be done for a better parsing typology for neural parsers which addresses these issues without losing the ability to predict extra arcs.

6 Conclusion

We present a neural parser with LSTM in this paper for the task of Chinese semantic dependency parsing. We have shown that such a parser may perform close to the top performer in the task with minimal feature engineering. We also discussed places where potential improvements can be made.

References

Emmon Bach. 1981. Discontinuous constituents in generalized categorial grammars. *Proceedings of the Annual Meeting of the Northeast Linguistic Society (NELS)*, 11:1–12.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *ICLR*, pages 1–15.

Wanxiang Che, Meishan Zhang, Yanqiu Shao, and Ting Liu. 2012. Semeval-2012 task 5 : Chinese semantic dependency parsing. In *SemEval*, pages 378–384.

Manjuan Duan and William Schuler. 2015. Parsing chinese with a generalized categorial grammar. In *Proceedings of Grammar Engineering Across Frameworks 2015*, pages 25–32.

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

David Graff and Ke Chen. 2003. Chinese gigaword ldc 2003t09.

Alex Graves, Navdeep Jaitly, and Abdel Rahman Mohamed. 2013a. Hybrid speech recognition with deep bidirectional lstm. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings*, pages 273–278.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013b. Speech recognition with deep recurrent neural networks. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (3):6645–6649.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3:1–12.

Luan Nguyen, Marten van Schijndel, and William Schuler. 2012. Accurate unbounded dependency recovery using generalized categorial grammars. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING '12)*, pages 2125–2140.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of {NAACL HLT} 2007*, pages 404–411. Association for Computational Linguistics, 4.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *NIPS*, pages 3104–3112.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *ACL*, pages 1556–1566.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *NIPS*.

Hua-Ping Zhang, Hong-Kui Yu, Xiong De-Yi, and Qun Liu. 2003. Hhmm-based chinese lexical analyzer ictclas. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*, volume 17, pages 184–187. Association for Computational Linguistics.