CUNY-PKU Parser at SemEval-2019 Task 1: Cross-lingual Semantic Parsing with UCCA

Weimin Lyu†, Sheng Huang‡, Abdul Rafae Khan†, Shengqiang Zhang‡ , Weiwei Sun‡, Jia Xu†⋄

Computer Science Department, †Graduate Center and ⋄Hunter College, City University of New York
{wlyu,akhan4}@gradcenter.cuny.edu, Jia.Xu@hunter.cuny.edu

‡Institute of Computer Science and Technology, Peking University
{huangsheng,ws,sq.zhang}@pku.edu.cn

Abstract

This paper describes the systems of the CUNY-PKU team in "SemEval 2019 Task 1: Cross-lingual Semantic Parsing with UCCA"[1]. We introduce a novel model by applying a cascaded MLP and BiLSTM model. Then, we ensemble multiple system-outputs by reparsing. In particular, we introduce a new decoding algorithm for building the UCCA representation. Our system won the first place in one track (French-20K-Open), second places in four tracks (English-Wiki-Open, English-20K-Open, German-20K-Open, and German-20K-Closed), and third place in one track (English-20K-Closed), among all seven tracks.

1 Introduction

We participate in all seven tracks in Cross-lingual Semantic Parsing at SemEval 2019. Our submission systems[2] are based on BiLSTM using TUPA (Hershcovich et al., 2017a, 2018).

Then, we built a second single parser using BiLSTM (Bi-directional LSTM) and Multi-Layer Perceptron (MLP) with TUPA (Hershcovich et al., 2017a, 2018). Most importantly, we introduce a new model Cascaded BiLSTM by first pre-training the BiLSTM and MLP model and then to continue training another MLP model. The cascaded BiLSTM parser significantly enhances the parsing accuracy on all tasks. We also complete a Self-Attentive Constituency Parser (Kitaev and Klein, 2018a,b) as comparison. Finally, we ensemble different parsers with a reparsing strategy (Sagae and Lavie, 2006). In particular, we introduce a novel algorithm based on dynamic programming to perform inference for the UCCA representation. This *decoder* can also be utilized as a core engine for a single parser.

In the post-evaluation stage, our improved systems are ranked first in three tracks (French-20K-Open, English-20K-Open and English-Wiki-Open) and second in the other four tracks. A shared task summary paper (Hershcovich et al., 2019) by competition organizers summaries the results.

We will describe our systems in detail, including three single parsers in Section 2 and a voter in Section 3. We focus on two novel technical contributions: the Cascaded BiLSTM model and the Reparsing strategy. In Section 4 we will present experimental setup and results.

2 Single Parsers

2.1 TUPA Parsers

The TUPA parser (Hershcovich et al., 2017a) builds on discontinuous constituency and dependency graph parsing and makes some improvements especially for the UCCA representation. The English parsing is based on Hershcovich et al. (2017a), while French and German parsing is based on Hershcovich et al. (2018).

It has been shown that the choice of model plays an important role in transition-based parsing (Hershcovich et al., 2017b). For TUPA, we built parsers with different models: MLP, BiLSTM, and also invent a new architecture, viz. Cascaded BiLSTM. The three single parsers are described as the following:

The MLP parser (Hershcovich et al., 2017b) applies a feedforward neural network with dense embedding features to predict optimal transitions given particular parser states. This parser adopts a similar architecture to Chen and Manning (2014).

The BiLSTM parser (Hershcovich et al., 2018) applies a bidirectional LSTM to learn con-

[1]https://competitions.codalab.org/competitions/19160

[2]https://github.com/weimin17/semEval-taks1

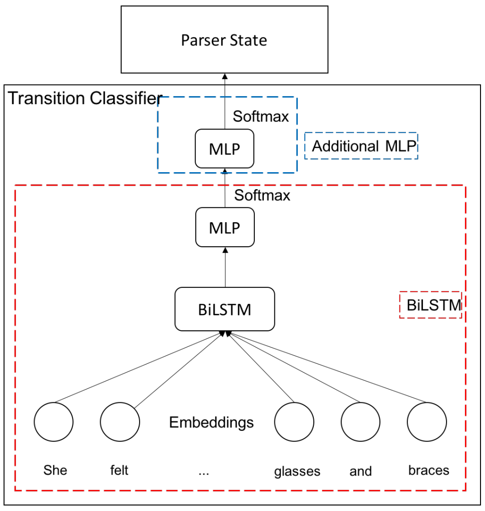

Figure 1: Illustration of the multi-stage Cascaded BiLSTM model. Top: parser state. Bottom: BiLTSM with two MLP architectures. The red box represents BiLSTM (Hershcovich et al., 2018), and the blue box represents a MLP that we add after implementing the BiLSTM architecture. Vector representation for the input tokens is computed by two layers of bidirectional LSTMs then fed into the double MLP with Softmax to select the next transition.

textualized vector-based representations for words that are then utilized for encoding a parser state, similarly to Kiperwasser and Goldberg (2016). The red box in Figure 1 shows the architecture of BiLSTM model, indicating that the representations after BiLSTM are fed into a Multiple-layer perceptron.

The Cascaded BiLSTM parser combines the above two parsing models, which contains a multi-stage training process. First, we use BiLSTM TUPA model to train 100 epochs, then retrain the model using MLP TUPA model for another 50 epochs. It's really interesting that the performances remains as good as BiLSTM TUPA model, even slightly better. Figure 1 shows the architecture of Cascaded BiLSTM model.

2.2 Phrase Constituency Parser

We also built a Constituency Parser as comparison, which uses a self-attentive architecture that makes explicit the manner considering information propagating between different locations in the sentences (Kitaev and Klein, 2018a,b). The constituency parser uses parsing tree structures as input and output. Therefore, we convert the phrase structure tree format into UCCA XML formation and vice versa.

3 The Reparsing System

The reparsing system (voter) takes multiple single parser (as in Section 2) results as input and produces a single, hopefully, improved UCCA graph as output. Briefly, each input UCCA graph is encoded to a chart of scores for standard CKY decoding. In this step, we utilize a number of auxiliary labels to encode remote edges and discontinuous constructions. These scores are summed up to get a new chart, which is used for CKY decoding for an immediate tree representation as the *voting* result. An immediate tree is then enhanced with reference relationships. Finally, a UCCA graph is built via interpreting auxiliary labels.

Span representation Graph nodes in a UCCA graph naturally create a hierarchical structure through the use of *primary edges*. Following this tree structure, we give the definition of span of nodes.

Definition 1. The span of node x is:

1. empty if x is an implicit node;
2. $[p, p+1)$ if x is a leaf node but not an implicit node, where p is the position of the lexical unit corresponding to x;
3. the union of spans of x's children, otherwise.

Assuming that each span of nodes is consecutive (we will deal with nonconsecutive spans in Section 3). We encode the label of edge from x's parent to x as the label of span of x. If there are some implicit nodes in x's children, the labels of edges from x to them are also encoded by the label of the span of x. If the span of x is the same as x's parent, the label of this span will be encoded ordered. This process is well-defined due to the acyclic graph structure. Each parser is assigned a weight to indicate its *contribution* to reparsing. The spans with labels encoded from a UCCA graph are assigned the same score according to which parser they come from. Thus, there is a set of scored spans for each UCCA graph. Following the parsing literature, we call this set a chart. We merge multiple charts produced by different parsers to a single chart simply by adding the corresponding scores.

Handling Remote Edges A remote edge with label L from node x to node y is equal to a primary edge with label L from x to an implicit node, which is referred to node y. Hence, if we can find the relationships of references, the remote edges are able to be recovered.

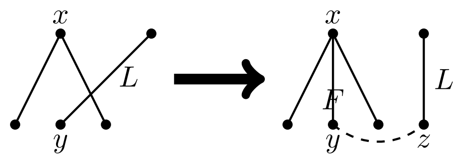

Figure 2: Remove nonconsecutive spans

Since all primary edges from nodes to their parent are encoded in labels of spans, each node could be represented as part of the label of a span. We encode each reference of a remote edge as a pair of two nodes with a score. After building all primary edges through dynamic programming, we search for available references with the maximum score in each implicit node greedily and leverage these references to recover remote edges.

Handling Discontinuous Spans Discontinuous spans are removed by repeating the following steps:

Step 1. Find a node x with a nonconsecutive span with the minimum starting point and minimum height, supposed its consecutive sub-span with minimum starting point is $[a, b)$.

Step 2. Find a node y with a consecutive span with starting point b and maximum height, supposed the primary edge from y's parent to y is e.

Step 3. Create a node z with a special type MIRROR and create a primary edge with the label of e from y's parent to z. Remove the primary edge e and create a primary edge with a special label FAKE from x to y.

After each iteration, the span of y is added to x, and the sum of the length of nonconsecutive spans decreases. Each primary edge in an original UCCA graph can only be removed once. To that end, the running time of this algorithm is linear in the number of lexical units. If all references of MIRROR nodes are correctly predicted, the expected UCCA graph will be obtained. In this way, remote edges can be handled.

4 Experiments

4.1 Data Statistics

The semantic parsing task is carried out in three languages: English, German and French, including three training data sets and parallel four test data sets. For English data, we use the Wikipedia UCCA corpus (henceforth Wiki) as training and development data, testing on English UCCA Wikipedia corpus as the in-domain test.

Tracks	Training		Dev	Test
	Closed	Open		
En-Wiki	4113	5132	514	515
En-20K	0	5132	0	492
Ge-20K	5211	6360	651	632
Fr-20K	15	547	238	239

Table 1: Sentence number in training, dev, and test sets for English, German and French UCCA data sets.

Meanwhile, English UCCA Twenty Thousand Leagues Under the Sea English-French-German parallel corpus (henceforth 20K Leagues) serves as an out-of-domain test set. For German data, we use 20K Leagues corpus for train, development, and test sets. For French data, they provide only limited training data, along with development and test data sets.

Table 1 shows the sentences number of data sets for all three languages. We use the closed track data and UCCA's annotation resources for open tracks. We merge those resources and build our open track data[3].

4.2 TUPA Parsers

We build MLP and BiLSTM systems using TUPA (Hershcovich et al., 2017b). For Cascaded BiLSTM model, we add another MLP after the BiLSTM model, which forms a cascaded BiSLTM. For closed tracks, we train models based on the gold-standard UCCA annotation from official resources. For open tracks, We use additional UCCA data from other open sources as training data set. We also generate synthetic data by automatically translating text (Khan et al., 2018) and its parsing labels across languages in our on-going work.

Table 2 shows the results for four models in different tracks. The italicized values are our official submission. However, we have made some improvement after the Evaluation Phrase, and the bold results are our best results. The first three models are single systems and the fourth model (Ensembled) ensembles different frameworks by reparsing systems. The "baseline" represents the baseline that competition provides for reference.

By using feedforward Neural Network and embedding features, MLP models get the lowest scores. BiLSTM models achieve better results than MLP models in F1 scores, both in the in-domain and out-of-domain data sets. However, the

[3]https://github.com/weimin17/semEval-taks1

Tracks		MLP	BiLSTM(Submit)	Cascaded BiLSTM	Ensembled	baseline
closed	En-Wiki	0.650	*0.718*	0.721	**0.728**	0.728
	En-20K	0.617	*0.669*	0.673	**0.681**	0.672
	Ge-20K	0.699	*0.797*	**0.797**	0.797	0.731
open	En-Wiki	0.784	*0.800*	0.843	**0.846**	0.735
	En-20K	0.715	*0.739*	0.764	**0.770**	0.684
	Ge-20K	0.598	*0.841*	**0.841**	0.840	0.791
	Fr-20K	0.535	*0.796*	0.795	**0.796**	0.487

Table 2: F1 scores for both closed and open tracks in SemEval Task 1 2019 competition. The italic text represents our official submission in competition and the bold text represents our best F1 scores.

Open Tracks	F1 Scores
English-Wiki	0.75
English-20K	0.785

Table 3: F1 scores on unlabeled data.

Models	test sets	dev sets
CharsLSTM	91.96	92.21
ELMO	94.31	94.75

Table 4: F1 scores for two constituency parsers on both Penn Treebank dev and test data sets.

combination of BiSLTM and MLP models (Cascaded BiLSTM model) performs best among the three models in all results of single systems.

Our in-house reparsing system ensembles the above parsers as described in Section 3. We can see that ensemble results are better at closed track, but not as good as the best results by Cascaded BiLSTM at Open track.

4.3 Phrase Constituency Parser

For Phrase Constituency Parer, we only test the performance on unlabeled data instead of labeled data, while for TUPA we test on labeled data.

First, we use Benepar[4], a parsing tool using out-of domain pre-trained models to predict the labels, the outputs are parsing tree structures.

Second, we convert the constituency parsing tree structure to Conllu Format. We develop a one-shot tool to improve the efficiency of conversion based on TreebankPreprocessing[5], which can automatically convert a batch of files in one directory.

Finally, we convert Conllu format to UCCA XML using format[6].

We only experiment on English-Wiki and English-20K open track, and the results are pretty bad, as shown in Table 3. We hypothesize there are two reasons: 1. The conversion process could unavoidably cause accuracy loss. 2. The third-party pre-trained models are not as efficient as the models trained directly on the specific UCCA data.

To test the accuracy on unlabeled data and to evaluate how many losses are there during the conversion process, we evaluate the accuracy in the parsing tree structure phrase before the conversion. We experimentally validate our system on the English Wiki data set. We use official training data set as training data, splitting official dev set into two parts and separately serving as our dev set and test set. We also use two models of constituency parser: ELMO and CharsLSTM, tested on Penn Treebank (Cross and Huang, 2019) data.

Table 4 indicates that, for Penn Treebank data sets, CharsLSTM model's F1 score achieves 92.21 on dev data set, with F1 score 91.96 on the test dataset. Using ELMo, The dev dataset's F1 score achieves 94.75, with F1 score achieves 94.31 on test data set.

5 Summary

Our submission systems mainly contain a BiLSTM, an MLP, and a cascaded BiLSTM parser, as well as a voted system of above. Our final system ranks first in three tracks, French-20K-Open, English-20K-Open and English-Wiki-Open, and the second place in the other four tracks in the post-evaluation.

[4]https://github.com/nikitakit/self-attentive-parser

[5]https://github.com/hankcs/TreebankPreprocessing

[6]https://github.com/huji-nlp/semstr/blob/master/semstr/convert.py

Contributions and Acknowledgements

Weimin Lyu: built all TUPA Parsers, a self-attentive Parser, convert UCCA graph as constituency tree, verify the voting systems, and draft the paper. **Sheng Huang** and **Shengqiang Zhang**: built the reparsing system and UCCA-Dependency graph transformer. **Abdul Rafae Khan**: built cross-lingual parsers by generating synthetic data with machine translation. **Weiwei Sun**: extensively supervised PKU team, and **Jia Xu**: closely supervised CUNY team, in algorithms and experiments. We thank the initial work of Mark Perelman. This research was partially funded by National Science Foundation (NSF) Award No. 1747728 and National Science Foundation of China (NSFC) Award No. 61772036 and 61331011 and partially supported by the Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology) and the Computer Science Department at CUNY Graduate Center as well as CUNY Hunter College.

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.

James Cross and Liang Huang. 2019. Span-based constituency parser.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017a. A transition-based directed acyclic graph parser for ucca. In *Proc. of ACL*, pages 1127–1138.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017b. A transition-based directed acyclic graph parser for ucca. *arXiv preprint arXiv:1704.00552*.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018. Multitask parsing across semantic representations. *arXiv preprint arXiv:1805.00287*.

Daniel Hershcovich, Zohar Aizenbud, Leshem Choshen, Elior Sulem, Ari Rappoport, and Omri Abend. 2019. Semeval 2019 task 1: Cross-lingual semantic parsing with ucca. *arXiv preprint arXiv:1903.02953*.

Abdul Khan, Subhadarshi Panda, Jia Xu, and Lampros Flokas. 2018. Hunter nmt system for wmt18 biomedical translation task: Transfer learning in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 655–661.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Nikita Kitaev and Dan Klein. 2018a. Constituency parsing with a self-attentive encoder. *arXiv preprint arXiv:1805.01052*.

Nikita Kitaev and Dan Klein. 2018b. Multilingual constituency parsing with self-attention and pre-training. *arXiv preprint arXiv:1812.11760*.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*.