Discourse Representation Structure Parsing with Recurrent Neural Networks and the Transformer Model

Jiangming Liu    Shay B. Cohen    Mirella Lapata
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB
jiangming.liu@ed.ac.uk, {scohen,mlap}@inf.ed.ac.uk

Abstract
We describe the systems we developed for Discourse Representation Structure (DRS) parsing as part of the IWCS-2019 Shared Task of DRS Parsing.1 Our systems are based on sequence-to-sequence modeling. To implement our model, we use the open-source neural machine translation system implemented in PyTorch, OpenNMT-py. We experimented with a variety of encoder-decoder models based on recurrent neural networks and the Transformer model. We conduct experiments on the standard benchmark of the Parallel Meaning Bank (PMB 2.2.0). Our best system achieves a score of 84.8% F1 in the DRS parsing shared task.

1 Introduction
Discourse Representation Theory is a popular theory of meaning representation designed to account for a variety of linguistic phenomena, including the interpretation of pronouns and temporal expressions within and across sentences (Kamp and Reyle, 1993). The Groningen Meaning Bank (GMB; Bos et al. 2017) provides a large collection of English texts annotated with Discourse Representation Structures (DRS), while the Parallel Meaning Bank (PMB; Abzianidze et al. 2017) provides DRSs in English, German, Italian and Dutch. Furthermore, the PMB introduces clause representation, as shown on the top of Figure 1.

With the recent introduction of neural network learning to the Natural Language Processing community, several neural DRS parsers have been developed for the problem of DRS parsing, i.e. the problem of taking a document or a sentence as input, and outputting their corresponding DRS. Liu et al. (2018) convert box-style DRSs to tree-style DRSs and propose the three-step tree DRS parser on the GMB, while van Noord et al. (2018) adopt a neural machine translation approach to parse sentences to their clause-style DRSs on PMB. Due to the different standard of annotations between GMB and PMB, and that the IWCS-2019 Shared Task of DRS Parsing mainly focuses on averagely short sentences in PMB annotations, our systems take sentences as input and output a clause-style DRS of PMB represented as a sequence for the IWCS-2018 Shared Task of DRS parsing (Abzianidze et al., 2019).

2 The Parsing System
Figure 2 shows the data pipeline in our system for both training and parsing. There are three main parts: (a) The component Preprocess, which prepares the input data to make it suitable for training and parsing models; (b) The component Neural Model which is based on OpenNMT; (c) The component Postprocess which contains some rules to ensure the system output is a well-formed DRSs.

1https://competitions.codalab.org/competitions/20220
2.1 Preprocessing

The Preprocess step works on the sentences and their DRSs of the training data and on the sentences of the development and the test data. We tried two levels of preprocessing, character-level and word-level.

Character Level We use the scripts of van Noord et al. (2018) to perform character-level preprocessing for sentences and their DRSs. Each sentence is separated into characters where a special symbol “|||” is used to mark a word boundary. The clauses are represented as a character sequence, except for the semantic roles, DRS operators and deictic constants, as shown in Figure 3(a). For example, “b1 REF e1” is preprocessed to “$NEW ||| REF”, which means that a new box (b1) is introduced by the box; “b2 ground_floor “n.01” x1” is preprocessed to “$0 ||| ground_floor ||| “ n . 0 1 ” ||| @0”, which means that the sense ground_floor.n.01 is constructed and then assigned to the referent @0, which is latest introduced, where @n (n∈ Z) denotes the referent |n|th latest introduced. Similarly, $n (n∈ Z) denotes the box |n|th latest constructed.

Word Level Each sentence is tokenized using the Moses script and then transformed to its lowercase form. Clauses are represented as sequences without changing the order, where a special symbol “||” is used to start a new clause. We rule out quotation marks in clauses (e.g. “‘tom’” is converted to tom) and

---

2Here, sentences are not further tokenized than they are in the data, and a token could be I’m or floor.
3if n is positive, @n denotes the referent is nth latest introduced in future.
4https://github.com/moses-smt/mosesdecoder
sentence: i live on the ground floor.

DRS: $NEW REF $0 REF $0 EQU @0 "now" $0 Theme @-1 "speaker" $0 Time @0 @-1 $0 live v.01 "n.08" @0 $0 Location @-1 @1 $NEW REF $0 ground_floor n.01 @0

(a) character level

sentence: i live on the ground floor.

DRS: $NEW REF $0 REF $0 EQU @0 Theme @-1 speaker $0 Time @0 @-1 $0 live v.01 @-1 $0 time n.08 @0 Location @-1 @1 $NEW REF $0 ground_floor n.01 @0

(b) word level

Figure 3: An example of preprocessing in character level and word level, respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>layers</td>
<td>2</td>
<td>batch_size</td>
<td>12</td>
<td>global_attention</td>
<td>general</td>
</tr>
<tr>
<td>rnn_size</td>
<td>300</td>
<td>batch_type</td>
<td>sents</td>
<td>copy_attention</td>
<td>True</td>
</tr>
<tr>
<td>rnn_type</td>
<td>LSTM</td>
<td>optim</td>
<td>sgd</td>
<td>copy_attn_type</td>
<td>dot</td>
</tr>
<tr>
<td>dropout</td>
<td>0.2</td>
<td>learning_rate</td>
<td>0.7</td>
<td>start_decay_steps</td>
<td>5000</td>
</tr>
<tr>
<td>bridge</td>
<td>True</td>
<td>learning_rate_decay</td>
<td>0.7</td>
<td>decay_steps</td>
<td>1000</td>
</tr>
<tr>
<td>encoder_type</td>
<td>brnn</td>
<td>max_grad_norm</td>
<td>5</td>
<td>decoder_type</td>
<td>rnn</td>
</tr>
</tbody>
</table>

LSTM

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>layers</td>
<td>6</td>
<td>batch_size</td>
<td>512</td>
<td>encoder_type</td>
<td>transformer</td>
</tr>
<tr>
<td>rnn_size</td>
<td>300</td>
<td>batch_type</td>
<td>tokens</td>
<td>decoder_type</td>
<td>transformer</td>
</tr>
<tr>
<td>transformer_ff</td>
<td>2048</td>
<td>optim</td>
<td>adam</td>
<td>position_encoding</td>
<td>True</td>
</tr>
<tr>
<td>heads</td>
<td>6</td>
<td>learning_rate</td>
<td>0.001</td>
<td>copy_attn_type</td>
<td>dot</td>
</tr>
<tr>
<td>dropout</td>
<td>0.2</td>
<td>global_attention</td>
<td>general</td>
<td>max_grad_norm</td>
<td>5</td>
</tr>
<tr>
<td>bridge</td>
<td>True</td>
<td>copy_attention</td>
<td>True</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transformer

Table 1: Choice of hyperparameters for our neural network models.

remain them case-sensitive. Following previous work (van Noord et al., 2018), the indices of variables in clauses are relative, as shown in Figure 3(b), which is the same to the character-level preprocessing.

2.2 Neural Models

We adopt Recurrent Neural Networks (RNNs) equipped with Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber 1997) units and the Transformer model (Vaswani et al., 2017) as our neural models. For the model implementation, we use the one provided by the OpenNMT-py toolkit (Klein et al., 2017). The hyperparameters we used are shown in Table 1 which are institutionally set without optimization.

Fine-tuning We propose a fine-tuning approach to enable the system to effectively use more training data in various quality, i.e. bronze and silver data. The fine-tuning approach allows the system train to convergence on one dataset (e.g. silver and gold data) and then continues to train to convergence on another dataset (e.g. gold data), where the optimizers are reset.
<table>
<thead>
<tr>
<th>LSTMs</th>
<th>character P</th>
<th>R</th>
<th>F1</th>
<th>time(h)</th>
<th>word P</th>
<th>R</th>
<th>F1</th>
<th>time(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg-data</td>
<td>73.91</td>
<td>75.00</td>
<td>74.45</td>
<td>13.1</td>
<td>73.81</td>
<td>73.75</td>
<td>73.78</td>
<td>7.8</td>
</tr>
<tr>
<td>sg-data + g-data</td>
<td>86.05</td>
<td>84.78</td>
<td>85.41</td>
<td>+2.0</td>
<td>84.80</td>
<td>82.83</td>
<td>83.80</td>
<td>+0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transformers</th>
<th>character P</th>
<th>R</th>
<th>F1</th>
<th>time(h)</th>
<th>word P</th>
<th>R</th>
<th>F1</th>
<th>time(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg-data</td>
<td>69.11</td>
<td>69.93</td>
<td>69.52</td>
<td>5.2</td>
<td>75.41</td>
<td>75.36</td>
<td>75.38</td>
<td>5.1</td>
</tr>
<tr>
<td>sg-data + g-data</td>
<td>82.32</td>
<td>81.19</td>
<td>81.75</td>
<td>+0.6</td>
<td>85.76</td>
<td>84.45</td>
<td>85.10</td>
<td>+0.6</td>
</tr>
</tbody>
</table>

Table 2: Results on test partition of the Parallel Meaning Bank.

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>R</th>
<th>F1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bsg-data</td>
<td>74.27</td>
<td>75.78</td>
<td>75.02</td>
<td></td>
</tr>
<tr>
<td>bsg-data + sg-data</td>
<td>77.74</td>
<td>78.78</td>
<td>78.26</td>
<td></td>
</tr>
<tr>
<td>bsg-data + g-data</td>
<td>86.98</td>
<td>86.55</td>
<td>86.76</td>
<td></td>
</tr>
<tr>
<td>bsg-data + sg-data + g-data</td>
<td>87.04</td>
<td>87.17</td>
<td>87.10</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Results on test dataset by word transformer

2.3 Postprocessing and Evaluation

We adopt the postprocessing scripts of van Noord et al. (2018) to transform back the output of our models to the clause format, and then use COUNTER (van Noord et al., 2018) as our evaluation metric.

3 Experiments

In this section, we introduce the training data that we used and the results on the PMB benchmarks.

3.1 Data

The training data consists of all of the bronze data (bronze), all of the silver data (silver), and the training section of the gold data (gold). All data is preprocessed. We mix bronze, silver and gold as bsg-data, and mix silver and gold as sg-data, and name the training section of gold data as g-data. Meanwhile, we adopt GloVe (Pennington et al., 2014) pre-trained word embeddings5 to initialize the representation of input tokens.

3.2 Results

Table 2 shows the results on test data, where sg-data means that the models are only trained on sg-data, and + g-data means that the models are continually fine-tuned on g-data. With LSTM, the character model performs marginally better than the word model. However, with Transformer, the word model performs significantly better than the character model. With both LSTM and Transformer, fine-tuning on g-data significantly improves the performance. Although the character LSTM is marginally better than the word Transformer, we still prefer the word Transformer as our final model, because it could be trained faster.

Table 3 shows the improved results on test dataset by using word Transformer with bronze data, where bsg-data means that the model is only trained on bsg-data, + sg-data means that the model is continually fine-tuned on sg-data, and + g-data means that the model is further fine-tuned on g-data. As shown in Tables 2 and 3, the improvement gap of fine-tuning on sg-data from bsg-data (3.24% F1) is narrower than that of fine-tuning on g-data from sg-data (8.84% F1). Fine-tuning on g-data may be the key to improve the performance on the test dataset. We believe this is due to the high similarity between

5https://nlp.stanford.edu/projects/glove/
Table 4: $F_1$-scores of fine-grained evaluation on test dataset.

g-data and the test data. Also, we discover that the model trained on bsg-data then fine-tuned on g-data can also have good performance, but slightly worse than the final models.

We submitted the word Transformer on bsg-data + sg-data + g-data as our final model to the DRS parsing shared task. On the test dataset of the shared task, our model achieves 84.80 $F_1$ score.

3.3 Analysis

We further analyze the output of the parsers trained on sg-data + g-data to see what components of the meaning representation are challenging. Table 4 shows the detailed results of Counter, where DRS Operators (e.g. negation), Roles (e.g. Agent), Concepts (i.e. predicates), synsets (e.g. “n.01”) are scored separately.

We compare four parsing models, LSTM with character-level preprocessing (char-LSTM), LSTM with word-level preprocessing (word-LSTM), Transformer with character-level preprocessing (char-transformer) and Transformer with word-level preprocessing (word-transformer). The char-LSTM and word-transformer models both achieve good performance, where word-transformer performs best on the construction of DRS operators, Concepts, Synsets-Noun and Synsets-Adjectives, and char-LSTM performs best on construction of Roles and Synsets-Verbs. The word-LSTM model is mediocre, but significantly outperforms the other models on the construction of Synsets-Adverbs with a large gap of average 35.14% $F_1$ score.

4 Conclusions

In this paper, we describe the system for the IWCS-2019 Shared Task of DRS parsing. We found that the character-level LSTM and the word-level transformer are competitive in the task. The training time of LSTM models increases as input sequences are longer, while training time are not sensitive to the lengths of input sequences in transformer. The output of LSTM models and transformers have different error distributions. There is still a large improvement space for the sequential models.

Acknowledgments

We thank Hessel Haagsma, Lasha Abzianidze, Rik van Noord and Johan Bos for their release of the latest version of PMB (2.2.0). We gratefully acknowledge the support of the European Research Council (Lapata, Liu; award number 681760), the EU H2020 project SUMMA (Cohen, Liu; grant agreement 688139) and Huawei Technologies (Cohen, Liu).

References


