Addressing the Data Sparsity Issue in Neural AMR Parsing

Xiaochang Peng[*1], **Chuan Wang**[*2], **Daniel Gildea**[1] and **Nianwen Xue**[2]
[1]Department of Computer Science, University of Rochester, Rochester, NY 14627
[2]Computer Science Department, Brandeis University, Waltham, MA 02453

Abstract

Neural attention models have achieved great success in different NLP tasks. However, they have not fulfilled their promise on the AMR parsing task due to the data sparsity issue. In this paper, we describe a sequence-to-sequence model for AMR parsing and present different ways to tackle the data sparsity problem. We show that our methods achieve significant improvement over a baseline neural attention model and our results are also competitive against state-of-the-art systems that do not use extra linguistic resources.

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic formalism where the meaning of a sentence is encoded as a rooted, directed graph. Figure 1 shows an example of an AMR in which the nodes represent the AMR concepts and the edges represent the relations between the concepts they connect. AMR concepts consist of predicate senses, named entity annotations, and in some cases, simply lemmas of English words. AMR relations consist of core semantic roles drawn from the Propbank (Palmer et al., 2005) as well as very fine-grained semantic relations defined specifically for AMR. These properties render the AMR representation useful in applications like question answering and semantics-based machine translation.

The task of AMR graph parsing is to map natural language strings to AMR semantic graphs. Recently, a sizable new corpus of English/AMR pairs (LDC2015E86) has been released. Different parsers have been developed to tackle this problem (Flanigan et al., 2014; Wang et al., 2015b; Artzi et al., 2015; Pust et al., 2015; Peng et al., 2015). Most of these parsers have used external resources such as dependency parses, semantic lexicons, etc., to tackle the sparsity issue.

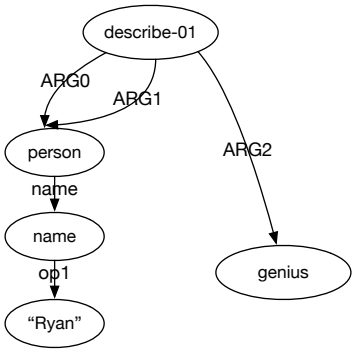

Figure 1: An example of AMR graph representing the meaning of: "Ryan's description of himself: a genius."

Recently, Sutskever et al. (2014) introduced a neural network model for solving the general sequence-to-sequence problem, and Bahdanau et al. (2014) proposed a related model with an attention mechanism that is capable of handling long sequences. Both models achieve state-of-the-art results on large scale machine translation tasks.

However, sequence-to-sequence models mostly work well for large scale parallel data, usually involving millions of sentence pairs. Vinyals et al. (2015) present a method which linearizes parse trees into a sequence structure and therefore a sequence-to-sequence model can be applied to the constituent parsing task. Competitive results have been achieved with an attention model on the Penn Treebank dataset, with only 40K annotated sentences.

AMR parsing is a much harder task in that the target vocabulary size is much larger, while the size of the dataset is much smaller. While for constituent parsing we only need to predict non-

*Both authors contribute equally.

terminal labels and the output vocabulary is limited to 128 symbols, AMR parsing has both concepts and relation labels, and the target vocabulary size consists of tens of thousands of symbols. Barzdins and Gosko (2016) applied a similar approach where AMR graphs are linearized using depth-first search and both concepts and relations are treated as tokens (see Figure 3). Due to the data sparsity issue, their AMR parsing results are significantly lower than state-of-the-art models when using the neural attention model.

In this paper, we present a method which linearizes AMR graphs in a way that captures the interaction of concepts and relations. To overcome the data sparsity issue for the target vocabulary, we propose a categorization strategy which first maps low frequency concepts and entity subgraphs to a reduced set of category types. In order to map each type to its corresponding target side concepts, we use heuristic alignments to connect source side spans and target side concepts or subgraphs. During decoding, we use the mapping dictionary learned from the training data or heuristic rules for certain types to map the target types to their corresponding translation as a postprocessing procedure.

Experiments show that our linearization strategy and categorization method are effective for the AMR parsing task. Our model improves significantly in comparison with the previously reported sequence-to-sequence results and provides a competitive benchmark in comparison with state-of-the-art results without using dependency parses or other external semantic resources.

2 Sequence-to-sequence Parsing Model

Our model is based on an existing sequence-to-sequence parsing model (Vinyals et al., 2015), which is similar to models used in neural machine translation.

2.1 Encoder-Decoder

Encoder. The encoder learns a context-aware representation for each position of the input sequence by mapping the inputs $w_1, \ldots, w_m$ into a sequence of hidden layers $h_1, \ldots, h_m$. To model the left and right contexts of each input position, we use a bidirectional LSTM (Bahdanau et al., 2014). First, each input's word embedding representation $x_1, \ldots, x_m$ is obtained though a lookup table. Then these embeddings serve as the input to two RNNs: a forward RNN and a backward RNN. The forward RNN can be seen as a recurrent function defined as follows:

$$h_i^{fw} = f(x_i, h_{i-1}^{fw}) \tag{1}$$

Here the recurrent function f we use is Long-Short-Term-Memory (LSTM) (Hochreiter and Schmidhuber, 1997). The backward RNN works similarly by repeating the process in reverse order. The outputs of forward RNN and backward RNN are then depth-concatenated to get the final representation of the input sequence.

$$h_i = [h_i^{fw}, h_{m-i+1}^{bw}] \tag{2}$$

Decoder. The decoder is also an LSTM model which generates the hidden layers recurrently. Additionally, it utilizes an attention mechanism to put a "focus" on the input sequence. At each output time step j, the attention vector $d_j^{'}$ is defined as a weighted sum of the input hidden layers, where the masking weight α_i^j is calculated using a feed-forward neural network. Formally, the attention vector is defined as follows:

$$u_i^j = v^T \tanh(W_1 h_i + W_2 d_j) \tag{3}$$

$$\alpha_i^j = \mathrm{softmax}(u_i^j) \tag{4}$$

$$d_j^{'} = \sum_{i=1}^{m} \alpha_i^j h_i \tag{5}$$

where d_j is the output hidden layer at time step j, and v, W_1, and W_2 are parameters for the model. Here the weight vector $\alpha_1^j, \ldots, \alpha_m^j$ is also interpreted as a soft alignment in the neural machine translation model, which similarly could also be treated as a soft alignment between token sequences and AMR relation/concept sequences in the AMR parsing task. Finally, we concatenate the hidden layer d_j and attention vector $d_j^{'}$ to get the new hidden layer, which is used to predict the output sequence label.

$$P(y_j|w, y_{1:j-1}) = \mathrm{softmax}(W_3[d_j, d_j^{'}]) \tag{6}$$

2.2 Parse Tree as Target Sequence

Vinyals et al. (2015) designed a reversible way of converting the parse tree into a sequence, which they call linearization. The linearization is performed in the depth-first traversal order. Figure 2 shows an example of the linearization result. The target vocabulary consists of 128 symbols.

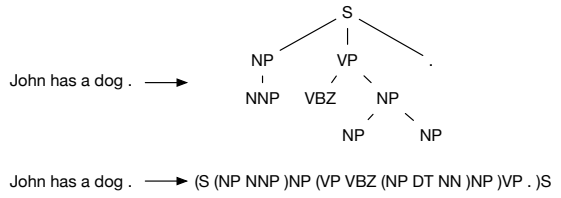

Figure 2: Example parsing task and its linearization.

In practice, they found that using the attention model is more data efficient and works well on the parsing task. They also reversed the input sentence and normalized the part-of-speech tags. After decoding, the output parse tree is recovered from the output sequence of the decoder in a post-processing procedure. Overall, the sequence-to-sequence model is able to match the performance of the Berkeley Parser (Petrov et al., 2006).

3 AMR Linearization

Barzdins and Gosko (2016) present a similar linearization procedure where the depth-first traversal result of an AMR graph is used as the AMR sequence (see Figure 3). The bracketing structure of AMR is hard to maintain because the prediction of relation (with left parenthesis) and the prediction of an isolated right parenthesis are not correlated. As a result, the output AMR sequences usually have parentheses that do not match.

We present a linearization strategy which captures the bracketing structure of AMR and the connection between relations and concepts. Figure 3 shows the linearization result of the AMR graph shown in Figure 1. Each relation connects the head concept to a subgraph structure rooted at the tail concept, which shows one branch below the head concept. We use the relation label and left parenthesis to show the beginning of the branch (subgraph) and use right parenthesis paired with the relation label to show the end of the branch. We additionally add "-TOP-(" at the beginning to show the start of the traversal of the AMR graph and add ")-TOP-" at the end to show the end of traversal. When a symbol is revisited, we replace the symbol with "-RET-". We additionally add the revisited symbol before "-RET-" to decide where the reentrancy is introduced to.[1] We also get rid of

[1]This is an approximation because one concept can appear multiple times, and we simply attach the reentrancy to the most recent appearance of the concept. An additional index would be needed to identify the accurate place of reentrancy.

Barzdins and Gosko (2016)

(describe-01 :ARG0 (person :name (name :op1 "Ryan"))
:ARG1 (person) :ARG2 genius)

Our linearization

-TOP-(describe-01 ARG0(person name(name op1("Ryan")op1)name)ARG0 ARG1(person -RET-)ARG1 ARG2(genius)ARG2)-TOP-

Figure 3: Comparison of AMR linearization

variables and only keep the full concept label. For example, "g / genius" to "genius".

We can easily recover the original AMR graph from its linearized sequence. The sequence also captures the branching information of each relation explicitly by representing it with a start symbol and an end symbol specific to that relation. During our experiments, most of the output sequences have a matching bracketing structure using this linearization strategy. The idea of linearization is basically a depth-first traversal of the AMR where the original graph structure can be reconstructed with the linearization result. Even though we call it a sequence, its core idea is actually generating a graph structure from top-down.

4 Dealing with the Data Sparsity Issue

While sequence-to-sequence models can be successfully applied to constituent parsing, they do not work well on the AMR parsing task as shown by Barzdins and Gosko (2016). The main bottleneck is that the size of target vocabulary for AMR parsing is much larger than constituent parsing, tens of thousands in comparison with 128, and the size of training data is less than half of that available for parsing.

In this section, we present a categorization method which significantly reduces the target vocabulary size, as the alignment from the attention model does not work well on the relatively small dataset. To adjust for the alignment errors made by the attention model, we propose to add supervision from an alignment produced by an external aligner which can use lexical information to overcome the limit of data size.

4.1 AMR Categorization

We define several types of categories and map low frequency words into these categories.

1. DATE: we reduce all the date entity subgraphs to this category, ignoring details of the specific date entity.

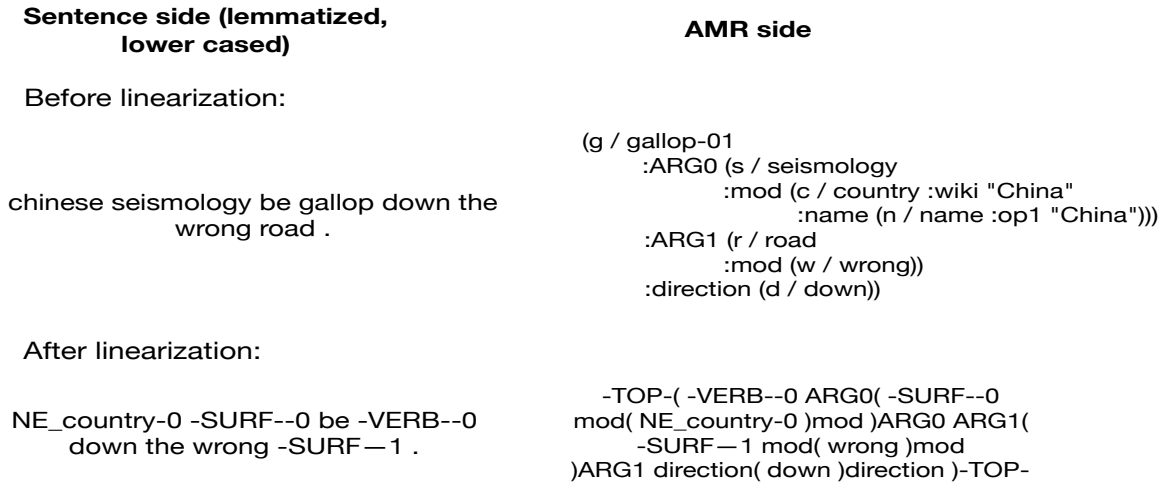

Figure 4: An example of categorized sentence-AMR pair.

2. NE_{*ent*}: we reduce all named entity subgraphs to this category, where *ent* is the root label of each subgraph, such as *country* or *person*.

3. -VERB-: we map predicate variables with low frequency ($n < 50$) to this category

4. -SURF-: we map non-predicate variables with low frequency ($n < 50$) to this category

5. -CONST-: we map constants other than numbers, "-", "interrogative", "expressive", "imperative" to this category.

6. -RET-: we map all revisited concepts to this category.

7. -VERBAL-: we additionally use the verbalization list [2] from the AMR website and map matched subgraphs to this category.

After the re-categorization, the vocabulary size is substantially reduced to around 2000, though this vocabulary size is still very large for the relatively small dataset. These categories and the frequent concepts amount to more than 90% of all the target words, and each of these are learned with a larger number of occurrences.

4.2 Categorize Source Sequence

The source side tokens also have sparsity issues. For example, even if we have mapped the number *1997* to "DATE", we can not easily generalize it to the token *1993* if it does not appear in the training data. Also, some special 6-digit date formats such as "YYMMDD" are hard to learn using co-occurrence statistics.

Our basic approach to dealing with this issue is to generalize these sparse tokens or spans to some special categories (currently we use the same set of categories defined in the previous section). On the training data, we can use the heuristic alignment. For example, if we learned from the heuristic alignment that "010911" is aligned to a date-entity of September 11, 2001 on the AMR side, we use the same category "DATE" to replace this token. We distinguish this alignment from other date alignments by assigning a unique indexed category "DATE-X" to both sides of the alignment, where "X" counts from 0 and adds one for each new date entity from left to right on the sentence side. The same index strategy goes for all the other categories. Figure 4 shows an example of the linearized parallel sequence. The first infrequent non-predicate variable "seismology" is mapped to "-SURF–0", then "wrong" to "-SURF–1" based on its position on the sentence side. The indexed category labels are then projected onto the target side based on the heuristic alignment. During this re-categorization procedure, we build a map Q from each token or span to its most likely concept or category on the target side based on relative frequency. We also dump a DATE template for recognizing new date entities by abstracting away specific date fields such as "1997" to "YEAR", "September" to "MONTH". For example, we build a template "MONTH DAY, YEAR"

[2] http://amr.isi.edu/download/lists/verbalization-list-v1.06.txt

from the specific date mention "June 6, 2007".

During decoding, we are only given the sentence. We first use the date templates learned from the training data to recognize dates in each sentence. We also use a named entity tagger to recognize named entity mentions in the sentence. We use the entity name and wiki information from Q if there is a match of the entity mention, otherwise for convenience we simply use "person" as the entity name and use wiki "-". For each of the other tokens, we first look it up in Q and replace it with the most likely mapping. If there is no match, we further look it up in the verbalization list. In case there is still no match, we use the part of speech information to assign its category. We replace verbs with category "-VERB-" and nouns with category "-SURF-". In accordance with the categorized token sequence, we also index each category in the resulting sequence from left to right.

4.3 Recovering AMR graph

During decoding, our output sequences usually have categories and we need to map each category to AMR concepts or subgraphs. When we categorize the tokens in each sentence before decoding, we save the mapping from each category to its original token as table D. As we use the same set of categories on both source and target sides, we heuristically align target side category label to its source side counterpart from left to right. Given table D, we know which source side token it comes from and use the most frequent concept or subgraph of the token to replace the category.

4.4 Supervised Attention Model

In this section, we propose to learn the attention vector in a supervised manner. There are two main motivations behind this. First, the neural attention model usually utilizes millions of data points to train the model, which learns a quite reasonable attention vector that at each output time step constrains the decoder to put a focus on the input sequences (Bahdanau et al., 2014; Vinyals et al., 2015). However, we only have 16k sentences in the AMR training data and our output vocabulary size is quite large, which makes it hard for the model to learn a useful alignment between the input sequence and AMR concepts/relations. Second, as argued by Liu et al. (2016), the sequence-to-sequence model tries to calculate the attention vector and predict the current output label simultaneously. This makes it impossible for the learned soft alignment to combine information from the whole output sentence context. However, traditional word alignment can easily use the whole output sequence, which will produce a more informed alignment.

Similar to the method used by Liu et al. (2016), we add an additional loss to the original objective function to model the disagreement between the reference alignment and the soft alignment produced by the attention mechanism. Formally, for each input/output sequence pair $(\mathbf{w}, \mathbf{y})$ in the training set, the objective function is defined as:

$$-\frac{1}{n}\sum_{j=1}^{n}\log p(y_j|\mathbf{w}, y_{1:j-1}) + \lambda\Theta(\bar{\alpha}^j, \alpha^j) \quad (7)$$

where $\bar{\alpha}^j$ is the reference alignment for output position j, which is provided by the existing aligner, α^j is the soft alignment, $\Theta()$ is cross-entropy function, n is the length of output sequence and λ is the hyperparameter which serves as a trade-off between sequence prediction and alignment supervision. Note that the supervised attention part doesn't affect the decoder which will predict the output label given learned weights.

One issue with this method is how we represent $\bar{\alpha}$. As the output of conventional aligner is a hard decision, alignment is either one or zero for each input position. In addition, multiple input words could be aligned to one single concept. Finally, in the AMR sequences, there are many output labels (mostly relations) which don't align to any word in the input sentence. We utilize a heuristic method to process the reference alignment. We assign an equal probability among the words that are aligned to one AMR concept. Then if the output label doesn't align to any input word, we assign an even probability for every input word.

5 Experiments

We evaluate our system on the released dataset (LDC2015E86) for SemEval 2016 task 8 on meaning representation parsing (May, 2016). The dataset contains 16,833 training, 1,368 development and 1,371 test sentences which mainly cover domains like newswire, discussion forum, etc. All parsing results are measured by Smatch (version 2.0.2) (Cai and Knight, 2013).

5.1 Experiment Settings

We first preprocess the input sentences with tokenization and lemmatization. Then we extract

the named entities using the Illinois Named Entity Tagger (Ratinov and Roth, 2009).

For training all the neural AMR parsing systems, we use 256 for both hidden layer size and word embedding size. Stochastic gradient descent is used to optimize the cross-entropy loss function and we set the drop out rate to be 0.5. We train our model for 150 epochs with initial learning rate of 0.5 and learning rate decay factor 0.95 if the model doesn't improve for the 3 last epochs.

5.2 Baseline Model

Our baseline model is a plain sequence-to-sequence model which has been used in the constituent parsing task (Vinyals et al., 2015). While they use a 3-layer deep LSTM, we only use a single-layer LSTM for both encoder and decoder since our data is relatively small and empirical comparison shows that stacking more layers doesn't help in our case. AMR linearization follows Section 3 without categorization. Since we don't restrict the input/output vocabulary in this case, our vocabulary size is quite large: 10,886 for output vocabulary and 2,2892 for input vocabulary. We set them to 10,000 and 20,000 respectively and replace the out of vocabulary words with `_UNK_`.

5.3 Impact of Re-Categorization

We first inspect the influence of utilizing categorization on the input and output sequence. Table 1 shows the Smatch evaluation score on development set.

System	P	R	F
Baseline	0.42	0.34	0.37
Re-Categorization ($n = 50$)	0.55	0.46	0.50

Table 1: Re-Categorization impact on development set

We see from the table that re-categorization improves the F-score by 13 points on the development set. As mentioned in section 4.1, by setting the low frequency threshold n to 50 and re-categorizing them into a reduced set of types, we now reduce the input/output vocabulary size to (2,000, 6,000). This greatly reduces the label sparsity and enables the neural attention model to learn a better representation on this small scale data. Another advantage of this method is that although AMR tries to abstract away surface forms and retain the semantic meaning structure of the sentence, a large portion of the concepts are coming from the surface form and have exactly same string form both in input sentence and AMR graph. For example, `nation` in sentence is mapped to concept `(n / nation)` in the AMR. For the frequent concepts in the output sequence, since the model can observe many training instances, we assume that it can be predicted by the attention model. For the infrequent concepts, because of the categorization step, we only require the model to predict the concept type and its relative position in the graph. By applying the post-processing step mentioned in Section 4.3, we can easily recover the categorized concepts to their original form.

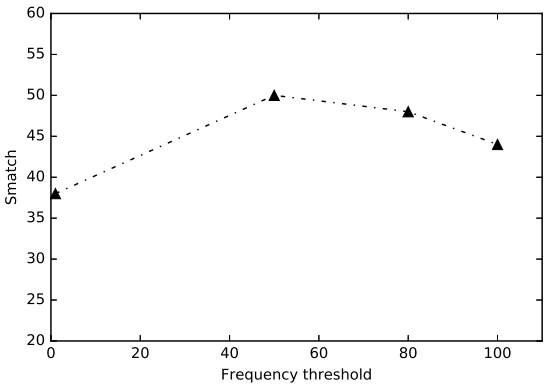

Figure 5: AMR parsing performance on development set given different categorization frequency.

We also inspect how the value of re-categorization frequency threshold n affects the AMR parsing result. As shown in Figure 5, setting n to 0, which means no output labels will be categorized into types -VERB- and -SURF-, doesn't improve the baseline system. The reason is that we still have a large output vocabulary size and training data is still sparse with respect to the low frequency output labels. Also, if we set n too high, although the output vocabulary size becomes smaller, some of the frequent output labels that the model handles well originally will be put into the coarse-grained types, losing information in the recovery process. Thus we can see from the plot that after the optimal point the Smatch score will drop. Therefore, we choose to set $n = 50$ in the subsequent experiments.

5.4 Impact of Supervised Alignment

Choice of External Aligner. There are two existing AMR aligners: one is a rule-based aligner coming with JAMR (Flanigan et al., 2014), which defines regular expression patterns to greedily match between AMR graph fragment and input token spans; another one is an unsupervised aligner (Pourdamghani et al., 2014) which adopts the traditional word alignment method in machine translation. Although evaluated on different set of manual alignment test sentences, both aligners achieved $\sim$90% F1 score. Here we choose to use the second aligner, as it covers broader domains.

Different alignment configurations To balance between the sequence learning and alignment agreement, We empirically tune the hyperparameter λ and set it to 0.3. For the external alignment we use for reference, we convert it to a vector with equal probability as discussed in Section 4.4. We then train a sequence-to-sequence model with re-categorized input/output and report the result on development set.

System	P	R	F
Baseline	0.42	0.34	0.37
Categorization ($n = 50$)	0.55	0.46	0.50
SuperAttn+Cate ($n = 50$)	0.56	0.49	0.52

Table 2: Supervised attention impact on development set

As shown in Table 2, the supervised attention model is able to further improve the Smatch score by another 2 points, which are mainly contributed by 3 points increase in recall. Since the reference/external alignment is mostly between the input tokens and AMR graph concepts, we believe that the supervised attention model is able to constrain the decoder so that it will output concepts which can be aligned to some tokens in the input sentence.

System	P	R	F
SuperAttn+Cate ($n = 50$)	0.56	0.49	0.52
NO-RELATION-ALIGN	0.46	0.40	0.43

Table 3: Supervised attention impact on development set

Because we have relations in the AMR graph, the alignment problem here is different from the word alignment in machine translation. To verify the effectiveness of our setup, we also compare our configuration to the condition **NO-RELATION-ALIGN** where we only ignore the alignment between sentence and AMR relations by putting an all zero vector as the reference attention for each output relation label. From Table 3 we see that simply ignoring the reference attention for relations would greatly affect the model performance, and how we effectively represent the reference alignment for relations is crucial for the supervised attention model.

5.5 Results

In this section we report our final result on the test set of SemEval 2016 Task 8 and compare our model with other parsers. We train our model utilizing re-categorization and supervised attention with hyperparameters tuned on the development set. Then we apply our trained model on the test set.

Firstly, we compare our model to the existing sequence-to-sequence AMR parsing model of Barzdins and Gosko (2016). As shown in table 4, the word-level model in Barzdins and Gosko (2016) is basically our baseline model. The second model they use is a character-based sequence-to-sequence model. Our model can also be regarded as a word-level model; however, by utilizing carefully designed categorization and supervised attention, our system outperforms both their results by a large margin.

System	P	R	F
Our system	**0.55**	**0.50**	**0.52**
Barzdins and Gosko (2016)†	-	-	0.37
Barzdins and Gosko (2016)*	-	-	0.43

Table 4: Compare to other sequence-to-sequence AMR parser. Barzdins and Gosko (2016)† is the word-level neural AMR parser, Barzdins and Gosko (2016)* is the character-level neural AMR parser.

Table 5 gives the comparison of our system to some of the teams participating in SemEval16 Task 8. Since a large portion of the teams extend on the state-of-the-art system CAMR (Wang et al., 2015b; Wang et al., 2015a; Wang et al., 2016), here we just pick typical teams that represent different approaches. We can see from the table that our system fails to outperform the state-

of-the-art system. However, the best performing system CAMR uses a dependency structure as a starting point, where dependency parsing has achieved high accuracy recently and can be trained on larger corpora. Also, it utilizes semantic role labeling and complex features, which makes the training process a long pipeline. Our system only needs minimal preprocessing, and doesn't need the dependency parsing step. Our approach is competitive with the SHRG (Synchronous Hyperedge Replacement Grammar) method of Peng et al. (2015), which does not require a dependency parser and uses SHRG to formalize the string-to-graph problem as a chart parsing task. However, they still need a concept identification stage, while our model can learn the concepts and relations jointly.

System	P	R	F
Our system	0.55	0.50	0.52
Peng and Gildea (2016)	0.56	0.55	0.55
CAMR	0.70	0.63	0.66

Table 5: Comparison to other AMR parsers.

6 Discussion

In this paper, we have proposed several methods to make the sequence-to-sequence model work competitively against conventional AMR parsing systems. Although we haven't outperformed state-of-the-art system using the conventional methods, our results show the effectiveness of our approaches to reduce the sparsity problem when training sequence-to-sequence model on a relatively small dataset. Our work could be aligned with the effort to handle low-resource data problems when building the end-to-end neural network model.

In neural machine translation, the attention model is traditionally trained on millions of sentence pairs, while facing low-resource language pairs, the neural MT system performance tends to downgrade (Zoph et al., 2016). There has been growing interest in tackling sparsity/low-resource problem in neural MT. Zoph et al. (2016) use a transfer learning method to first pre-train the neural model on rich-resource language pairs and then import the learned parameters to continue training on low-resource language pairs so that the model can alleviate the sparsity problem through shared parameters. Firat et al. (2016) builds a multilingual neural system where the attention mechanism can be shared between different language pairs. Our work could be seen as parallel efforts to handle the sparsity problem since we focus on the input/output categorization and external alignment, which are both handy for low-resource languages.

In this paper, we haven't used any syntactic parser. However, as shown in previous works (Flanigan et al., 2014; Wang et al., 2015b; Artzi et al., 2015; Pust et al., 2015), using dependency features helps improve the parsing performance significantly because of the linguistic similarity between the dependency tree and AMR structure. An interesting extension would be to use a linearized dependency tree as the source sequence and apply sequence-to-sequence to generate the AMR graph. Our parser could also benefit from the modeling techniques in Wu et al. (2016).

7 Conclusion

Neural attention models have achieved great success in different NLP tasks. However, they have not been as successful on AMR parsing due to the data sparsity issue. In this paper, we described a sequence-to-sequence model for AMR parsing and present different ways to tackle the data sparsity problems. We show that our methods have led to significant improvement over a baseline neural attention model, and our model is also competitive against models that do not use extra linguistic resources.

Acknowledgments Funded in part by a Google Faculty Award.

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal, September. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation

for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Guntis Barzdins and Didzis Gosko. 2016. Riga at semeval-2016 task 8: Impact of smatch extensions and character-level neural translation on AMR parsing accuracy. *arXiv preprint arXiv:1604.01278*.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria, August. Association for Computational Linguistics.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, San Diego, California, June. Association for Computational Linguistics.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Baltimore, Maryland, June. Association for Computational Linguistics.

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

L. Liu, M. Utiyama, A. Finch, and E. Sumita. 2016. Neural Machine Translation with Supervised Attention. *ArXiv e-prints*, September.

Jonathan May. 2016. Semeval-2016 task 8: Meaning representation parsing. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1063–1073, San Diego, California, June. Association for Computational Linguistics.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Xiaochang Peng and Daniel Gildea. 2016. UofR at semeval-2016 task 8: Learning synchronous hyperedge replacement grammar for AMR parsing. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1185–1189, San Diego, California, June. Association for Computational Linguistics.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for AMR parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 32–41, Beijing, China, July. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July. Association for Computational Linguistics.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English strings with abstract meaning representation graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 425–429, Doha, Qatar, October. Association for Computational Linguistics.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Parsing English into abstract meaning representation using syntax-based machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1143–1154, Lisbon, Portugal, September. Association for Computational Linguistics.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado, June. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 857–862, Beijing, China, July. Association for Computational Linguistics.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for AMR parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado, May–June. Association for Computational Linguistics.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. 2016. CAMR at semeval-2016 task 8: An extended transition-based AMR parser. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1173–1178, San Diego, California, June. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

B. Zoph, D. Yuret, J. May, and K. Knight. 2016. Transfer Learning for Low-Resource Neural Machine Translation. *ArXiv e-prints*, April.