Pushing the Limits of AMR Parsing with Self-Learning

Young-Suk Lee[*] **Ramón Fernandez Astudillo**[*] **Tahira Naseem**[*]
Revanth Gangi Reddy[*][†] **Radu Florian, Salim Roukos**
IBM Research
{ysuklee,tnaseem}@us.ibm.com ramon.astudillo@ibm.com

Abstract

Abstract Meaning Representation (AMR) parsing has experienced a notable growth in performance in the last two years, due both to the impact of transfer learning and the development of novel architectures specific to AMR. At the same time, self-learning techniques have helped push the performance boundaries of other natural language processing applications, such as machine translation or question answering. In this paper, we explore different ways in which trained models can be applied to improve AMR parsing performance, including generation of synthetic text and AMR annotations as well as refinement of actions oracle. We show that, without any additional human annotations, these techniques improve an already performant parser and achieve state-of-the-art results on AMR 1.0 and AMR 2.0.

1 Introduction

Abstract Meaning Representation (AMR) are broad-coverage sentence-level semantic representations expressing *who does what to whom*. Nodes in an AMR graph correspond to concepts such as entities or predicates and are not always directly related to words. Edges in AMR represent relations between concepts such as subject/object.

AMR has experienced unprecedented performance improvements in the last two years, partly due to the rise of pre-trained transformer models (Radford et al., 2019; Devlin et al., 2019; Liu et al., 2019), but also due to AMR-specific architecture improvements. A non-exhaustive list includes latent node-word alignments through learned permutations (Lyu and Titov, 2018a), minimum risk training via REINFORCE (Naseem et al., 2019), a sequence-to-graph modeling of linearized trees with copy mechanisms and re-entrance features (Zhang et al., 2019a) and more recently a highly performant graph-sequence iterative refinement model (Cai and Lam, 2020) and a hard-attention transition-based parser (F. A. et al., 2020), both based on the Transformer architecture.

Given the strong improvements in architectures for AMR, it becomes interesting to explore alternative avenues to push performance even further. AMR annotations are relatively expensive to produce and thus typical corpora have on the order of tens of thousands of sentences. In this work we explore the use self-learning techniques as a means to escape this limitation.

We explore the use of a trained parser to iteratively refine a rule-based AMR oracle (Ballesteros and Al-Onaizan, 2017; F. A. et al., 2020) to yield better action sequences. We also exploit the fact that a single AMR graph maps to multiple sentences in combination with AMR-to-text (Mager et al., 2020), to generate additional training samples without using external data. Finally we revisit silver data training (Konstas et al., 2017a). These techniques reach 77.3 and 80.7 Smatch (Cai and Knight, 2013) on AMR1.0 and AMR2.0 respectively using only gold data as well as 78.2 and 81.3 with silver data.

2 Baseline Parser and Setup

To test the proposed ideas, we used the AMR setup and parser from (F. A. et al., 2020) with improved embedding representations. This is a transition-based parsing approach, following the original AMR oracle in (Ballesteros and Al-Onaizan, 2017) and further improvements in (Naseem et al., 2019).

Briefly, rather than predicting a graph g from a sentence s directly, transition-based parsers predict instead an action sequence a. This action sequence, when applied to a state machine, produces the graph $g = M(a, s)$. This turns the problem of

[*] Equal contribution.
[†] Work done during AI Residency at IBM Research.

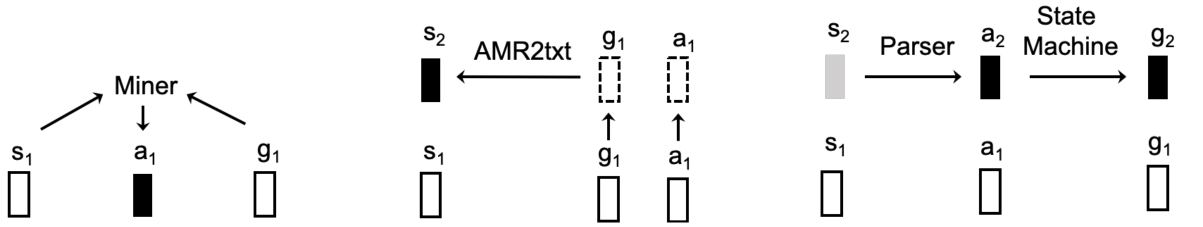

Figure 1: Role of sentence s, AMR graph g and oracle actions a in the different self-learning strategies. Left: Replacing rule-based actions by machine generated ones. Middle: synthetic text generation for existing graph annotations. Right: synthetic AMR generation for external data. Generated data (■). External data (■).

predicting the graph into a sequence to sequence problem, but introduces the need for an oracle to determine the action sequence $a = O(g, s)$. As in previous works, the oracle in (F. A. et al., 2020) is rule-based, relying on external word-to-node alignments (Flanigan et al., 2014; Pourdamghani et al., 2016) to determine action sequences. It however force-aligns unaligned nodes to suitable words, notably improving oracle performance.

As parser, (F. A. et al., 2020) introduces the stack-Transformer model. This is a modification of the sequence to sequence Transformer (Vaswani et al., 2017) to account for the parser state. It modifies the cross-attention mechanism dedicating two heads to attend the stack and buffer of the state machine $M(a, s)$. This parser is highly performant achieving the best results for a transition-based parser as of date and second overall for AMR2.0 and tied with the best for AMR1.0.

The stack-Transformer is trained as a conventional sequence to sequence model of $p(a \mid s)$ with a cross entropy loss. We used the full stack and full buffer setting from (F. A. et al., 2020) with same hyper-parameters for training and testing with the exception of the embeddings strategy detailed below. All models use checkpoint averaging (Junczys-Dowmunt et al., 2016) of the best 3 checkpoints and use a beam size of 10[1] while decoding. We refer to the original paper for exact details.

Unlike in the original work, we use RoBERTa-large, instead of RoBERTa-base embeddings, and we feed the average of all layers as input to the stack-Transformer. This considerably strengthens the baseline model from the original 76.3/79.5 for the AMR1.0/AMR2.0 development sets to 77.6/80.8 Smatch[2]. This baseline will be henceforth referred to as (F. A. et al., 2020) plus Strong Embeddings (+SE).

3 Oracle Self-Training

As explained in Section 2, transition-based parsers require an Oracle $a = O(g, s)$ to determine the action sequence producing the graph $g = M(a, s)$. Previous AMR oracles (Ballesteros and Al-Onaizan, 2017; Naseem et al., 2019; F. A. et al., 2020) are rule based and rely on external word-to-node alignments. Rule-based oracles for AMR are sub-optimal and they do not always recover the original graph. The oracle score for AMR 2.0, measured in Smatch, is 98.1 (F. A. et al., 2020) and 93.7 for (Naseem et al., 2019). In this work, we explore the idea of using a previously trained parser, $p(a \mid s)$ to improve upon an existing oracle, initially rule-based.

For each training sentence s with graph g and current oracle action sequence a^*, we first sample an action sequence $\tilde{a} \sim p(a \mid s)$. Both $\tilde{a}$ and a^* are run through the state machine $M()$ to get graphs $\tilde{g}$ and g^* respectively. We then replace a^* by $\tilde{a}$ if $\text{Smatch}(\tilde{g}, g) > \text{Smatch}(g^*, g)$ or $(\text{Smatch}(\tilde{g}, g) = \text{Smatch}(g^*, g)$ and $|\tilde{a}| < |a^*|)$. This procedure is guaranteed to either increase Smatch, shorten action length or leave it unaltered. The downside is that many samples have to be drawn in order to obtain a single new best action, we therefore refer to this method as *mining*.

Starting from the improved (F. A. et al., 2020), we performed 2 rounds of mining, stopping after less than 20 action sequences were obtained in a single epoch, which takes around 10 epochs[3]. Between rounds we trained a new model from scratch with the new oracle to improve mining. This led to 2.0% actions with better Smatch and 3.7% shorter length for AMR1.0 and 2.8% and 3.2% respectively for AMR2.0. This results in an improvement in oracle Smatch from 98.0 to 98.2 for AMR 1.0 and 98.1 to 98.3 for AMR 2.0.

[1] This increased scores at most 0.8/0.4 for AMR1.0/2.0.

[2] We used the latest version available, 1.0.4

[3] One round of mining takes around 20h, while normal model training takes 6h on a Tesla V100.

Table 1 shows that mining for AMR leads to an overall improvement of up to 0.2 Smatch across the two tasks with both shorter sequences and better Smatch increasing model performance when combined. Example inspection revealed that mining corrected oracle errors such as detached nodes due to wrong alignments. It should also be noted that such type of errors are much more present in previous oracles such as (Naseem et al., 2019) compared to (F. A. et al., 2020) and margins of improvement are therefore smaller.

Technique	AMR1.0	AMR2.0
(F. A. et al., 2020)+SE	77.6 ±0.1	80.8 ±0.1
< length ∪ > smatch	77.8 ±0.1	80.9 ±0.2

Table 1: Dev-set Smatch for AMR 1.0 and AMR 2.0 for different mining criteria. Average results for 3 seeds with standard deviation.

4 Self-Training with Synthetic Text

AMR abstracts away from the surface forms i.e. one AMR graph corresponds to many different valid sentences. The AMR training data, however, provides only one sentence per graph with minor exceptions. AMR 1.0 and AMR 2.0 training corpora have also only 10k and 36k sentences, respectively, making generalization difficult. We hypothesize that if the parser is exposed to allowable variations of text corresponding to each gold graph, it will learn to generalize better.

To this end, we utilize the recent state-of-the-art AMR-to-text system of Mager et al. (2020), a generative model based on fine-tuning of GPT-2 (Radford et al., 2019). We use the trained model $p(s \mid g)$ to produce sentences from gold AMR graphs. For each graph g in the training data, we generate 20 sentences via sampling $\tilde{s} \sim p(s \mid g)$ and one using the greedy best output. We then use the following cycle-consistency criterion to filter this data. We use the improved stack-Transformer parser in Table 1 to generate two AMR graphs: one from the generated text $\tilde{s}$, $\tilde{g}$ and one from the original text s, $\hat{g}$. We then use the Smatch between these two graphs to filter out samples, selecting up to three samples per sentence if their Smatch was not less than 80.0. We remove sentences identical to the original gold sentence or repeated. Filtering prunes roughly 90% of the generated sentences. This leaves us with 18k additional sentences for AMR 1.0 and 68k for AMR 2.0. Note that the use of parsed graph, rather than the gold graph, for filtering accounts for parser error and yielded better results as a filter.

Two separate GPT-2-based AMR-to-text systems were fine-tuned using AMR 1.0 and AMR 2.0 train sets and then sampled to generate the respective text data[4] and conventional training was carried out over the extended dataset. As shown in Table 2, synthetic text generation, henceforth denoted synTxt, improves parser performance over the (F. A. et al., 2020)+SE baseline for AMR2.0 and particularly for AMR1.0, possibly due to its smaller size.

Technique	AMR1.0	AMR2.0
(F. A. et al., 2020)+SE	77.6 ±0.1	80.8 ±0.1
synTxt	78.2 ±0.1	81.2 ±0.1

Table 2: Dev-set Smatch for AMR 1.0 and AMR 2.0. for synthetic text. Average results for 3 seeds with standard deviation.

5 Self-Training with Synthetic AMR

A trained parser can be used to parse unlabeled data and produce synthetic AMR graphs, henceforth synAMR. Although these graphs do not have the quality of human-annotated AMRs, they have been shown to improve AMR parsing performance (Konstas et al., 2017b; van Noord and Bos, 2017). The performance of prior works is however not any more comparable to current systems and it is therefore interesting to revisit this approach.

For this, we used the improved (F. A. et al., 2020) parser of Sec. 2 to parse unlabeled sentences from the context portion of SQuAD-2.0, comprising 85k sentences and 2.3m tokens, creating an initial synAMR corpus. This set is optionally filtered to reduce the training corpus size for AMR 2.0 experiments and is left unfiltered for AMR 1.0, due to its smaller size. The filtering combines two criteria. First, it is easy to detect when the transition-based system produces disconnected AMR graphs. Outputs with disconnected graphs are therefore filtered out. Second, we use a cycle-consistency criteria as in Section 4 whereby synthetic text is generated for each synthetic AMR with (Mager et al., 2020). For each pair of original text and generated text, the synAMR is filtered out if BLEU score (Papineni et al., 2002) is lower than a pre-specified

[4] synTxt training takes 17h for AMR 2.0 and 5h hours for AMR 1.0 on a Tesla V100. AMR-to-text training for 15 epochs takes 4.5h on AMR 1.0 and 15h on AMR 2.0.

threshold, 5 in our experiments. Because the AMR-to-text generation system is trained on the human-annotated AMR only, generation performance may be worse on synthetic AMR and out of domain data. Consequently we apply BLEU-based filtering only to the input texts with no out of vocabulary (OOV) tokens with respect to the original human-annotated corpus. After filtering, the synAMR data is reduced to 58k sentences.

Following prior work, we tested pre-training on synAMR only, as in (Konstas et al., 2017b), or on the mix of human-annotated AMR and synAMR, as in (van Noord and Bos, 2017) and then fine-tuned on the AMR1.0 or AMR2.0 corpora. Table 3 shows the results for AMR1.0 and AMR2.0 under the two pre-training options. Results show that pre-training on the mix of human-annotated AMR and synAMR works better than pre-training on synAMR only, for both AMR1.0 and AMR 2.0[5].

Technique	AMR1.0	AMR2.0
(F. A. et al., 2020)+SE	77.6 ±0.1	80.8 ±0.1
synAMR only	78.1±0.0	80.7 ±0.0
human+synAMR	78.6±0.1	81.6 ±0.0

Table 3: Dev-set Smatch for AMR1.0 and AMR2.0. for the baseline parser and synthetic AMR training. Average results for 3 seeds with standard deviation.

6 Detailed Analysis

6.1 Comparison Background

We compare the proposed methods with recent prior art in Table 4. Pre-trained embeddings are indicated as BERT base^{b} and large^{B} (Devlin et al., 2019), RoBERTa base^{r} and large^{R} (Liu et al., 2019). Note that RoBERTA large, particularly with layer average, can be expected to be more performant then BERT. Graph Recategorization is used in (Lyu and Titov, 2018b; Zhang et al., 2019b) and indicated as ^{G}. This is a pre-processing stage that segments text and graph to identify named entities and other relevant sub-graphs. It also removes senses and makes use of Stanford's CoreNLP to lemmatize input sentences and add POS tags. Graph recategorization also requires post-processing with Core-NLP at test time to reconstruct the graph. See (Zhang et al., 2019b, Sec. 6) for details.

Model	AMR1.0	AMR2.0
(Lyu and Titov, 2018b)^{G}	73.7	74.4
(Naseem et al., 2019)^{B}	-	75.5
(Zhang et al., 2019b) ^{B,G}	71.3	77.0
(F. A. et al., 2020) ^{r}	75.4 ±0.0	79.0 ±0.1
(Cai and Lam, 2020) ^{b}	74.0	78.7
(Cai and Lam, 2020) ^{b,G}	75.4	80.2
(F. A. et al., 2020)+SE^{R}	76.9 ±0.1	80.2 ±0.0
oracle mining	76.9 ±0.0	80.3 ±0.1
synTxt	77.3 ±0.2	80.7 ±0.2
synAMR^{U}	77.6 ±0.1	81.0 ±0.1
mining + synTxt	77.5 ±0.1	80.4 ±0.0
mining + synAMR^{U}	77.7 ±0.1	80.9 ±0.0
synTxt + synAMR^{U}	78.1 ±0.1	81.0 ±0.2
mining + synTxt + synAMR^{U}	**78.2 ±0.1**	**81.3 ±0.0**

Table 4: Test-set Smatch for AMR1.0 and AMR2.0

Both (Naseem et al., 2019; F. A. et al., 2020) use a similar transition-based AMR oracle, but (Naseem et al., 2019) uses stack-LSTM and Reinforcement Learning fine-tuning. These oracles require external alignments and a lemmatizer at train time, but only a lemmatizer at test time. It is important to underline that for the presented methods we do not use additional human annotations throughout the experiments and that the only external source of data is additional text data for synthetic AMR, which we indicate with ^{U}.

6.2 Results

As displayed in Table 4, the baseline system is close to the best published system with better results for AMR1.0 (+0.8) and worse for AMR2.0 (−0.5). Transition-based systems process the sentence from left to right and model the AMR graph only indirectly through its action history and the alignments of actions to word tokens. This can be expected to generate a strong inductive bias that helps in lower resource scenarios.

Regarding the introduced methods, mining shows close to no improvement in individual results. SynAMR provides the largest gain (0.7/0.8) for AMR1.0/AMR2.0 while synTxt provides close to half that gain (0.4/0.3). The combination of both methods also yields an improvement over their individual scores, but only for AMR1.0 with a 0.9 improvement. Combination of mining with synTxt and synAMR hurt results, however synTxt and synAMR does improve for AMR2.0 attaining a 1.1 improvement.

Overall, the proposed approach achieves 81.3 Smatch in AMR2.0 combining the three methods,

[5] human+synAMR and synAMR training take about 54h and 19h respectively for AMR2.0 and 17h and 13h respectively for AMR1.0. Fine-tuning takes 4h for AMR2.0 and 3h for AMR1.0 on a Tesla V100.

System	Smatch	Unlabeled	No WSD	Concepts	Named Ent.	Negations	Wikif.	Reentr.	SRL
(Cai and Lam, 2020) ^{b}	78.7	81.5	79.2	88.1	87.1	66.1	81.3	63.8	74.5
(Cai and Lam, 2020) ^{b,G}	80.2	82.8	80.8	88.1	81.1	**78.9**	**86.3**	64.6	74.2
(F. A. et al., 2020)+SE^{R}	80.2	84.2	80.7	88.1	87.5	64.5	78.8	70.3	78.2
oracle mining	80.3	84.2	79.0	87.8	87.7	65.4	79.0	70.4	78.2
synTxt	80.7	84.6	81.1	88.5	88.3	69.8	78.8	71.1	79.0
synAMR^{U}	81.0	85.0	81.5	88.6	88.5	65.4	79.0	71.1	79.0
mining+synTxt	80.4	84.5	80.9	87.9	87.7	65.8	79.3	70.5	78.5
mining+synAMR^{U}	80.9	84.9	81.4	88.4	88.0	66.0	79.3	70.9	78.9
synTxt+synAMR^{U}	81.0	84.9	81.5	88.6	88.3	67.4	78.9	71.5	79.1
mining+synTxt+synAMR^{U}	**81.3**	**85.3**	**81.8**	**88.7**	**88.7**	66.3	79.2	**71.9**	**79.4**

Table 5: Detailed scoring of the final system on AMR2.0 test sets

which is the best result obtained at the time of submission for AMR2.0, improving 1.1 over (Cai and Lam, 2020). It also obtains 78.2 for AMR1.0, which is 2.8 points above best previous results. Excluding silver data training, synTxt achieves 80.7 $(+0.5)$ in AMR2.0 and 77.5 $(+2.1)$ with minining in AMR1.0.

We also provide the detailed AMR analysis from (Damonte et al., 2017) for the best previously published system, baseline and the proposed methods in Table 5. This analysis computes Smatch for sub-sets of AMR to loosely reflect particular subtasks, such as Word Sense Disambiguation (WSD), Named Entity recognition or Semantic Role Labeling (SRL). The proposed approaches and the baseline consistently outperform prior art in a majority of categories and the main observable differences seems due to differences between the transition-based and graph recategorization approaches. Wikification and negation, the only categories where the proposed methods do not outperform (Cai and Lam, 2020), are handled by graph recategorization post-processing in this approach. Graph recategorization comes however at the cost of a large drop in the Name Entity category, probably due to need for graph post-processing using Core-NLP. Compared to this, transition-based approaches provide a more uniform performance across categories, and in this context the presented self-learning methods are able to improve in all categories. One aspect that merits further study, is the increase in the Negation category when using synTxt, which improves 5.4 points, probably due to generation of additional negation examples.

7 Related Works

Mining for gold, introduced in Section 3, can be related to previous works addressing oracle limitations such as dynamic oracles (Goldberg and Nivre, 2012; Ballesteros et al., 2016), imitation learning (Goodman et al., 2016) and minimum risk training (Naseem et al., 2019). All these approaches increase parser robustness to its own errors by exposing it to actions that are often inferior to the oracle sequence in score. The approach presented here seeks only the small set of sequences improving over the oracle and uses them for conventional maximum likelihood training.

Synthetic text, introduced in Section 4, is related to Back-translation in Machine Translation (Sennrich et al., 2016). The approach presented here exploits however the fact that multiple sentences correspond to a single AMR and thus needs no external data. This is closer to recent work on question generation for question answering systems (Alberti et al., 2019), which also uses cycle consistency filtering.

Finally, regarding synthetic AMR, discussed in Section 5, with respect to prior work (Konstas et al., 2017b; van Noord and Bos, 2017) we show that synthetic AMR parsing still can yield improvements for high performance baselines, and introduce the cycle-consistency filtering.

8 Conclusions

In this work[6], we explored different ways in which trained models can be applied to improve AMR parsing performance via self-learning. Despite the recent strong improvements in performance through novel architectures, we show that the proposed techniques improve performance further, achieving new state-of-the-art on AMR 1.0 and AMR 2.0 tasks without the need for extra human annotations.

[6] https://github.com/IBM/transition-amr-parser/.

References

Chris Alberti, Daniel Andor, Emily Pitler, Jacob Devlin, and Michael Collins. 2019. Synthetic QA corpora generation with roundtrip consistency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275, Copenhagen, Denmark. Association for Computational Linguistics.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack LSTM parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010, Austin, Texas. Association for Computational Linguistics.

Deng Cai and Wai Lam. 2020. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for Abstract Meaning Representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546, Valencia, Spain. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ramon Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodget, and Radu Florian. 2020. Transition-based parsing with stack-transformers. In *Findings of the EMNLP2020 (to appear)*.

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.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. Noise reduction and targeted exploration in imitation learning for Abstract Meaning Representation parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–11, Berlin, Germany. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Tomasz Dwojak, and Rico Sennrich. 2016. The AMU-UEDIN submission to the WMT16 news translation task: Attention-based NMT models as feature functions in phrase-based SMT. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 319–325, Berlin, Germany. Association for Computational Linguistics.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017a. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada. Association for Computational Linguistics.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017b. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Chunchuan Lyu and Ivan Titov. 2018a. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, Melbourne, Australia. Association for Computational Linguistics.

Chunchuan Lyu and Ivan Titov. 2018b. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, Melbourne, Australia. Association for Computational Linguistics.

Manuel Mager, Ramón Fernandez Astudillo, Tahira Naseem, Md Arafat Sultan, Young-Suk Lee, Radu Florian, and Salim Roukos. 2020. Gpt-too: A language-model-first approach for amr-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Seattle, USA. Association for Computational Linguistics.

Tahira Naseem, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding Smatch: Transition-based AMR parsing with reinforcement learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4586–4592, Florence, Italy. Association for Computational Linguistics.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *arXiv preprint arXiv:1705.09980v2*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. 2016. Generating english from abstract meaning representations. In *Proceedings of the 9th international natural language generation conference*, pages 21–25.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. AMR parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94, Florence, Italy. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3784–3796, Hong Kong, China. Association for Computational Linguistics.