Pushing the Limits of AMR Parsing with Self-Learning

Anonymous

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 oracles. 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-entrancy 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 (Anon., 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 model to iteratively refine a rule-based AMR oracle (Ballesteros and Al-Onaizan, 2017; Anon., 2020) to yield better performing 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), using a combination of filtering criteria and a fine-tuning approach that pushes AMR performance to 77.8 on AMR1.0 and 80.9 on AMR2.0.

2 Baseline Parser and Setup

To test the proposed ideas, we utilized the stack-Transformer model (Anon., 2020) on the AMR1.0 (LDC2015E68) and AMR2.0 (LDC2017T10) datasets. The stack-Transformer is a transition-based parser with a modified Transformer architecture to encode the parser state. It uses a cross-entropy training loss with hyper-parameters similar to those of Machine Translation. We follow the setup in (Anon., 2020) for AMR2.0 and apply it to AMR1.0. Smatch (Cai and Knight, 2013) is used for evaluation and model selection. All models use checkpoint averaging (Junczys-Dowmunt et al., 2016) of the best 3 checkpoints and use a beam
size of $10^4$ while decoding. \cite{Anon.2020} uses embeddings from the last layer of RoBERTa base \cite{Liu et al.2019} as input to the parser. Instead, we use RoBERTa large and take an average of embeddings from all 24 layers. This considerably strengthens the baseline model from the original 79.2 Smatch on the AMR2.0 dev-set to 80.2 and attains 77.0 for the AMR1.0 dev-set. Henceforth, this will be referred to as the improved \cite{Anon.2020} baseline.

3 Oracle Self-Training

AMR parsing produces a graph $g$ from a sentence $s$. Transition-based AMR parsers \cite{Ballesteros and Al-Onaizan, 2017; Naseem et al., 2019; Anon., 2020} accomplish this by predicting an action sequence $a$ from $s$, which when applied over a state machine produces the graph $g = M(a, s)$. At training time, these parsers require the action sequence for each graph in the training data. These are determined by a rule-based oracle $a = O(g, s)$, relying on external word-to-node alignments \cite{Flanigan et al., 2014; Pourdamghani et al., 2016}. However, using a rule-based oracle is sub-optimal. The rule-based action sequences might be hard to learn, and more importantly, due to incomplete or incorrect alignments, the rule-based action sequences do not always produce the original graph. The oracle score for AMR 2.0, measured in Smatch, is 97.9 \cite{Anon., 2020}.

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 $\bar{a} \sim p(a \mid s)$. Both $\bar{a}$ and $a^*$ are run through the state machine to get graphs $\tilde{g}$ and $g^*$ respectively. We then replace $a^*$ by $\bar{a}$ if $\text{Smatch}(\tilde{g}, g) > \text{Smatch}(g^*, g)$ or $(\text{Smatch}(\tilde{g}, g) = \text{Smatch}(g^*, g)$ and $|\bar{a}| < |a^*|)$. 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 \cite{Anon., 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. Between rounds we trained a new model from scratch with the new oracle to improve mining. This led to 3.6% actions with better Smatch and 4.2% shorter length for AMR1.0 and 3.5% and 5.2% respectively for AMR2.0. This results in an improvement in oracle Smatch from 97.4 to 97.6 for AMR 1.0 and 97.9 to 98.1 for AMR 2.0.

Table 1 shows that mining for AMR leads to an overall improvement of 0.3 Smatch consistent 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.

![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 (□).](image)

<table>
<thead>
<tr>
<th>Technique</th>
<th>AMR1.0</th>
<th>AMR2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved \cite{Anon., 2020}</td>
<td>77.0 ±0.1</td>
<td>80.2 ±0.1</td>
</tr>
<tr>
<td>&lt; length</td>
<td>77.1 ±0.0</td>
<td>80.2 ±0.2</td>
</tr>
<tr>
<td>&gt; smatch</td>
<td>77.3 ±0.3</td>
<td>80.3 ±0.1</td>
</tr>
<tr>
<td>&lt; length ∪ &gt; smatch</td>
<td>77.3 ±0.1</td>
<td>80.5 ±0.1</td>
</tr>
</tbody>
</table>

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

Mining can be related to previous works addressing oracle limitations such as dynamic oracles \cite{Goldberg and Nivre, 2012; Ballesteros et al., 2016}, imitation learning \cite{Goodman et al., 2016} and minimum risk training \cite{Naseem et al., 2019}. All these approaches increase parser robustness to its own errors by exposing it to actions that are

---

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

2One round of mining takes around 20h, while normal model training takes 6h on a Tesla V100.
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.

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 generation 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 \( p(s | g) \) to produce sentences from gold AMR graphs. For each graph \( g \) in the training data, we generate 20 sentences via sampling \( \tilde{s} \sim p(s | g) \) 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 \( s \), \( g \). 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. Note that scoring directly against gold graph yielded worse results as a filter. If duplicate sentences are generated we keep only one; samples identical to the original gold sentence are also filtered out. Filtering prunes roughly 90% of the generated sentences. Two separate AMR-to-text systems are fine-tuned using AMR 1.0 and AMR 2.0 train sets to generate the respective data.

Table 2 shows that synthetic text generation can substantially enhance parser performance over the improved (Anon., 2020) baseline (see Sec. 2), for AMR2.0 and particularly for AMR1.0, possibly due to its smaller size.

Back-translation in Machine Translation (Sennrich et al., 2016) is perhaps the best known example of using pre-trained models to generate new input text data. The approach presented here exploits the fact that multiple sentences correspond to a single AMR and thus needs no external data. In this sense, it is closer to recent work on question generation for question answering systems (Alberti et al., 2019), which also uses cycle consistency filtering.

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 (Anon., 2020) parser of Sec. 2 to parse unlabeled sentences from the context portion of SQuAD-2.0\(^4\), 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 (Papineni et al., 2002) score is lower than a pre-specified 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 the external data. Consequently we apply BLEU-based filtering only to the input texts with no OOV with

<table>
<thead>
<tr>
<th>Technique</th>
<th>AMR1.0</th>
<th>AMR2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sents</td>
<td>Smatch</td>
<td>Sents</td>
</tr>
<tr>
<td>Impr. (Anon., 2020)</td>
<td>10K</td>
<td>77.0 ±0.1</td>
</tr>
<tr>
<td>synTxt</td>
<td>28K</td>
<td>77.3 ±0.1</td>
</tr>
</tbody>
</table>

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

\(^3\)synTxt training takes 17h for AMR 2.0 and 5h hours for AMR 1.0 on Tesla V100. AMR-to-text training for 15 epochs takes 4.5h on AMR 1.0 and 15h on AMR 2.0.

\(^4\)https://github.com/rajpurkar/SQuAD-explorer/tree/master/dataset/train-v2.0.json
respect to the human annotated corpus. After filtering, the synAMR data is reduced to 58k sentences.

Following prior work, we used a pre-training and fine-tuning strategy. We tested both 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 improvements for both corpora but the mix of human-annotated AMR and synAMR works particularly better for AMR2.0.

We also experimented with synAMR derived from various external data sources including Ontonotes5.0, BOLT-DF, Wikipedia, observing similar improvements.

6 Final Results

Table 4 compares the different proposed self-learning approaches and their combination with prior art parsers. Following (Anon., 2020), pre-trained embeddings are indicated as BERT base and large (Devlin et al., 2019), RoBERTa base and large (Liu et al., 2019). Graph Recategorization (Lyu and Titov, 2018b; Zhang et al., 2019b), indicated as $G$, is a pre-processing stage that segments text and graph to identify named entities and splits multi-sentence among other normalization operations. We also indicate the use of additional unlabeled data as $U$.

Results for AMR 1.0 are already state-of-the-art for the improved (Anon., 2020). This is remarkable taking into account that, excluding changes in RoBERTa, the original model hyperparameters were not changed. Transition based approaches (Ballesteros and Al-Onaizan, 2017; Naseem et al., 2019; Anon., 2020) are limited by the stack and buffer formalism and represent nodes by the words

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.

<table>
<thead>
<tr>
<th>Technique</th>
<th>AMR1.0</th>
<th>AMR2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved (Anon., 2020)</td>
<td>77.0 ±0.1</td>
<td>80.2 ±0.1</td>
</tr>
<tr>
<td>synAMR only</td>
<td>77.8±0.0</td>
<td>80.3 ±0.0</td>
</tr>
<tr>
<td>human+synAMR</td>
<td>78.0±0.0</td>
<td>81.2 ±0.0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>AMR1.0</th>
<th>AMR2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Lyu and Titov, 2018b)$^{U}$</td>
<td>73.7</td>
<td>74.4</td>
</tr>
<tr>
<td>(Naseem et al., 2019)$^{G}$</td>
<td>-</td>
<td>75.5</td>
</tr>
<tr>
<td>(Zhang et al., 2019b) $^{R,G}$</td>
<td>71.3</td>
<td>77.0</td>
</tr>
<tr>
<td>(Anon., 2020)$^{R}$</td>
<td>-</td>
<td>78.6 ±0.3</td>
</tr>
<tr>
<td>(Cai and Lam, 2020)$^{b}$</td>
<td>74.0</td>
<td>78.7</td>
</tr>
<tr>
<td>(Cai and Lam, 2020)$^{b,G}$</td>
<td>75.4</td>
<td>80.2</td>
</tr>
<tr>
<td>Improved (Anon., 2020) $^{A}$</td>
<td>76.2 ±0.1</td>
<td>79.7 ±0.2</td>
</tr>
<tr>
<td>oracle mining</td>
<td>76.2 ±0.1</td>
<td>79.8 ±0.1</td>
</tr>
<tr>
<td>synTxt</td>
<td>76.6 ±0.0</td>
<td>80.3 ±0.0</td>
</tr>
<tr>
<td>synAMR$^{U}$</td>
<td>76.8 ±0.0</td>
<td>80.5 ±0.1</td>
</tr>
<tr>
<td>mining + synTxt</td>
<td>76.5 ±0.1</td>
<td>80.3 ±0.1</td>
</tr>
<tr>
<td>mining + synAMR$^{U}$</td>
<td>76.7 ±0.0</td>
<td>80.4 ±0.0</td>
</tr>
<tr>
<td>synTxt + synAMR$^{U}$</td>
<td>77.7 ±0.2</td>
<td>80.7 ±0.2</td>
</tr>
<tr>
<td>mining + synTxt + synAMR$^{U}$</td>
<td>77.8 ±0.0</td>
<td>80.9 ±0.1</td>
</tr>
</tbody>
</table>

they are set-Smatch for AMR1.0 and AMR2.0

It is usual for AMR models to experience a performance drop from dev to test in both AMR corpora. This fall is however larger than expected for the oracle mining technique, which shows close to no improvement in individual results. The opposite happens for synthetic text (synTxt) which provides large gains against the baseline. Synthetic AMR (synAMR) drops in performance as well, but overall provides large improvements over the baseline.

Both synTxt and synAMR provide results above current state-of-the-art for AMR2.0 and a large margin improvement for AMR1.0. Combinations of these approaches push these numbers furthermore. The combination of the three methods achieves up to 80.9 Smatch, which is the best result obtained at the time of submission for AMR2.0. This same model obtains 77.8 for AMR1.0, which is 2.4 points above best previously published result (Cai and Lam, 2020).

7 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.

$^{6}$Full code will be made available. Examples of generated data available in additional material.

$^{5}$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.
References


