Aligning English Strings with Abstract Meaning Representation Graphs

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, Kevin Knight
Information Sciences Institute
Department of Computer Science
University of Southern California
{damghani,yanggao,ulf,knight}@isi.edu

Abstract

We align pairs of English sentences and corresponding Abstract Meaning Representations (AMR), at the token level. Such alignments will be useful for downstream extraction of semantic interpretation and generation rules. Our method involves linearizing AMR structures and performing symmetrized EM training. We obtain 86.5% and 83.1% alignment F score on development and test sets.

1 Introduction

Banarescu et al. (2013) describe a semantics bank of English sentences paired with their logical meanings, written in Abstract Meaning Representation (AMR). The designers of AMR leave open the question of how meanings are derived from English sentences (and vice-versa), so there are no manually-annotated alignment links between English words and AMR concepts. This paper studies how to build such links automatically, using co-occurrence and other information. Automatic alignments may be useful for downstream extraction of semantic interpretation and generation rules.

AMRs are directed, acyclic graphs with labeled edges, e.g., the sentence *The boy wants to go* is represented as:

```
(w / want-01
  :arg0 (b / boy)
  :arg1 (g / go-01
           :arg0 b))
```

We have hand-aligned a subset of the 13,050 available AMR/English pairs. We evaluate our automatic alignments against this gold standard. A sample hand-aligned AMR is here ("~n" specifies a link to the nth English word):

```
the boy wants to go
(w / want-01~3
  :arg0 (b / boy~2)
  :arg1 (g / go-01~5
           :arg0 b))
```

This alignment problem resembles that of statistical machine translation (SMT). It is easier in some ways, because AMR and English are highly cognate. It is harder in other ways, as AMR is graph-structured, and children of an AMR node are unordered. There are also fewer available training pairs than in SMT.

One approach is to define a generative model from AMR graphs to strings. We can then use EM to uncover hidden derivations, which alignments weakly reflect. This approach is used in string/string SMT (Brown et al., 1993). However, we do not yet have such a generative graph-to-string model, and even if we did, there might not be an efficient EM solution. For example, in syntax-based SMT systems (Galley et al., 2004), the generative tree/string transduction story is clear, but in the absence of alignment constraints, there are too many derivations and rules for EM to efficiently consider.

We therefore follow syntax-based SMT custom and use string/string alignment models in aligning our graph/string pairs. However, while it is straightforward to convert syntax trees into strings data (by taking yields), it is not obvious how to do this for unordered AMR graph elements. The example above also shows that gold alignment links reach into the internal nodes of AMR.

Prior SMT work (Jones et al., 2012) describes alignment of semantic graphs and strings, though their experiments are limited to the GeoQuery domain, and their methods are not described in detail. Flanigan et al (2014) describe a heuristic AMR/English aligner. While heuristic aligners can achieve good accuracy, they will not automatically improve as more AMR/English data comes

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 425–429,
October 25-29, 2014, Doha, Qatar.

online.

The contributions of this paper are:

- A set of gold, manually-aligned AMR/English pairs.
- An algorithm for automatically aligning AMR/English pairs.
- An empirical study establishing alignment accuracy of 86.5% and 83.1% F score for development and test sets respectively.

2 Method

We divide the description of our method into three parts: preprocessing, training, and postprocessing. In the preprocessing phase, we linearize the AMR graphs to change them into strings, clean both the AMR and English sides by removing stop words and simple stemming, and add a set of corresponding AMR/English token pairs to the corpus to help the training phase. The training phase is based on IBM models, but we modify the learning algorithm to learn the parameters symmetrically. Finally, in the postprocessing stage we rebuild the aligned AMR graph. These components are described in more detail below.

2.1 Preprocessing

The first step of the preprocessing component is to linearize the AMR structure into a string. In this step we record the original structure of nodes in the graph for later reconstruction of AMR. AMR has a rooted graph structure. To linearize this graph we run a depth first search from the root and print each node as soon as it it visited. We print but not expand the nodes that are seen previously. For example the AMR:

```
(w / want-01
  :arg0 (b / boy)
  :arg1 (g / go-01
           :arg0 b))
```

is linearized into this order: *w / want-01 :arg0 b / boy :arg1 g / go-01 :arg0 b*.

Note that semantically related nodes often stay close together after linearization.

After linearizing the AMR graph into a string, we perform a series of preprocessing steps including lowercasing the letters, removing stop words, and stemming.

The AMR and English stop word lists are generated based on our knowledge of AMR design. We know that tokens like *an*, *the* or *to be* verbs will very rarely align to any AMR token; similarly, AMR role tokens like *:arg0*, *:quant*, *:opt1* etc. as well as the *instance-of* token /, and tokens like *temporal-quantity* or *date-entity* rarely align to any English token. We remove these tokens from the parallel corpus, but remember their position to be able to convert the resulting string/string alignment back into a full AMR graph/English string alignment. Although some stopwords participate in gold alignments, by removing them we will buy a large precision gain for some recall cost.

We remove the word sense indicator and quotation marks for AMR concepts. For instance we will change *want-01* to *want* and *"ohio"* to *ohio*. Then we stem AMR and English tokens into their first four letters, except for role tokens in AMR. The purpose of stemming is to normalize English morphological variants so that they are easier to match to AMR tokens. For example English tokens *wants*, *wanting*, *wanted*, and *want* as well as the AMR token *want-01* will all convert to *want* after removing the AMR word sense indicator and stemming.

In the last step of preprocessing, we benefit from the fact that AMR concepts and their corresponding English ones are frequently cognates. Hence, after stemming, an AMR token often can be translated to a token spelled similarly in English. This is the case for English token *want* and AMR token *want* in the previous paragraph. To help the training model learn from this fact, we extend our sentence pair corpus with the set of AMR/English token pairs that are spelled identically after preprocessing. Also, for English tokens that can be translated into multiple AMR tokens, like *higher* and *high :degree more* we add the corresponding string/string pairs to the corpus. This set is extracted from existing lexical resources, including lists of comparative/superlative adjectives, negative words, etc.

After preprocessing, the AMR at the start of this section will change into: *want boy go* and the sentence *The boy wants to go* changes into *boy want to go*, and we will also add the identity pairs *want/want*, *boy/boy*, and *go/go* to the corpus.

2.2 Training

Our training method is based on IBM word alignment models (Brown et al., 1993). We modify the objective functions of the IBM models to en-

courage agreement between learning parameters in English-to-AMR and AMR-to-English directions of EM. The solution of this objective function can be approximated in an extremely simple way that requires almost no extra coding effort.

Assume that we have a set of sentence pairs $\{(E, A)\}$, where each E is an English sentence and each A is a linearized AMR. According to IBM models, A is generated from E through a generative story based on some parameters.

For example, in IBM Model 2, given E we first decide the length of A based on some probability $l = p(len(A)|len(E))$, then we decide the distortions based on a distortion table: $d = p(i|j, len(A), len(E))$. Finally, we translate English tokens into AMR ones based on a translation table $t = p(a|e)$ where a and e are AMR and English tokens respectively.

IBM models estimate these parameters to maximize the conditional likelihood of the data: $\theta_{A|E} = argmax L_{\theta_{A|E}}(A|E)$ or $\theta_{E|A} = argmax L_{\theta_{E|A}}(E|A)$ where θ denotes the set of parameters. The conditional likelihood is intrinsic to the generative story of IBM models. However, word alignment is a symmetric problem. Hence it is more reasonable to estimate the parameters in a more symmetric manner.

Our objective function in the training phase is:

$$\theta_{A|E}, \theta_{E|A} = argmax L_{\theta_{A|E}}(A|E) + L_{\theta_{E|A}}(E|A)$$
$$\text{subject to} \quad \theta_{A|E}\theta_E = \theta_{E|A}\theta_A = \theta_{A,E}$$

We approximate the solution of this objective function with almost no change to the existing implementation of the IBM models. We relax the constraint to $\theta_{A|E} = \theta_{E|A}$, then apply the following iterative process:

1. Optimize the first part of the objective function: $\theta_{A|E} = argmax L_{\theta_{A|E}}(A|E)$ using EM
2. Satisfy the constraint: set $\theta_{E|A} \propto \theta_{A|E}$
3. Optimize the second part of the objective function: $\theta_{E|A} = argmax L_{\theta_{E|A}}(E|A)$ using EM
4. Satisfy the constraint: set $\theta_{A|E} \propto \theta_{E|A}$
5. Iterate

Note that steps 1 and 3 are nothing more than running the IBM models, and steps 2 and 4 are just initialization of the EM parameters, using tables from the previous iteration. The initialization steps only make sense for the parameters that involve both sides of the alignment (i.e., the translation table and the distortion table). For the translation table we set $t_{E|A}(e|a) = t_{A|E}(a|e)$ for English and AMR tokens e and a and then normalize the t table. The distortion table can also be initialized in a similar manner. We initialize the fertility table with its value in the previous iteration.

Previously Liang et al. (2006) also presented a symmetric method for training alignment parameters. Similar to our work, their objective function involves summation of conditional likelihoods in both directions; however, their constraint is on agreement between predicted alignments while we directly focus on agreement between the parameters themselves. Moreover their method involves a modification of the E step of EM algorithm which is very hard to implement for IBM Model 3 and above.

After learning the parameters, alignments are computed using the Viterbi algorithm in both directions of the IBM models. We tried merging the alignments of the two directions using methods like grow-diag-final heuristic or taking intersection of the alignments and adding some high probability links in their union. But these methods did not help the alignment accuracy.

2.3 Postprocessing

The main goal of the postprocessing component is to rebuild the aligned AMR graph. We first insert words removed as stop words into their positions, then rebuild the graph using the recorded original structure of the nodes in the AMR graph.

We also apply a last modification to the alignments in the postprocessing. Observing that pairs like *worker* and *person :arg0-of work-01* appear frequently, and in all such cases, all the AMR tokens align to the English one, whenever we see any of AMR tokens *person*, *product*, *thing* or *company* is followed by *arg0-of*, *arg1-of* or *arg2-of* followed by an AMR concept, we align the two former tokens to what the concept is aligned to.

3 Experiments

3.1 Data Description

Our data consists of 13,050 publicly available AMR/English sentence pairs[1]. We have hand

[1]LDC AMR release 1.0, Release date: June 16, 2014 https://catalog.ldc.upenn.edu/LDC2014T12

aligned 200 of these pairs to be used as development and test sets[2]. We train the parameters on the whole data. Table 1 presents a description of the data. We do not count parenthesis, slash and AMR variables as AMR tokens. Role tokens are those AMR tokens that start with a colon. They do not represent any concept, but provide a link between concepts. For example in:

```
(w / want-01
  :arg0 (b / boy)
  :arg1 (g / go-01
           :arg0 b))
```

the first *:arg0* states that the first argument of the concept *wanting* is the *boy* and the second argument is *going*.

	train	dev	test
Sent. pairs	13050	100	100
AMR tokens	465 K	3.8 K (52%)	2.3 K (%55)
AMR role tokens	226 K	1.9 K (23%)	1.1 K (%22)
ENG tokens	248 K	2.3 K (76%)	1.7 K (%74)

Table 1: AMR/English corpus. The number in parentheses is the percent of the tokens aligned in gold annotation. Almost half of AMR tokens are role tokens, and less than a quarter of role tokens are aligned.

3.2 Experiment Results

We use MGIZA++ (Gao and Vogel, 2008) as the implementation of the IBM models. We run Model 1 and HMM for 5 iterations each, then run our training algorithm on Model 4 for 4 iterations, at which point the alignments become stable. As alignments are usually many to one from AMR to English, we compute the alignments from AMR to English in the final step.

Table 2 shows the alignment accuracy for Model 1, HMM, Model 4, and Model 4 plus the modification described in section 2.2 (Model 4+).

The alignment accuracy on the test set is lower than the development set mainly because it is intrinsically a harder set, as we only made small modifications to the system based on the development set. Recall error due to stop words is one difference.

[2]The development and test AMR/English pairs can be found in /data/split/dev/amr-release-1.0-dev-consensus.txt and /data/split/test/amr-release-1.0-test-consensus.txt, respectively. The gold alignments are not included in these files but are available separately.

	model	precision	recall	F score
Dev	Model 1	70.9	71.1	71.0
	HMM	87.6	80.1	83.7
	Model 4	89.7	80.4	84.8
	Model 4+	94.1	80.0	86.5
Test	Model 1	74.8	71.8	73.2
	HMM	83.8	73.8	78.5
	Model 4	85.8	74.9	80.0
	Model 4+	92.4	75.6	83.1

Table 2: Results on different models. Our training method (Model 4+) increases the F score by 1.7 and 3.1 points on dev and test sets respectively.

Table 3 breaks down precision, recall, and F score for role and non-role AMR tokens, and also shows in parentheses the amount of recall error that was caused by removing either side of the alignment as a stop word.

	token type	precision	recall	F score
Dev	role	77.1	48.7	59.7
	non-role	97.2	88.2	92.5
	all	94.1	80.0 (34%)	86.5
Test	role	71.0	37.8	49.3
	non-role	95.5	84.7	89.8
	all	92.4	75.6 (36%)	83.1

Table 3: Results breakdown into role and non-role AMR tokens. The numbers in the parentheses show the percent of recall errors caused by removing aligned tokens as stop words.

While the alignment method works very well on non-role tokens, it works poorly on the role tokens. Role tokens are sometimes matched with a word or part of a word in the English sentence. For example *:polarity* is matched with the *un* part of the word *unpopular*, *:manner* is matched with most adverbs, or even in the pair:

```
thanks
(t / thank-01
    :arg0 (i / i)
    :arg1 (y / you))
```

all AMR tokens including *:arg0* and *:arg1* are matched to the only English word *thanks*. Inconsistency in aligning role tokens has made this a hard problem even for human experts.

4 Conclusions and Future Work

In this paper we present the first set of manually aligned English/AMR pairs, as well as the first published system for learning the alignments between English sentences and AMR graphs that provides a strong baseline for future research in this area. As the proposed system learns the alignments automatically using very little domain knowledge, it can be applied in any domain and for any language with minor adaptations.

Computing the alignments between English sentences and AMR graphs is a first step for extraction of semantic interpretation and generation rules. Hence, a natural extension to this work will be automatically parsing English sentences into AMR and generating English sentences from AMR.

Acknowledgments

This work was supported by DARPA contracts HR0011-12-C-0014 and FA-8750-13-2-0045. The authors would like to thank David Chiang, Tomer Levinboim, and Ashish Vaswani (in no particular order) for their comments and suggestions.

References

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 *Linguistic Annotation Workshop (LAW VII-ID), ACL.*

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *ACL.*

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *HLT-NAACL.*

Qin Gao and Stephan Vogel. 2008. Parallel implementations of word alignment tool. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing Workshop, ACL.*

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In *COLING.*

Percy Liang, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *HLT-NAACL.*