Oxford at SemEval-2017 Task 9: Neural AMR Parsing with Pointer-Augmented Attention

Jan Buys[1] **and Phil Blunsom**[1,2]
[1]Department of Computer Science, University of Oxford [2]DeepMind
{jan.buys,phil.blunsom}@cs.ox.ac.uk

Abstract

We present an end-to-end neural encoder-decoder AMR parser that extends an attention-based model by predicting the alignment between graph nodes and sentence tokens explicitly with a pointer mechanism. Candidate lemmas are predicted as a pre-processing step so that the lemmas of lexical concepts, as well as constant strings, are factored out of the graph linearization and recovered through the predicted alignments. The approach does not rely on syntactic parses or extensive external resources. Our parser obtained 59% Smatch on the SemEval test set.

1 Introduction

The task of parsing sentences to Abstract Meaning Representation (AMR) (Banarescu et al., 2013) has recently received increased attention. AMR represents sentence meaning with directed acyclic graphs (DAGs) with labelled nodes and edges. No assumptions are made about the relation between an AMR and the structure of the sentence it represents: the representation is not assumed to have any relation to the sentence syntax, no alignments are given and no distinction is made between concepts that correspond directly to lexemes in the input sentences and those that don't.

This underspecification creates significant challenges for training an end-to-end AMR parser, which are exacerbated by the relatively small sizes of available training sets. Consequently most AMR parsers are pipelines that make extensive use of additional resources. Neural encoder-decoders have previously been proposed for AMR parsing, but reported accuracies are well below the state-of-the-art (Barzdins and Gosko, 2016), even with sophisticated pre-processing and categorization (Peng et al., 2017). The end-to-end neural approach contrasts with approaches based on a pipeline of multiple LSTMs (Foland Jr and Martin, 2016) or neural network classifiers inside a feature- and resource-rich parser (Damonte et al., 2017), which have performed competitively.

Our approach addresses these challenges in two ways: This first is to utilize (noisy) alignments, aligning each graph node to an input token. The alignments are predicted explicitly by the neural decoder with a pointer network (Vinyals et al., 2015), in addition to a standard attention mechanism. Our second contribution is to introduce more structure in the AMR linearization by distinguishing between lexical and non-lexical concepts, noting that lexical concepts (excluding sense labels) can be predicted with high accuracy from their lemmas. The decoder predicts only delexicalized concepts, recovering the lexicalization through the lemmas corresponding to the predicted alignments.

Experiments show that our extensions increase parsing accuracy by a large margin over a standard attention-based model.

2 Graph Linearization and Lemmatization

We start by discussing how to linearize AMR graphs to enable sequential prediction. AMR node labels are referred to as *concepts* and edge labels as *relations*. A special class of node modifiers, called *constants*, are used to denote the string values of named entities and numbers. An example AMR graph is visualized in Figure 1.

In AMR datasets, graphs are represented as spanning trees with designated root nodes. Edges whose direction in the spanning tree are reversed are marked by adding "-of" to the argument label.

Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017), pages 914–919,
Vancouver, Canada, August 3 - 4, 2017.

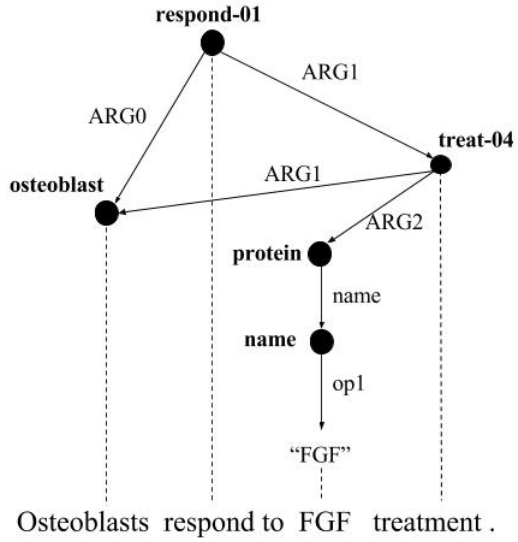

Figure 1: AMR graph aligned to the sentence it represents.

```
:focus( respond-01
  :ARG0( obsteoblast )
  :ARG1( treat-04
    :ARG1*( obsteoblast )
    :ARG2( protein
      :name( name
        :op1( "FGF" ) ) ) ) )
```

Figure 2: Standard linearized representation of the AMR in Figure 1.

Edges not included in the spanning tree (*reentrancies*) are indicated by adding dummy nodes pointing back to the original nodes.

The first linearization we propose (which we refer to as *standard*) is similar, except that nodes are identified through their concepts rather than explicit node identifiers. Constants are also treated as nodes. Reentrancy edges are marked with $\star$ and the concepts of their dependent nodes are simply repeated. During post-processing reentrancies are recovered heuristically by finding the closest nodes in the linear representation with the same concepts. An example of this representation is given in Figure 2.

In the second representation (*lexicalized*) every graph node is aligned to an input token. The alignments could be encoded as strings in the graph linerization, but in our model we will predict them separately. Every constant is replaced with a placeholder `CONST` token; the constant string is then recovered as a post-processing step through the predicted token alignment.

```
:focus( <2> -01
  :ARG0( <1> -u )
  :ARG1( <5> -04
    :ARG1*( <1> -u )
    :ARG2( <4> protein
      :name( <4> name
        :op1( <4> CONST ) ) ) ) )
```

Figure 3: Delexicalized linearization, with alignments, of the AMR in Figure 1.

We classify concepts in an AMR graph as either lexical, i.e. corresponding directly to the meaning of an aligned token, or non-lexical. This distinction, together with alignments, is annotated explicitly in Minimal Recursion Semantics predicates in the English Resource Grammar (ERG) (Copestake et al., 2005). However for AMR we classify concepts heuristically, based on automatic alignments. We assume that each word in a sentence aligns to at most one lexical node in its AMR graph. Where multiple nodes are aligned to the same token, usually forming a subgraph, the lowest element is taken to be the lexical concept.

A subset of AMR concepts are predicates based on PropBank framesets (Palmer et al., 2005), represented as sense-labeled lemmas. The remaining lexical concepts are usually English words in lemma form, while non-lexical concepts are usually special keywords. Lemmas can be predicted with high accuracy from the words they align to.

Our third linearization (*delexicalized*) factorizes the lemmas of lexical concepts out of the linerization, so that they are represented by their alignments and sense labels, e.g. `-01` for predicates and `-u` for other concepts. Candidate lemmas are predicted independently and lexicalized concepts are recovered as a post-processing step. This representation (see Figure 3) decreases the vocabulary of the decoder, which simplifies the learning problem and speeds up the parser.

2.1 Pre-processing

We tokenize the data with the Stanford CoreNLP toolkit (Manning et al., 2014). This tokenization corresponds more closely to AMR concepts and constants than other tokenizers we experimented with, especially due to its handling of hyphenation in the biomedical domain. We perform POS and

NE tagging with the same toolkit.

The training data is aligned with the rule-based JAMR aligner (Flanigan et al., 2014). However, our approach requires single-token alignments for all nodes, which JAMR is not guaranteed to give. We align each Wiki node to the token with the highest prefix overlap. Other nodes without alignments are aligned to the left-most alignment of their children (if they have any), otherwise to that of their parents. JAMR aligns multi-word named entities as single subgraph to token span alignments. We split these alignments to be 1-1 between tokens and constants. For other nodes with multi-token alignments we use the start of the given span.

For each token we predict candidate lexemes using a number of lexical resources. A summary of the resources used for each lexical type is given in Table 1. The first resource is dictionaries extracted from the aligned training data of each type, mapping each token or span of tokens to its most likely concept lemma or constant. A similar dictionary is extracted from Propbank framesets (included in LDC2016E25) for predicate lemmas. Next we use WordNet (Miller, 1995), as available through NLTK (Bird et al., 2009), to map words to verbalized forms (for predicates) or nominalized forms (for other concepts) via their synsets, where available. To predict constant strings corresponding to unseen named entities we use the forms predicted by the Stanford NE tagger (Finkel et al., 2005), which are broadly consistent with the conventions used for AMR annotation. The same procedure converts numbers to numerals. We use SUTime (Chang and Manning, 2012) to extract normalized forms of dates and time expressions.

Input sentences and output graphs in the training data are pre-processed independently. This introduces some noise in the training data, but makes it more comparable to the setup used during testing. The (development set) oracle accuracy is 98.7% Smatch for the standard representation, 96.16% for the aligned lexicalized representation and 93.48% for the unlexicalized representation.

3 Pointer-augmented neural attention

Let $\mathbf{e}_{1:I}$ be a tokenized English sentence, $\mathbf{f}_{1:J}$ a sequential representation of its AMR graph and $\mathbf{a}_{1:J}$ an alignment sequence of integers in the range 1 to I. We propose an attention-based encoder-decoder model (Bahdanau et al., 2015) to encode $\mathbf{e}$ and predict $\mathbf{f}$ and $\mathbf{a}$, the latter with a pointer network (Vinyals et al., 2015). We use a standard LSTM architecture (Jozefowicz et al., 2015).

For every token e we embed its word, POS tag and named entity (NE) tag as vectors; these embeddings are concatenated and passed through a linear layer such that the output $g(e)$ has the same dimension as the LSTM cell. This representation of $\mathbf{e}$ is then encoded with a bidirectional RNN. Each token e_i is represented by a hidden state h_i, which is the concatenation of its forward and backward LSTM state vectors.

Let s_j be the RNN decoder hidden state at output position j. We set s_0 to be the final RNN state of the backward encoder LSTM. The alignment a_j is predicted at each time-step with a pointer network (Vinyals et al., 2015), although it will only affect the output when f_j is a lexical concept or constant. The alignment logits are computed with an MLP (for $i = 1, \ldots, I$):

$$u_j^i = w^T \tanh(W^{(1)} h_i + W^{(2)} s_j).$$

The alignment distribution is then given by

$$p(a_j | \mathbf{a}_{1:j-1}, \mathbf{f}_{1:j-1}, \mathbf{e}) = \text{softmax}(u_j).$$

Attention is computed similarly, but parameterized separately, and the attention distribution α_j is not observed. Instead $q_j = \sum_{i=1}^{i=I} \alpha_j^i h_i$ is a weighted average of the encoder states.

The output distribution is computed as follows: RNN state s_j, aligned encoder representation h_{a_j} and attention vector q_j are fed through a linear layer to obtain o_j, which is then projected to the output logits $v_j = Ro_j + b$, such that

$$p(f_j | \mathbf{f}_{1:j-1}, \mathbf{e}) = \text{softmax}(v_j).$$

Let $v(f_j)$ be the decoder embedding of f_j. To compute the RNN state at the next time-step, let d_j be the output of a linear layer over $d(f_j)$, q_j and h_{a_j}. The next RNN state is then computed as

$$s_{j+1} = RNN(d_j, s_j).$$

We perform greedy decoding. We ensure that the output is well-formed by skipping over out-of-place symbols. Repeated occurrences of sibling subtrees are removed when equivalent up to the argument number of relations.

Candidate Type	JAMR alignments	PropBank	WordNet	NE Tagger	Lemmatizer
Predicates	✓	✓	✓	✗	✓
Other concepts	✓	✗	✓	✗	✓
Constants	✓	✗	✗	✓	✓
Wikification	✓	✗	✗	✓	✗

Table 1: Resources used to predict candidate lemmas for different types of AMR outputs. The left-most resource that has a prediction available is used.

Model	Smatch F1
Attention, no tags	54.60
Attention, with tags	57.27
Pointer, lexicalized	57.99
Pointer, delexicalized	59.18

Table 2: Development set results for the Bio AMR corpus.

Metric	Neural AMR (average)
Smatch	59 (53.67)
Unlabeled	63 (57.83)
No WSD	59 (53.67)
Named Entities	66 (55.83)
Wikification	18 (33.00)
Negation	27 (23.17)
Concepts	74 (71.17)
Reentrancies	43 (34.17)
SRL	57 (50.33)

Table 3: SemEval test set results on various metrics, reported as rounded to the nearest percentage.

Model	Smatch F1
Bio AMR	59.27
LDC	61.89

Table 4: Test set results for the Bio AMR and LDC2016E25 corpora.

4 Experiments

We train our models with the two AMR datasets provided for the shared task: LDC2016E25, a large corpus of newswire, weblog and discussion forum text with a training set of 35,498 sentences, and a smaller dataset in the biomedical domain (Bio AMR Corpus) with 5,542 training sentences. When training a parser for the biomedical domain with minibatch SGD, we sample Bio AMR sentences with a weight of 7 to each LDC sentence to balance the two sources in sampled minibatches.

Our models are implemented in TensorFlow (Abadi et al., 2015). We train models with Adam (Kingma and Ba, 2015) with learning rate 0.01 and minibatch size 64. Gradients norms are clipped to 5.0 (Pascanu et al., 2013). We use single-layer LSTMs with hidden state size 256, with dropout 0.3 on the input and output connections. The encoder takes word embeddings of size 512, initialized (in the first 100 dimensions) with embeddings trained with a structured skip-gram model (Ling et al., 2015), and POS and NE embeddings of size 32. Singleton tokens are replaced with an unknown word symbol with probability 0.5 during training.

We compare our pointer-based architecture against an attention-based encoder-decoder that does not make use of alignments or external lexical resources. We report results for two versions of this baseline: In the first, the input is purely word-based. The second embeds named entity and POS embeddings in the encoder, and utilizes pre-trained word embeddings. Development set results are given in Table 2. We see that POS and NE embeddings give a substantial improvement. The performance of the baseline with richer embeddings is similar to that of the first pointer-based model. The main difference between these two models is that the latter uses pointers to predict constants, so the results show that the gain due to this improved generalization is relatively small. The delexicalized representation with separate lemma prediction improves accuracy by 1.2%.

Official results on the shared task test set are presented in Table 3. AMR graphs are evaluated with Smatch (Cai and Knight, 2013), and further analysis is done with the metrics proposed by Damonte et al. (2017). The performance of our model is consistently better than the shared task average on all metrics except for Wikification; the reason for this is that we are not using a Wikifier to predict Wiki entries. The performance on predicting reentrancies is particularly encouraging, as it shows that our pointer-based model is able to learn to point to concepts with multiple occurrences.

To enable future comparison we also report results on the Bio AMR test set, as well as for training and testing on the newswire and discussion forum data (LDC2016E25) only (Table 4).

5 Conclusion

We proposed a novel approach to neural AMR parsing. Results show that neural encoder-decoder models can obtain strong performance on AMR parsing by explicitly modelling structure implicit in AMR graphs.

Acknowledgments

The first author thanks the financial support of the Clarendon Fund and the Skye Foundation. We thank the anonymous reviewers for their feedback.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. http://tensorflow.org/.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*. http://arxiv.org/abs/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*. pages 178–186. http://www.aclweb.org/anthology/W13-2322.

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. In *Proceedings of SemEval*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Shu Cai and Kevin Knight. 2013. Smatch: An evaluation metric for semantic feature structures. In *Proceedings of ACL (2)*.

Angel X Chang and Christopher D Manning. 2012. SUTime: A library for recognizing and normalizing time expressions. In *Proceedings of LREC*.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A Sag. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation* 3(2-3):281–332.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proceedings of EACL*. pages 536–546. http://www.aclweb.org/anthology/E17-1051.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of ACL*. pages 363–370. http://dx.doi.org/10.3115/1219840.1219885.

Jeffrey Flanigan, Sam Thomson, Jaime G. Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of ACL*. pages 1426–1436. http://aclweb.org/anthology/P/P14/P14-1134.pdf.

William R Foland Jr and James H Martin. 2016. CU-NLP at SemEval-2016 Task 8: AMR parsing using LSTM-based recurrent neural networks. In *Proceedings of SemEval*. pages 1197–1201.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *Proceedings of ICML*. pages 2342–2350.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*. http://arxiv.org/abs/1412.6980.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of NAACL*. pages 1299–1304. http://www.aclweb.org/anthology/N15-1142.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL System Demonstrations*. pages 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010.

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM* 38(11):39–41. https://doi.org/10.1145/219717.219748.

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

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *Proceedings of ICML (3)* 28:1310–1318.

Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. 2017. Addressing the data sparsity issue in neural amr parsing. In *Proceedings of EACL*. pages 366–375. http://www.aclweb.org/anthology/E17-1035.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, Curran Associates, Inc., pages 2692–2700. http://papers.nips.cc/paper/5866-pointer-networks.pdf.