Factorising AMR generation through syntax

Kris Cao and **Stephen Clark**
Computer Laboratory
University of Cambridge
United Kingdom
{kc391,sc609}@cam.ac.uk

Abstract

Generating from Abstract Meaning Representation (AMR) is an underspecified problem, as many syntactic decisions are not constrained by the semantic graph. To explicitly account for this underspecification, we break down generating from AMR into two steps: first generate a syntactic structure, and then generate the surface form. We show that decomposing the generation process this way leads to state-of-the-art single model performance generating from AMR without additional unlabelled data. We also demonstrate that we can generate meaning-preserving syntactic paraphrases of the same AMR graph, as judged by humans.

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic annotation framework which abstracts away from the surface form of text to capture the core 'who did what to whom' structure. As a result, generating from AMR is underspecified (see Figure 1 for an example). Single-step approaches to AMR generation (Flanigan et al., 2016; Konstas et al., 2017; Song et al., 2016, 2017) therefore have to decide the syntax and surface form of the AMR realisation in one go. We instead explicitly try and capture this syntactic variation and factor the generation process through a syntactic representation (Walker et al., 2001; Dušek and Jurcicek, 2016; Gardent and Perez-Beltrachini, 2017; Currey and Heafield, 2018).

First, we generate a delexicalised constituency structure from the AMR graph using a syntax model. Then, we fill out the constituency structure with the semantic content in the AMR graph using a lexicalisation model to generate the final surface form. Breaking down the AMR generation process this way provides us with several advantages: we disentangle the variance caused by the choice of syntax from that caused by the choice of words. We can therefore realise the same AMR graph with a variety of syntactic structures by sampling from the syntax model, and deterministically decoding using the lexicalisation model. We hypothesise that this generates better paraphrases of the reference realisation than sampling from a single-step model.

We linearise both the AMR graphs (Konstas et al., 2017) and constituency trees (Vinyals et al., 2015b) to allow us to use sequence-to-sequence models (Sutskever et al., 2014; Bahdanau et al., 2015) for the syntax and lexicalisation models. Further, as the AMR dataset is relatively small, we have issues with data sparsity causing poor parameter estimation for rarely seen words. We deal with this by anonymizing named entities, and including a *copy mechanism* (Vinyals et al., 2015a; See et al., 2017; Song et al., 2018) into our decoder, which allows open-vocabulary token generation.

We show that factorising the generation process in this way leads to improvements in AMR generation, setting a new state of the art for single-model AMR generation performance training only on labelled data. We also verify our diverse generation hypothesis with a human annotation study.

2 Data

Abstract Meaning Repreentation Abstract Meaning Representation is a semantic annotation formalism which represents the meaning of an English utterance as a rooted directed acyclic graph. Nodes in the graph represent entities, events, properties and states mentioned in the text, while leaves of the graph label the nodes with concepts (which do not have to be aligned to spans in the text). Re-entrant nodes correspond to coreferent entities. Edges in the graph represent

```
(g / give-01
   :ARG0 (i / I)
   :ARG1 (b / ball)
   :ARG2 (d / dog))
```

```
give :arg0 i :arg1 ball :arg2 dog
```

I [gave]$_{\text{VP}}$ [the dog]$_{\text{NP}}$ [a ball]$_{\text{NP}}$
I [gave]$_{\text{VP}}$ [the ball]$_{\text{NP}}$ [to a dog]$_{\text{PP}}$

Figure 1: An example AMR graph, with variable names and verb senses, followed by the input to our system after preprocessing, and finally two sample realisations different in syntax.

relations between entities in the text. See Figure 1 for an example of an AMR graph, together with sample realisations.

Konstas et al. (2017) outline a set of preprocessing procedures for AMR graphs to both render them suitable for sequence-to-sequence learning and to ameliorate data sparsity; we follow the same pipeline. We train our models on the two most recent AMR releases. LDC2017T10 has roughly 36k training sentences, while LDC2015E86 is about half this size. Both share dev and test sets, facilitating comparison.

Constituency syntax While there are many syntactic annotation formalisms, we use delexicalised Penn treebank-style constituency trees to represent syntax. Constituency trees have the advantage of a well-defined linearization order compared to dependency trees. Further, constituency trees may be easier to realise, as they effectively correspond to a bracketing of the surface form.

Unfortunately, AMR annotated data does not come with syntactic annotation. We therefore parse the training and dev splits of both corpora with the Stanford parser (Manning et al., 2014) to provide silver-standard reference parse trees. We then delexicalise the parse trees by trimming the trees of the surface words; after this stage, the leaves of the tree are the preterminal POS tags. After this, we linearise the delexicalised constituency trees with depth-first traversal, following Vinyals et al. (2015b).

3 Model implementation and training

3.1 Model details

We wish to estimate $P(Y, Z|X)$, the joint probability of a parse Y and surface form Z given an AMR graph X. We model this in two parts, using the chain rule to decompose the joint distribution. The first model, which we call the syntax model, approximates $P(Y|X)$, the probability of a particular syntactic structure for a meaning representation. The second is $P(Z|X, Y)$, the lexicalisation model. This calculates the probability of a surface realisation given a parse tree and an AMR graph. We implement both as recurrent sequence-to-sequence models.

As we are able to linearise both the AMR graph and the parse tree, we use LSTMs (Hochreiter and Schmidhuber, 1997) both as the encoder and the decoder of our seq2seq models. Given an input sequence $X_1, \ldots, X_n$, which can either be an AMR graph or a parse tree, we first embed the tokens to obtain a dense vector representation of each token $x_1, \ldots, x_n$. Then we feed this into a stacked bidirectional LSTM encoder to obtain contextualised representations of each input token c_i. As far as possible, we share parameters between our two models. Concretely, this means that the syntax model uses the same AMR and parse embeddings, and AMR encoder, as the lexicalisation model. We find that this speeds up model inference, as we only have to encode the AMR sequence once for both models. Further, it regularises the joint model by reducing the number of parameters.

In our decoder, we use the dot-product formulation of attention (Luong et al., 2015): the attention potentials a_i at timestep t are given by

$$a_i = h_{t-1}^T W_{att} c_i$$

where h_{t-1} is the decoder hidden state at the previous timestep, and c_i is the context representation at position i given by the encoder. The attention weight w_i is then given by a softmax over the attention potentials, and the overall context representation s_t is given by $\sum w_i c_i$. The syntax model only attends over the input AMR graph; the linearisation model attends over both the input AMR and syntax tree independently, and the resulting context representation s_t is given by the concatenation of the AMR context representation and the syntax tree context representation (Libovický and Helcl, 2017).

We use s_t to augment the input to the LSTM: $\tilde{y}_t = W_{in} \tanh([y_t; s_t])$. Then the LSTM hidden and cell state are updated according to the LSTM equations: $h_t, c_t = LSTM(h_{t-1}, c_{t-1}, \tilde{y}_t)$. Finally, we again concatenate s_t to h_t before calculating the logits over the distribution of tokens:

$$\widetilde{h}_t = \tanh(W_{out}[h_t; s_t]) \tag{1}$$

$$p(y_t|y_{<t}) = \text{softmax}(W\widetilde{h}_t) \tag{2}$$

For the syntax model, we further constrain the decoder to only produce valid parse trees; as we build the parse tree left-to-right according to a depth-first traversal, the permissible actions at any stage are to open a new constituent, produce a terminal (i.e. a POS tag), or close the currently open constituent. We implement this constraint by setting the logits of all impermissible actions to negative infinity before taking the softmax. We find that this improves both training speed and final model performance, as we imbue the decoder with an intrinsic bias towards producing well-formed parse trees.

3.2 Generation with a copy mechanism

Despite the preprocessing procedures referred to in Section 2, we found that the lexicalisation model still had trouble with out-of-vocabulary words, due to the small size of the training corpus. This led to poor vocabulary coverage on the unseen test portions of the dataset. On closer inspection, many out-of-vocabulary words in the dev split are open-class nouns and verbs, which correspond to concept nodes in the AMR graph. We therefore incorporate a copy mechanism (Vinyals et al., 2015a; See et al., 2017) into our lexicalisation model to make use of these alignments.

We implement this by decomposing the word generation probability into a weighted sum of two terms. One is the vocabulary generation term. This models the probability of generating the next token from the model vocabulary, and is calculated in the same way as the base model. The other is a copy term, which calculates the probability of generating the next token by copying a token from the input. This uses the attention distribution over the input tokens calculated in the decoder to decide which input token to copy. The weighting between these two terms is calculated as a function of the current decoder input token, the decoder hidden state, and the AMR and parse context vectors. To sum up, the per-word generation probability in the decoder is given by

$$p(y_t|y_{<t}) = (1-\theta_t)p_{lex}(y_t|y_{<t}) + \theta_t \sum_{i:X_i=y_t} w_i \tag{3}$$

where $p_{lex}(y_t|y_{<t})$ is as in Equation 2 and w_i is the attention weight on the input token X_i. θ is the weighting between the generation term and the copy term: this is implemented as a 2-layer MLP.

3.3 Model training procedures

The AMR training corpus, together with the automatically derived parse trees, give us aligned triples of AMR graph, parse tree and realisation. We train our model to minimise the sum of the parse negative log-likelihood from the syntax model and the text negative log-likelihood from the lexicalisation model. We use the ADAM optimizer (Kingma and Ba, 2015) with batch size 40 for 200 epochs. We evaluate model BLEU score on the dev set during training, and whenever this did not increase after 5 epochs, we multiplied the learning rate by 0.8. We select the model with the highest dev BLEU score during training as our final model.

We apply layer normalization (Ba et al., 2016) to all matrix multiplications inside our network, including in the LSTM cell, and drop out all non-recurrent connections with probability 0.5 (Srivastava et al., 2014). We also drop out recurrent connections in both encoder and decoder LSTMs with probability 0.3, tying the mask across timesteps as suggested by Gal and Ghahramani (2016). All model hidden states are size 500, and token embeddings are size 300. Word embeddings are initialised with pretrained `word2vec` embeddings (Mikolov et al., 2013). We replace words with count 1 in the training corpus with the UNK token with probability 0.5, and replace POS tags in the parse tree and AMR concepts with the UNK token with probability 0.1 regardless of count.

Decoding from our model During test time, we would like to estimate

$$\arg\max_Z \sum_Y P(Z, Y|X) \tag{4}$$

the most likely text realisation of an AMR, marginalising out over the possible parses. To do this, we heuristically find the n best parses $Y_1, \dots, Y_n$ from the syntax model, generate a realisation Z_i for each parse Y_i, and take the highest scoring parse-realisation pair as the model output.

We use beam search with width 2 for both steps, removing complete hypotheses from the active beam and appending them to a k-best list. We terminate search after a predetermined number of steps, or if there are no active beam items left. After termination, if $k > n$, we return the top n items of the k-best list; otherwise we return additional

Model	Unlabelled F1	Labelled F1
Text-to-parse	87.5	85.8
AMR-to-parse	60.4	54.8
Unconditional	38.5	31.7

Table 1: Parsing scores on LDC2017T10 dev set.

Model	# good realisations
Syntax-aware model	1.52
Baseline s2s	1.19

Table 2: Average number of acceptable realisations out of 3. The difference is significant with $p < 0.001$.

items from the beam. In our experiments, we find that considering realisations of the 2 best parses (i.e. setting $n = 2$ above) gives the highest BLEU score on the dev set.

4 Experiment 1: AMR and syntax

We first investigate how much information AMR contains about possible syntactic realisations. We train two seq2seq models of the above architecture to predict the delexicalised constituency tree of an example given either the AMR graph or the text. We then evaluate both models on labelled and unlabelled F1 score on the dev split of the corpus. As neither model is guaranteed to produce trees with the right number of terminals, we first run an insert/delete aligner between the predicted and reference terminals (i.e. POS tags) before calculating span F1s. We also report the results of running our aligner on the most probable parse tree as estimated by an unconditional LSTM as a baseline both to control for our aligner and also to see how much extra signal is in the AMR graph. The results in Table 1 show that predicting a syntactic structure from an AMR graph is a much harder task than predicting from the text, but there is information in the AMR graph to improve over a blind baseline.

5 Experiment 2: Generating natural language from AMR

Table 3 shows the results of our model on the AMR generation task. We evaluate using BLEU score (Papineni et al., 2002) against the reference realisations. As a baseline, we train a straight AMR-to-text model with the same architecture as above to control for the extra regularisation in our model compared to previous work. Our results show that adding syntax into the model dramatically boosts performance, resulting in state-of-the-art single model performance on both datasets without using external training data.

Model	Dev BLEU	Test BLEU
Trained on LDC2017T10		
Our model	**26.1**	**26.8**
Our model + oracle parse	57.5	-
Baseline s2s + copy	23.7	23.5
Beck et al. (2018)	-	23.3
Trained on LDC2015E86		
Our model	**23.6**	**23.5**
Our model + oracle parse	53.1	-
Konstas et al. (2017)	21.7	22.0
Song et al. (2018)	22.8	23.3
Trained on LDC2015E86 or earlier + additional unlabelled data		
Song et al. (2018)	-	33.0
Konstas et al. (2017)	33.1	33.8
Pourdamghani et al. (2016)	27.2	26.9
Song et al. (2017)	25.2	25.6

Table 3: BLEU results for generation.

As an oracle experiment, we also generate from the realisation model conditioned on the ground truth parse. The outstanding result here – BLEU scores in the 50s – demonstrates that being able to predict the gold reference parse tree is a bottleneck in the performance of our model. However, given the inherent difficulty of predicting a single syntax realisation (cf. Section 4), we suspect that there is an intrinsic limit to how well generating from an AMR graph can replicate the reference realisation.

We further note that we do not use models tailored to graph-structured data or character-level features as in Song et al. (2018); Beck et al. (2018), or additional unlabelled data to perform semi-supervised learning (Konstas et al., 2017). We believe that we can improve our results even further if we use these techniques.

6 Experiment 3: Generating varied realisations

Our model explicitly disentangles variation caused by syntax choice from that caused by lexical choice. This means that we can generate diverse realisations of the same AMR graph by sampling from the syntax model and deterministically decoding from the realisation model. We hypothesise that this procedure generates more meaning-preserving realisations than just sampling from a straight AMR-to-text model, which can result in incoherent output (Cao and Clark, 2017).

We selected the first 50 AMR graphs in the dev set on linearised length between 15 and 40 with coherent reference realisations and generated 5 different realisations with our joint model and our baseline model. For our joint model, we first sampled 3 parse structures from the syntax model with temperature 0.3. This means we divide the per-timestep logits of the syntax decoder by 0.3; this serves to sharpen the outputs of the syntax model and constrains the sampling process to produce relatively high-probability syntactic structures for the given AMR. Then, we realised each parse deterministically with the lexicalisation model. For the baseline model, we sample 3 realisations from the decoder with the same temperature. This gave us 100 examples in total.

We then crowdsourced acceptability judgments for each example from 100 annotators: we showed the reference realisation of an AMR graph, together with model realisations, and asked each annotator to mark all the grammatical realisations which have the same meaning as the reference realisation. Each annotator was presented 30 examples selected randomly. Our results in Table 2 show that the joint model can generate more meaning-preserving realisations compared to a syntax-agnostic baseline. This shows the utility of separating out syntactic and lexical variation: we model explicitly meaning-preserving invariances, and can therefore generate better paraphrases.

7 Conclusions and further work

We present an AMR generation model that factors the generation process through a syntactic decision, and show that this leads to improved AMR generation performance. In addition, we show that separating the syntactic decisions from the lexicalisation decisions allows the model to generate higher quality paraphrases of a given AMR graph.

In future work, we would like to integrate a semantic parser into our model (Yin et al., 2018). Annotating data with AMR is expensive, and existing AMR treebanks are small. By integrating a component which parses into AMR into our model, we can do semi-supervised learning on plentiful unannotated natural language sentences, and improve our AMR generation performance even further. In addition, we would be able to generate text-to-text paraphrases by parsing into AMR first and then carrying out the paraphrase generation procedure described in this paper (Iyyer et al., 2018). This opens up scope for data augmentation for downstream NLP tasks, such as machine translation.

Acknowledgements

The authors would like to thank Amandla Mabona and Guy Emerson for fruitful discussions. KC is funded by an EPSRC studentship.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR*.

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. Association for Computational Linguistics.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 273–283. Association for Computational Linguistics.

Kris Cao and Stephen Clark. 2017. Latent variable dialogue models and their diversity. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 182–187. Association for Computational Linguistics.

Anna Currey and Kenneth Heafield. 2018. Multi-source syntactic neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2961–2966, Brussels, Belgium. Association for Computational Linguistics.

Ondřej Dušek and Filip Jurcicek. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 45–51, Berlin, Germany. Association for Computational Linguistics.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 731–739, San Diego, California. Association for Computational Linguistics.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*.

Claire Gardent and Laura Perez-Beltrachini. 2017. A statistical, grammar-based approach to microplanning. *Computational Linguistics*, 43(1):1–30.

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

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. 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.

Jindřich Libovický and Jindřich Helcl. 2017. Attention strategies for multi-source sequence-to-sequence learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 196–202, Vancouver, Canada. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

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

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, Edinburgh, UK. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083. Association for Computational Linguistics.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. Amr-to-text generation with synchronous node replacement grammar. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 7–13. Association for Computational Linguistics.

Linfeng Song, Yue Zhang, Xiaochang Peng, Zhiguo Wang, and Daniel Gildea. 2016. Amr-to-text generation as a traveling salesman problem. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2084–2089. Association for Computational Linguistics.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1616–1626. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015a. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing*

Systems 28, pages 2692–2700. Curran Associates, Inc.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015b. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2773–2781. Curran Associates, Inc.

Marilyn A. Walker, Owen Rambow, and Monica Rogati. 2001. Spot: A trainable sentence planner. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. 2018. Structvae: Tree-structured latent variable models for semi-supervised semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 754–765, Melbourne, Australia. Association for Computational Linguistics.