Structural Neural Encoders for AMR-to-text Generation

Marco Damonte **Shay B. Cohen**
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK
m.damonte@sms.ed.ac.uk
scohen@inf.ed.ac.uk

Abstract

AMR-to-text generation is a problem recently introduced to the NLP community, in which the goal is to generate sentences from Abstract Meaning Representation (AMR) graphs. Sequence-to-sequence models can be used to this end by converting the AMR graphs to strings. Approaching the problem while working directly with graphs requires the use of graph-to-sequence models that encode the AMR graph into a vector representation. Such encoding has been shown to be beneficial in the past, and unlike sequential encoding, it allows us to explicitly capture reentrant structures in the AMR graphs. We investigate the extent to which reentrancies (nodes with multiple parents) have an impact on AMR-to-text generation by comparing graph encoders to tree encoders, where reentrancies are not preserved. We show that improvements in the treatment of reentrancies and long-range dependencies contribute to higher overall scores for graph encoders. Our best model achieves 24.40 BLEU on LDC2015E86, outperforming the state of the art by 1.1 points and 24.54 BLEU on LDC2017T10, outperforming the state of the art by 1.24 points.

1 Introduction

Abstract Meaning Representation (AMR; Banarescu et al. 2013) is a semantic graph representation that abstracts away from the syntactic realization of a sentence, where nodes in the graph represent concepts and edges represent semantic relations between them. AMRs are graphs, rather than trees, because co-references and control structures result in nodes with multiple parents, called reentrancies. For instance, the AMR of Figure 1(a) contains a reentrancy between *finger* and *he*, caused by the possessive pronoun *his*. AMR-to-text generation is the task of automatically generating natural language from AMR graphs.

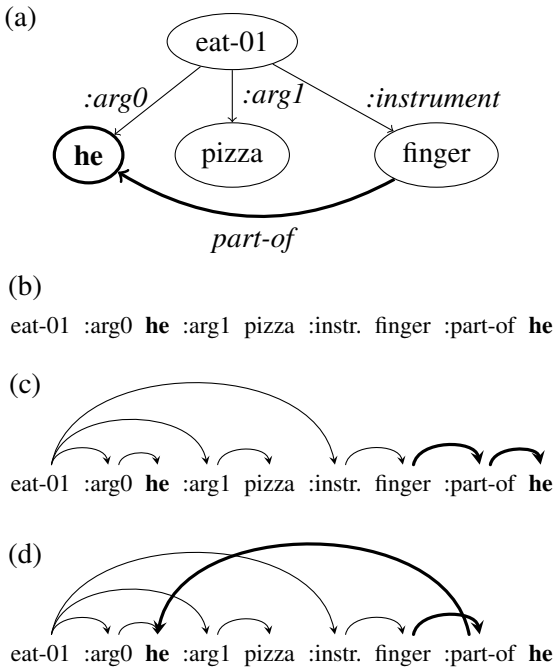

Figure 1: (a) AMR for the sentence *He ate the pizza with his fingers* and different input representations: (b) sequential; (c) tree-structured; (d) graph-structured. The nodes and edges in bold highlight a reentrancy.

Attentive encoder/decoder architectures, commonly used for Neural Machine Translation (NMT), have been explored for this task (Konstas et al., 2017; Song et al., 2018; Beck et al., 2018). In order to use sequence-to-sequence models, Konstas et al. (2017) reduce the AMR graphs to sequences, while Song et al. (2018) and Beck et al. (2018) directly encode them as graphs. Graph encoding allows the model to explicitly encode reentrant structures present in the AMR graphs. While central to AMR, reentrancies are often hard to treat both in parsing and in generation. Previous work either removed them from the graphs, hence obtaining sequential (Konstas et al.,

Proceedings of NAACL-HLT 2019, pages 3649–3658
Minneapolis, Minnesota, June 2 - June 7, 2019.

2017) or tree-structured (Liu et al., 2015; Takase et al., 2016) data, while other work maintained them but did not analyze their impact on performance (e.g., Song et al., 2018; Beck et al., 2018). Damonte et al. (2017) showed that state-of-the-art parsers do not perform well in predicting reentrant structures, while van Noord and Bos (2017) compared different pre- and post-processing techniques to improve the performance of sequence-to-sequence parsers with respect to reentrancies. It is not yet clear whether explicit encoding of reentrancies is beneficial for generation.

In this paper, we compare three types of encoders for AMR: 1) sequential encoders, which reduce AMR graphs to sequences; 2) tree encoders, which ignore reentrancies; and 3) graph encoders. We pay particular attention to two phenomena: reentrancies, which mark co-reference and control structures, and long-range dependencies in the AMR graphs, which are expected to benefit from structural encoding. The contributions of the paper are two-fold:

- We present structural encoders for the encoder/decoder framework and show the benefits of graph encoders not only compared to sequential encoders but also compared to tree encoders, which have not been studied so far for AMR-to-text generation.
- We show that better treatment of reentrancies and long-range dependencies contributes to improvements in the graph encoders.

Our best model, based on a graph encoder, achieves state-of-the-art results for both the LDC2015E86 dataset (24.40 on BLEU and 23.79 on Meteor) and the LDC2017T10 dataset (24.54 on BLEU and 24.07 on Meteor).

2 Input Representations

Graph-structured AMRs AMRs are normally represented as rooted and directed graphs:

$$G_0 = (V_0, E_0, L),$$
$$V_0 = \{v_1, v_2, \ldots, v_n\},$$
$$root \in V_0,$$

where V_0 are the graph vertices (or nodes) and $root$ is a designated root node in V_0. The edges in the AMR are labeled:

$$E_0 \subseteq V_0 \times L \times V_0,$$
$$L = \{\ell_1, \ell_2, \ldots, \ell_{n'}\}.$$

Each edge $e \in E_0$ is a triple: $e = (i, label, j)$, where $i \in V_0$ is the parent node, $label \in L$ is the edge label and $j \in V_0$ is the child node.

In order to obtain unlabeled edges, thus decreasing the total number of parameters required by the models, we replace each labeled edge $e = (i, label, j)$ with two unlabeled edges: $e_1 = (i, label), e_2 = (label, j)$:

$$G = (V, E),$$
$$V = V_0 \cup L = \{v_1, \ldots, v_n, \ell_1, \ldots, \ell_{n'}\},$$
$$E \subseteq (V_0 \times L) \cup (L \times V_0).$$

Each unlabeled edge $e \in E$ is a pair: $e = (i, j)$, where one of the following holds:

1. $i \in V_0$ and $j \in L$;
2. $i \in L$ and $j \in V_0$.

For instance, the edge between *eat-01* and *he* with label *:arg0* of Figure 1(a) is replaced by two edges in Figure 1(d): an edge between *eat-01* and *:arg0* and another one between *:arg0* and *he*. The process, also used in Beck et al. (2018), tranforms the input graph into its equivalent Levi graph (Levi, 1942).

Tree-structured AMRs In order to obtain tree structures, it is necessary to discard the reentrancies from the AMR graphs. Similarly to Takase et al. (2016), we replace nodes with $n > 1$ incoming edges with n identically labeled nodes, each with a single incoming edge.

Sequential AMRs Following Konstas et al. (2017), the input sequence is a linearized and anonymized AMR graph. Linearization is used to convert the graph into a sequence:

$$x = x_1, \ldots, x_N,$$
$$x_i \in V.$$

The depth-first traversal of the graph defines the indexing between nodes and tokens in the sequence. For instance, the root node is x_1, its leftmost child is x_2 and so on. Nodes with multiple parents are visited more than once. At each visit, their labels are repeated in the sequence, effectively losing reentrancy information, as shown in Figure 1(b).

Anonymization removes names and rare words with coarse categories to reduce data sparsity. An alternative to anonymization is to employ a copy

mechanism (Gulcehre et al., 2016), where the models learn to copy rare words from the input itself. In this paper, we follow the anonymization approach.

3 Encoders

In this section, we review the encoders adopted as building blocks for our tree and graph encoders.

3.1 Recurrent Neural Network Encoders

We reimplement the encoder of Konstas et al. (2017), where the sequential linearization is the input to a bidirectional LSTM (BiLSTM; Graves et al. 2013) network. The hidden state of the BiLSTM at step i is used as a context-aware word representation of the i-th token in the sequence:

$$e_{1:N} = \text{BiLSTM}(x_{1:N}),$$

where $e_i \in \mathbb{R}^d$, d is the size of the output embeddings.

3.2 TreeLSTM Encoders

Tree-Structured Long Short-Term Memory Networks (TreeLSTM; Tai et al. 2015) have been introduced primarily as a way to encode the hierarchical structure of syntactic trees (Tai et al., 2015), but they have also been applied to AMR for the task of headline generation (Takase et al., 2016). TreeLSTMs assume tree-structured input, so AMR graphs must be preprocessed to respect this constraint: reentrancies, which play an essential role in AMR, must be removed, thereby transforming the graphs into trees.

We use the Child-Sum variant introduced by Tai et al. (2015), which processes the tree in a bottom-up pass. When visiting a node, the hidden states of its children are summed up in a single vector which is then passed into recurrent gates.

In order to use information from both incoming and outgoing edges (parents and children), we employ bidirectional TreeLSTMs (Eriguchi et al., 2016), where the bottom-up pass is followed by a top-down pass. The top-down state of the root node is obtained by feeding the bottom-up state of the root node through a feed-forward layer:

$$h^{\downarrow}_{\text{root}} = \tanh(W_r h^{\uparrow}_{\text{root}} + b),$$

where $h^{\uparrow}_i$ is the hidden state of node $x_i \in V$ for the bottom-up pass and $h^{\downarrow}_i$ is the hidden state of node x_i for the top-down pass.

The bottom up states for all other nodes are computed with an LSTM, with the cell state given by their parent nodes:

$$h^{\downarrow}_i = \text{LSTM}(h^{\uparrow}_{p(i)}, h^{\uparrow}_i),$$

where $p(i)$ is the parent of node x_i in the tree. The final hidden states are obtained by concatenating the states from the bottom-up pass and the top-down pass:

$$h_i = \left[h^{\downarrow}_i; h^{\uparrow}_i\right].$$

The hidden state of the root node is usually used as a representation for the entire tree. In order to use attention over all nodes, as in traditional NMT (Bahdanau et al., 2015), we can however build node embeddings by extracting the hidden states of each node in the tree:

$$e_{1:N} = h_{1:N},$$

where $e_i \in \mathbb{R}^d$, d is the size of the output embeddings.

The encoder is related to the TreeLSTM encoder of Takase et al. (2016), which however encodes labeled trees and does not use a top-down pass.

3.3 Graph Convolutional Network Encoders

Graph Convolutional Network (GCN; Duvenaud et al. 2015; Kipf and Welling 2016) is a neural network architecture that learns embeddings of nodes in a graph by looking at its nearby nodes. In Natural Language Processing, GCNs have been used for Semantic Role Labeling (Marcheggiani and Titov, 2017), NMT (Bastings et al., 2017), Named Entity Recognition (Cetoli et al., 2017) and text generation (Marcheggiani and Perez-Beltrachini, 2018).

A graph-to-sequence neural network was first introduced by Xu et al. (2018). The authors review the similarities between their approach, GCN and another approach, based on GRUs (Li et al., 2015). The latter recently inspired a graph-to-sequence architecture for AMR-to-text generation (Beck et al., 2018). Simultaneously, Song et al. (2018) proposed a graph encoder based on LSTMs.

The architectures of Song et al. (2018) and Beck et al. (2018) are both based on the same core computation of a GCN, which sums over the embeddings of the immediate neighborhood of each

node:

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W_{(j,i)}^{(k)} h_j^{(k)} + b^{(k)}\right),$$

where $h_i^{(k)}$ is the embeddings of node $x_i \in V$ at layer k, σ is a non-linear activation function, $\mathcal{N}(i)$ is the set of the immediate neighbors of x_i, $W_{(j,i)}^{(k)} \in \mathbb{R}^{m \times m}$ and $b^{(k)} \in \mathbb{R}^m$, with m being the size of the embeddings.

It is possible to use recurrent networks to model the update of the node embeddings. Specifically, Beck et al. (2018) uses a GRU layer where the gates are modeled as GCN layers. Song et al. (2018) did not use the activation function σ and perform an LSTM update instead.

The systems of Song et al. (2018) and Beck et al. (2018) further differ in design and implementation decisions such as in the use of edge label and edge directionality. Throughout the rest of the paper, we follow the traditional, non-recurrent, implementation of GCN also adopted in other NLP tasks (Marcheggiani and Titov, 2017; Bastings et al., 2017; Cetoli et al., 2017). In our experiments, the node embeddings are computed as follows:

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W_{\mathrm{dir}(j,i)}^{(k)} h_j^{(k)} + b^{(k)}\right), \quad (1)$$

where $\mathrm{dir}(j,i)$ indicates the direction of the edge between x_j and x_i (i.e., outgoing or incoming edge). The hidden vectors from the last layer of the GCN network are finally used to represent each node in the graph:

$$e_{1:N} = h_1^{(K)}, \ldots, h_N^{(K)},$$

where K is the number of GCN layers used, $e_i \in \mathbb{R}^d$, d is the size of the output embeddings.

To regularize the models we apply dropout (Srivastava et al., 2014) as well as edge dropout (Marcheggiani and Titov, 2017). We also include highway connections (Srivastava et al., 2015) between GCN layers.

While GCN can naturally be used to encode graphs, they can also be applied to trees by removing reentrancies from the input graphs. In the experiments of Section 5, we explore GCN-based models both as graph encoders (reentrancies are maintained) as well as tree encoders (reentrancies are ignored).

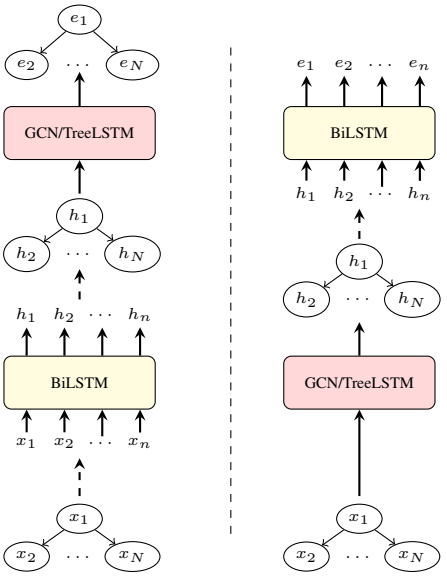

Figure 2: Two ways of stacking recurrent and structural models. Left side: structure on top of sequence, where the structural encoders are applied to the hidden vectors computed by the BiLSTM. Right side: sequence on top of structure, where the structural encoder is used to create better embeddings which are then fed to the BiLSTM. The dotted lines refer to the process of converting the graph into a sequence or vice-versa.

4 Stacking Encoders

We aimed at stacking the explicit source of structural information provided by TreeLSTMs and GCNs with the sequential information which BiLSTMs extract well. This was shown to be effective for other tasks with both TreeLSTMs (Eriguchi et al., 2016; Chen et al., 2017) and GCNs (Marcheggiani and Titov, 2017; Cetoli et al., 2017; Bastings et al., 2017). In previous work, the structural encoders (tree or graph) were used on top of the BiLSTM network: first, the input is passed through the sequential encoder, the output of which is then fed into the structural encoder. While we experiment with this approach, we also propose an alternative solution where the BiLSTM network is used on top of the structural encoder: the input embeddings are refined by exploiting the explicit structural information given by the graph. The refined embeddings are then fed into the BiLSTM networks. See Figure 2 for a graphical representation of the two approaches. In our experiments, we found this approach to be more effective. Compared to models that interleave structural and recurrent components such as the systems of Song et al. (2018) and Beck et al. (2018), stacking the components allows us to test

for their contributions more easily.

4.1 Structure on Top of Sequence

In this setup, BiLSTMs are used as in Section 3.1 to encode the linearized and anonymized AMR. The context provided by the BiLSTM is a sequential one. We then apply either GCN or TreeLSTM on the output of the BiLSTM, by initializing the GCN or TreeLSTM embeddings with the BiLSTM hidden states. We call these models SEQGCN and SEQTREELSTM.

4.2 Sequence on Top of Structure

We also propose a different approach for integrating graph information into the encoder, by swapping the order of the BiLSTM and the structural encoder: we aim at using the structured information provided by the AMR graph as a way to refine the original word representations. We first apply the structural encoder to the input graphs. The GCN or TreeLSTM representations are then fed into the BiLSTM. We call these models GCNSEQ and TREELSTMSEQ.

The motivation behind this approach is that we know that BiLSTMs, given appropriate input embeddings, are very effective at encoding the input sequences. In order to exploit their strength, we do not amend their output but rather provide them with better input embeddings to start with, by explicitly taking the graph relations into account.

5 Experiments

We use both BLEU (Papineni et al., 2002) and Meteor (Banerjee and Lavie, 2005) as evaluation metrics.[1] We report results on the AMR dataset LDC2015E86 and LDC2017T10. All systems are implemented in PyTorch (Paszke et al., 2017) using the framework OpenNMT-py (Klein et al., 2017). Hyperparameters of each model were tuned on the development set of LDC2015E86. For the GCN components, we use two layers, ReLU activations, and tanh highway layers. We use single layer LSTMs. We train with SGD with the initial learning rate set to 1 and decay to 0.8. Batch size is set to 100.[2]

We first evaluate the overall performance of the models, after which we focus on two phenomena that we expect to benefit most from structural encoders: reentrancies and long-range dependencies. Table 1 shows the comparison on the development split of the LDC2015E86 dataset between sequential, tree and graph encoders. The sequential encoder (SEQ) is a re-implementation of Konstas et al. (2017). We test both approaches of stacking structural and sequential components: structure on top of sequence (SEQTREELSTM and SEQGCN), and sequence on top of structure (TREELSTMSEQ and GCNSEQ). To inspect the effect of the sequential component, we run ablation tests by removing the RNNs altogether (TREELSTM and GCN). GCN-based models are used both as tree encoders (reentrancies are removed) and graph encoders (reentrancies are maintained).

Input	Model	BLEU	Meteor
Seq	SEQ	21.40	22.00
Tree	SEQTREELSTM	21.84	22.34
	TREELSTMSEQ	22.26	22.87
	TREELSTM	22.07	22.57
	SEQGCN	21.84	22.21
	GCNSEQ	**23.62**	**23.77**
	GCN	15.83	17.76
Graph	SEQGCN	22.06	22.18
	GCNSEQ	**23.95**	**24.00**
	GCN	15.94	17.76

Table 1: BLEU and Meteor (%) scores on the development split of LDC2015E86.

For both TreeLSTM-based and GCN-based models, our proposed approach of applying the structural encoder before the RNN achieves better scores. This is especially true for GCN-based models, for which we also note a drastic drop in performance when the RNN is removed, highlighting the importance of a sequential component. On the other hand, RNN layers seem to have less impact on TreeLSTM-based models. This outcome is not unexpected, as TreeLSTMs already use LSTM gates in their computation.

The results show a clear advantage of tree and graph encoders over the sequential encoder. The best performing model is GCNSEQ, both as a tree and as a graph encoder, with the latter obtaining the highest results.

Table 2 shows the comparison between our best sequential (SEQ), tree (GCNSEQ without reentrancies, henceforth called TREE) and graph en-

[1] We used the evaluation script available at https://github.com/sinantie/NeuralAmr.

[2] Our code is available at https://github.com/mdtux89/OpenNMT-py-AMR-to-text.

Model	BLEU	Meteor
LDC2015E86		
SEQ	21.43	21.53
TREE	23.93	23.32
GRAPH	**24.40**	**23.60**
Konstas et al. (2017)	22.00	-
Song et al. (2018)	23.30	-
LDC2017T10		
SEQ	22.19	22.68
TREE	24.06	23.62
GRAPH	**24.54**	**24.07**
Beck et al. (2018)	23.30	-

Table 2: Scores on the test split of LDC2015E86 and LDC2017T10. TREE is the tree-based GCNSEQ and GRAPH is the graph-based GCNSEQ.

# reentrancies	# dev sents.	# test sents.
0	619	622
1-5	679	679
6-20	70	70

Table 3: Counts of reentrancies for the development and test split of LDC2017T10

coders (GCNSEQ with reentrancies, henceforth called GRAPH) on the test set of LDC2015E86 and LDC2017T10. We also include state-of-the-art results reported on these datasets for sequential encoding (Konstas et al., 2017) and graph encoding (Song et al., 2018; Beck et al., 2018).[3] In order to mitigate the effects of random seeds, we train five models with different random seeds and report the results of the median model, according to their BLEU score on the development set (Beck et al., 2018). We achieve state-of-the-art results with both tree and graph encoders, demonstrating the efficacy of our GCNSeq approach. The graph encoder outperforms the other systems and previous work on both datasets. These results demonstrate the benefit of structural encoders over purely sequential ones as well as the advantage of explicitly including reentrancies. The differences between our graph encoder and that of Song et al. (2018) and Beck et al. (2018) were discussed in Section 3.3.

[3]We run comparisons on systems without ensembling nor additional data.

Model	Number of reentrancies		
	0	1-5	6-20
SEQ	42.94	31.64	23.33
TREE	+0.63	+1.41	+0.76
GRAPH	**+1.67**	**+1.54**	**+3.08**

Table 4: Differences, with respect to the sequential baseline, in the Meteor score of the test split of LDC2017T10 as a function of the number of reentrancies.

5.1 Reentrancies

Overall scores show an advantage of graph encoder over tree and sequential encoders, but they do not shed light into how this is achieved. Because graph encoders are the only ones to model reentrancies explicitly, we expect them to deal better with these structures. It is, however, possible that the other models are capable of handling these structures implicitly. Moreover, the dataset contains a large number of examples that do not involve any reentrancies, as shown in Table 3, so that the overall scores may not be representative of the ability of models to capture reentrancies. It is expected that the benefit of the graph models will be more evident for those examples containing more reentrancies. To test this hypothesis, we evaluate the various scenarios as a function of the number of reentrancies in each example, using the Meteor score as a metric.[4]

Table 4 shows that the gap between the graph encoder and the other encoders is widest for examples with more than six reentrancies. The Meteor score of the graph encoder for these cases is 3.1% higher than the one for the sequential encoder and 2.3% higher than the score achieved by the tree encoder, demonstrating that explicitly encoding reentrancies is more beneficial than the overall scores suggest. Interestingly, it can also be observed that the graph model outperforms the tree model also for examples with no reentrancies, where tree and graph structures are identical. This suggests that preserving reentrancies in the training data has other beneficial effects. In Section 5.2 we explore one: better handling of long-range dependencies.

[4]For this analysis we use Meteor instead of BLEU because it is a sentence-level metric, unlike BLEU, which is a corpus-level metric.

5.1.1 Manual Inspection

In order to further explore how the graph model handles reentrancies differently from the other models, we performed a manual inspection of the models' output. We selected examples containing reentrancies, where the graph model performs better than the other models. These are shown in Table 5. In Example (1), we note that the graph model is the only one that correctly predicts the phrase *he finds out*. The wrong verb tense is due to the lack of tense information in AMR graphs. In the sequential model, the pronoun is chosen correctly, but the wrong verb is predicted, while in the tree model the pronoun is missing. In Example (2), only the graph model correctly generates the phrase *you tell them*, while none of the models use *people* as the subject of the predicate *can*. In Example (3), both the graph and the sequential models deal well with the control structure caused by the *recommend* predicate. The sequential model, however, overgenerates a wh-clause. Finally, in Example (4) the tree and graph models deal correctly with the possessive pronoun to generate the phrase *tell your ex*, while the sequential model does not. Overall, we note that the graph model produces a more accurate output than sequential and tree models by generating the correct pronouns and mentions when control verbs and co-references are involved.

5.1.2 Contrastive Pairs

For a quantitative analysis of how the different models handle pronouns, we use a method to inspect NMT output for specific linguistic analysis based on contrastive pairs (Sennrich, 2017). Given a reference output sentence, a contrastive sentence is generated by introducing a mistake related to the phenomenon we are interested in evaluating. The probability that the model assigns to the reference sentence is then compared to that of the contrastive sentence. The accuracy of a model is determined by the percentage of examples in which the reference sentence has a higher probability than the contrastive sentence.

We produce contrastive examples by running CoreNLP (Manning et al., 2014) to identify coreferences, which are the primary cause of reentrancies, and introducing a mistake. When an expression has multiple mentions, the antecedent is repeated in the linearized AMR. For instance, the linearization of Figure 1(b) contains the token *he* twice, which instead appears only once in the sentence. This repetition may result in generating the token *he* twice, rather than using a pronoun to refer back to it. To investigate this possible mistake, we replace one of the mentions with the antecedent (e.g., *John ate the pizza with his fingers* is replaced with *John ate the pizza with John fingers*, which is ungrammatical and as such should be less likely).

An alternative hypothesis is that even when the generation system correctly decides to predict a pronoun, it selects the wrong one. To test for this, we produce contrastive examples where a pronoun is replaced by either a different type of pronoun (e.g., *John ate the pizza with his fingers* is replaced with *John ate the pizza with him fingers*) or by the same type of pronoun but for a different number (*John ate the pizza with their fingers*) or different gender (*John ate the pizza with her fingers*). Note from Figure 1 that the graph-structured AMR is the one that more directly captures the relation between *finger* and *he*, and as such it is expected to deal better with this type of mistakes.

From the test split of LDC2017T10, we generated 251 contrastive examples due to antecedent replacements, 912 due to pronoun type replacements, 1840 due to number replacements and 95 due to gender replacements.[5] The results are shown in Table 6. The sequential encoder performs surprisingly well at this task, with better or on par performance with respect to the tree encoder. The graph encoder outperforms the sequential encoder only for pronoun number and gender replacements. Future work is required to more precisely analyze if the different models cope with pronomial mentions in significantly different ways. Other approaches to inspect phenomena of co-reference and control verbs can also be explored, for instance by devising specific training objectives (Linzen et al., 2016).

5.2 Long-range Dependencies

When we encode a long sequence, interactions between items that appear distant from each other in the sequence are difficult to capture. The problem of long-range dependencies in natural language is well known for RNN architectures (Bengio et al., 1994). Indeed, the need to solve this problem motivated the introduction of LSTM models, which are known to model long-range dependencies better than traditional RNNs.

[5]The generated contrastive examples are available at https://github.com/mdtux89/OpenNMT-py.

(1)	REF	i dont tell him but **he finds out** ,
	SEQ	i did n't tell him but **he was out** .
	TREE	i do n't tell him but **found out** .
	GRAPH	i do n't tell him but **he found out** .
(2)	REF	if **you tell people** they can help you ,
	SEQ	if **you tell him** , you can help you !
	TREE	if **you tell person_name_0 you** , you can help you .
	GRAPH	if **you tell them** , you can help you .
(3)	REF	**i 'd recommend** you go and see your doctor too .
	SEQ	**i recommend** you go to see your doctor who is going to see your doctor .
	TREE	**you recommend** going to see your doctor too .
	GRAPH	**i recommend** you going to see your doctor too .
(4)	REF	(you) **tell your ex** that all communication needs to go through the lawyer .
	SEQ	(you) **tell** that all the communication go through lawyer .
	TREE	(you) **tell your ex** , tell your ex , the need for all the communication .
	GRAPH	(you) **tell your ex** the need to go through a lawyer .

Table 5: Examples of generation from AMR graphs containing reentrancies. REF is the reference sentence.

Model	Antec.	Type	Num.	Gender
SEQ	96.02	97.70	94.89	94.74
TREE	96.02	96.38	93.70	92.63
GRAPH	96.02	96.49	95.11	95.79

Table 6: Accuracy (%) of models, on the test split of LDC201T10, for different categories of contrastive errors: antecedent (Antec.), pronoun type (Type), number (Num.), and gender (Gender).

# max length	# dev sents.	# test sents.
0-10	292	307
11-50	350	297
51-250	21	18

Table 7: Counts of longest dependencies for the development and test split of LDC2017T10

Model	Max dependency length		
	0-10	11-50	51-200
SEQ	50.49	36.28	24.14
TREE	-0.48	+1.66	+2.37
GRAPH	**+1.22**	**+2.05**	**+3.04**

Table 8: Differences, with respect to the sequential baseline, in the Meteor score of the test split of LDC2017T10 as a function of the maximum dependency length.

Because the nodes in the graphs are not aligned with words in the sentence, AMR has no notion of distance between the nodes taking part in an edge. In order to define the length of an AMR edge, we resort to the AMR linearization discussed in Section 2. Given the linearization of the AMR $x_1, \ldots, x_N$, as discussed in Section 2, and an edge between two nodes x_i and x_j, the length of the edge is defined as $|j - i|$. For instance, in the AMR of Figure 1, the edge between *eat-01* and *:instrument* is a dependency of length five, because of the distance between the two words in the linearization *eat-01 :arg0 he :arg1 pizza :instrument*. We then compute the maximum dependency length for each AMR graph.

To verify the hypothesis that long-range dependencies contribute to the improvements of graph models, we compare the models as a function of the maximum dependency length in each example. Longer dependencies are sometimes caused by reentrancies, as in the dependency between *:part-of* and *he* in Figure 1. To verify that the contribution in terms of longer dependencies is complementary to that of reentrancies, we exclude sentences with reentrancies from this analysis. Table 7 shows the statistics for this measure. Results are shown in Table 8. The graph encoder always outperforms both the sequential and the tree encoder. The gap with the sequential encoder increases for longer dependencies. This indicates that longer dependencies are an important factor

in improving results for both tree and graph encoders, especially for the latter.

6 Conclusions

We introduced models for AMR-to-text generation with the purpose of investigating the difference between sequential, tree and graph encoders. We showed that encoding reentrancies improves overall performance. We observed bigger benefits when the input AMR graphs have a larger number of reentrant structures and longer dependencies. Our best graph encoder, which consists of a GCN wired to a BiLSTM network, improves over the state of the art on all tested datasets. We inspected the differences between the models, especially in terms of co-references and control structures. Further exploration of graph encoders is left to future work, which may result crucial to improve performance further.

Acknowledgments

The authors would like to thank the three anonymous reviewers and Adam Lopez, Ioannis Konstas, Diego Marcheggiani, Sorcha Gilroy, Sameer Bansal, Ida Szubert and Clara Vania for their help and comments. This research was supported by a grant from Bloomberg and by the H2020 project SUMMA, under grant agreement 688139.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of 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. *Linguistic Annotation Workshop*.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of EMNLP*.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. *Proceedings of ACL*.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Alberto Cetoli, Stefano Bragaglia, Andrew O'Harney, and Marc Sloan. 2017. Graph convolutional networks for named entity recognition. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of ACL*.

Marco Damonte, Shay B Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. *Proceedings of EACL*.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of NIPS*.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of ACL*.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of ICASSP*.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of ACL*.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. *Proceedings of ACL*.

Friedrich Wilhelm Levi. 1942. *Finite geometrical systems*. University of Calcutta.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A Smith. 2015. Toward abstractive summarization using semantic representations. *Proceedings of NAACL*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of ACL*.

Diego Marcheggiani and Laura Perez-Beltrachini. 2018. Deep graph convolutional encoders for structured data to text generation. *Proceedings of INLG*.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of EMNLP*.

Rik van Noord and Johan Bos. 2017. Dealing with coreference in neural semantic parsing. In *Proceedings of the 2nd Workshop on Semantic Deep Learning*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS Workshop*.

Rico Sennrich. 2017. How grammatical is character-level neural machine translation? assessing mt quality with contrastive translation pairs. In *Proceedings of EACL*.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for AMR-to-text generation. In *Proceedings of ACL*.

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.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *arXiv preprint arXiv:1505.00387*.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of ACL*.

Sho Takase, Jun Suzuki, Naoaki Okazaki, Tsutomu Hirao, and Masaaki Nagata. 2016. Neural headline generation on abstract meaning representation. In *Proceedings of EMNLP*.

Kun Xu, Lingfei Wu, Zhiguo Wang, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.