Graph-to-Sequence Learning using Gated Graph Neural Networks

Daniel Beck[†] **Gholamreza Haffari**[‡] **Trevor Cohn**[†]
[†]School of Computing and Information Systems
University of Melbourne, Australia
{d.beck,t.cohn}@unimelb.edu.au
[‡]Faculty of Information Technology
Monash University, Australia
gholamreza.haffari@monash.edu

Abstract

Many NLP applications can be framed as a graph-to-sequence learning problem. Previous work proposing neural architectures on this setting obtained promising results compared to grammar-based approaches but still rely on linearisation heuristics and/or standard recurrent networks to achieve the best performance. In this work, we propose a new model that encodes the full structural information contained in the graph. Our architecture couples the recently proposed Gated Graph Neural Networks with an input transformation that allows nodes and edges to have their own hidden representations, while tackling the parameter explosion problem present in previous work. Experimental results show that our model outperforms strong baselines in generation from AMR graphs and syntax-based neural machine translation.

1 Introduction

Graph structures are ubiquitous in representations of natural language. In particular, many whole-sentence semantic frameworks employ directed acyclic graphs as the underlying formalism, while most tree-based syntactic representations can also be seen as graphs. A range of NLP applications can be framed as the process of transducing a graph structure into a sequence. For instance, language generation may involve realising a semantic graph into a surface form and syntactic machine translation involves transforming a tree-annotated source sentence to its translation.

Previous work in this setting rely on grammar-based approaches such as tree transducers (Flanigan et al., 2016) and hyperedge replacement grammars (Jones et al., 2012). A key limitation of these approaches is that alignments between graph nodes and surface tokens are required. These alignments are usually automatically generated so they can propagate errors when building the grammar. More recent approaches transform the graph into a linearised form and use off-the-shelf methods such as phrase-based machine translation (Pourdamghani et al., 2016) or neural sequence-to-sequence (henceforth, `s2s`) models (Konstas et al., 2017). Such approaches ignore the full graph structure, discarding key information.

In this work we propose a model for graph-to-sequence (henceforth, `g2s`) learning that leverages recent advances in neural encoder-decoder architectures. Specifically, we employ an encoder based on Gated Graph Neural Networks (Li et al., 2016, GGNNs), which can incorporate the full graph structure without loss of information. Such networks represent edge information as label-wise parameters, which can be problematic even for small sized label vocabularies (in the order of hundreds). To address this limitation, we also introduce a graph transformation that changes edges to additional nodes, solving the parameter explosion problem. This also ensures that edges have graph-specific hidden vectors, which gives more information to the attention and decoding modules in the network.

We benchmark our model in two graph-to-sequence problems, generation from Abstract Meaning Representations (AMRs) and Neural Machine Translation (NMT) with source dependency information. Our approach outperforms strong `s2s` baselines in both tasks *without relying on standard RNN encoders*, in contrast with previous work. In particular, for NMT we show that we avoid the need for RNNs by adding sequential edges between contiguous words in the dependency tree. This illustrates the generality of our

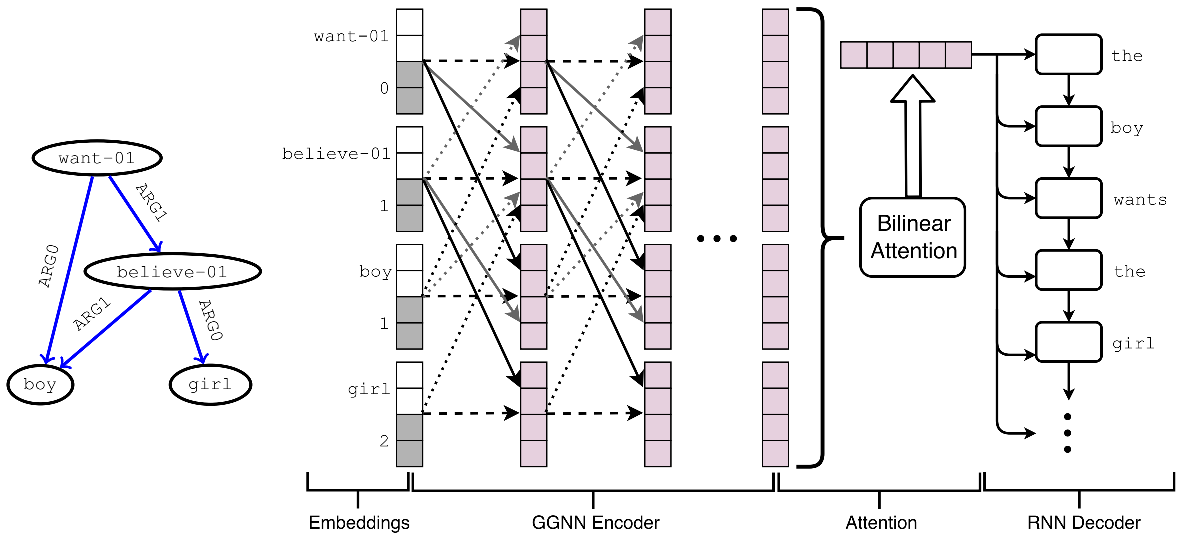

Figure 1: Left: the AMR graph representing the sentence "The boy wants the girl to believe him.". Right: Our proposed architecture using the same AMR graph as input and the surface form as output. The first layer is a concatenation of node and positional embeddings, using distance from the root node as the position. The GGNN encoder updates the embeddings using edge-wise parameters, represented by different colors (in this example, `ARG0` and `ARG1`). The encoder also add corresponding reverse edges (dotted arrows) and self edges for each node (dashed arrows). All parameters are shared between layers. Attention and decoder components are similar to standard `s2s` models. This is a pictorial representation: in our experiments the graphs are transformed before being used as inputs (see §3).

approach: linguistic biases can be added to the inputs by simple graph transformations, without the need for changes to the model architecture.

2 Neural Graph-to-Sequence Model

Our proposed architecture is shown in Figure 1, with an example AMR graph and its transformation into its surface form. Compared to standard `s2s` models, the main difference is in the encoder, where we employ a GGNN to build a graph representation. In the following we explain the components of this architecture in detail.[1]

2.1 Gated Graph Neural Networks

Early approaches for recurrent networks on graphs (Gori et al., 2005; Scarselli et al., 2009) assume a fixed point representation of the parameters and learn using contraction maps. Li et al. (2016) argues that this restricts the capacity of the model and makes it harder to learn long distance relations between nodes. To tackle these issues, they propose Gated Graph Neural Networks, which extend these architectures with gating mechanisms in a similar fashion to Gated Recurrent Units (Cho et al., 2014). This allows the network to be learnt via modern backpropagation procedures.

In following, we formally define the version of GGNNs we employ in this study. Assume a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, L_{\mathcal{V}}, L_{\mathcal{E}}\}$, where $\mathcal{V}$ is a set of nodes (v, ℓ_v), $\mathcal{E}$ is a set of edges (v_i, v_j, ℓ_e) and $L_{\mathcal{V}}$ and $L_{\mathcal{E}}$ are respectively vocabularies for nodes and edges, from which node and edge labels (ℓ_v and ℓ_e) are defined. Given an input graph with nodes mapped to embeddings $\mathbf{X}$, a GGNN is defined as

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

$$\mathbf{r}_v^t = \sigma\left(c_v^r \sum_{u \in \mathcal{N}_v} \mathbf{W}_{\ell_e}^r \mathbf{h}_u^{(t-1)} + \mathbf{b}_{\ell_e}^r \right)$$

$$\mathbf{z}_v^t = \sigma\left(c_v^z \sum_{u \in \mathcal{N}_v} \mathbf{W}_{\ell_e}^z \mathbf{h}_u^{(t-1)} + \mathbf{b}_{\ell_e}^z \right)$$

$$\widetilde{\mathbf{h}}_v^t = \rho\left(c_v \sum_{u \in \mathcal{N}_v} \mathbf{W}_{\ell_e} \left(\mathbf{r}_u^t \odot \mathbf{h}_u^{(t-1)} \right) + \mathbf{b}_{\ell_e} \right)$$

$$\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(i-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}}_v^t$$

where $e = (u, v, \ell_e)$ is the edge between nodes u and v, $\mathcal{N}(v)$ is the set of neighbour nodes for v, ρ is a non-linear function, σ is the sigmoid function

[1]Our implementation uses MXNet (Chen et al., 2015) and is based on the Sockeye toolkit (Hieber et al., 2017). Code is available at `github.com/beckdaniel/acl2018_graph2seq`.

and $c_v = c_v^z = c_v^r = |\mathcal{N}_v|^{-1}$ are normalisation constants.

Our formulation differs from the original GGNNs from Li et al. (2016) in some aspects: 1) we add bias vectors for the hidden state, reset gate and update gate computations; 2) label-specific matrices do not share any components; 3) reset gates are applied to all hidden states before any computation and 4) we add normalisation constants. These modifications were applied based on preliminary experiments and ease of implementation.

An alternative to GGNNs is the model from Marcheggiani and Titov (2017), which add edge label information to Graph Convolutional Networks (GCNs). According to Li et al. (2016), the main difference between GCNs and GGNNs is analogous to the difference between convolutional and recurrent networks. More specifically, GGNNs can be seen as multi-layered GCNs where layer-wise parameters are tied and gating mechanisms are added. A large number of layers can propagate node information between longer distances in the graph and, unlike GCNs, GGNNs can have an arbitrary number of layers without increasing the number of parameters. Nevertheless, our architecture borrows ideas from GCNs as well, such as normalising factors.

2.2 Using GGNNs in attentional encoder-decoder models

In s2s models, inputs are sequences of tokens where each token is represented by an embedding vector. The encoder then transforms these vectors into hidden states by incorporating context, usually through a recurrent or a convolutional network. These are fed into an attention mechanism, generating a single context vector that informs decisions in the decoder.

Our model follows a similar structure, where the encoder is a GGNN that receives *node* embeddings as inputs and generates *node* hidden states as outputs, using the graph structure as context. This is shown in the example of Figure 1, where we have 4 hidden vectors, one per node in the AMR graph. The attention and decoder components follow similar standard s2s models, where we use a bilinear attention mechanism (Luong et al., 2015) and a 2-layered LSTM (Hochreiter and Schmidhuber, 1997) as the decoder. Note, however, that other decoders and attention mechanisms can be easily employed instead. Bastings et al. (2017) employs a similar idea for syntax-based NMT, but using GCNs instead.

2.3 Bidirectionality and positional embeddings

While our architecture can in theory be used with general graphs, rooted directed acyclic graphs (DAGs) are arguably the most common kind in the problems we are addressing. This means that node embedding information is propagated in a top down manner. However, it is desirable to have information flow from the reverse direction as well, in the same way RNN-based encoders benefit from right-to-left propagation (as in bidirectional RNNs). Marcheggiani and Titov (2017) and Bastings et al. (2017) achieve this by adding reverse edges to the graph, as well as self-loops edges for each node. These extra edges have specific labels, hence their own parameters in the network.

In this work, we also follow this procedure to ensure information is evenly propagated in the graph. However, this raises another limitation: because the graph becomes essentially undirected, the encoder is now unaware of any intrinsic hierarchy present in the input. Inspired by Gehring et al. (2017) and Vaswani et al. (2017), we tackle this problem by adding positional embeddings to every node. These embeddings are indexed by integer values representing the minimum distance from the root node and are learned as model parameters.[2] This kind of positional embedding is restricted to rooted DAGs: for general graphs, different notions of distance could be employed.

3 Levi Graph Transformation

The g2s model proposed in §2 has two key deficiencies. First, GGNNs have three linear transformations *per edge type*. This means that the number of parameters can explode: AMR, for instance, has around 100 different predicates, which correspond to edge labels. Previous work deal with this problem by explicitly grouping edge labels into a single one (Marcheggiani and Titov, 2017; Bastings et al., 2017) but this is not an ideal solution since it incurs in loss of information.

[2]Vaswani et al. (2017) also proposed fixed positional embeddings based on sine and cosine wavelengths. Preliminary experiments showed that this approach did not work in our case: we speculate this is because wavelengths are more suitable to sequential inputs.

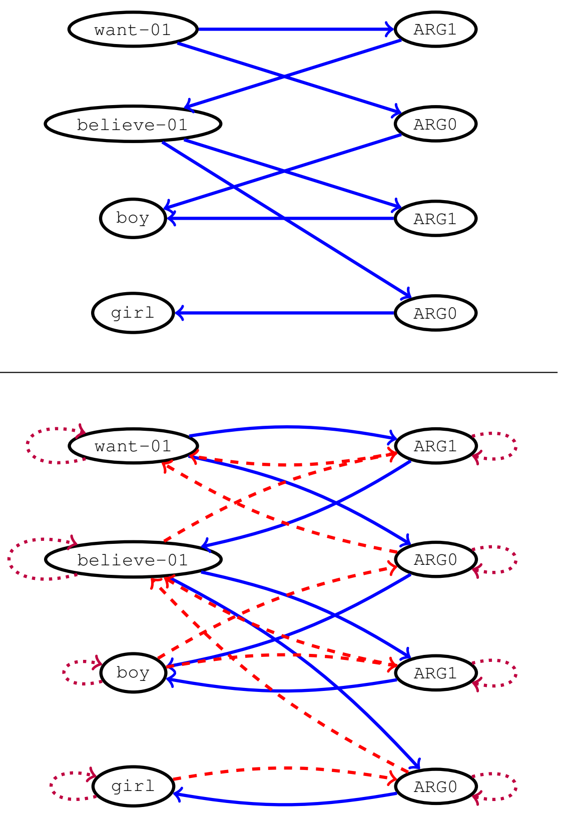

Figure 2: Top: the AMR graph from Figure 1 transformed into its corresponding Levi graph. Bottom: Levi graph with added reverse and self edges (colors represent different edge labels).

The second deficiency is that edge label information is encoded in the form of GGNN parameters in the network. This means that each label will have the same "representation" across all graphs. However, the latent information in edges can depend on the content in which they appear in a graph. Ideally, edges should have instance-specific *hidden states*, in the same way as nodes, and these should also inform decisions made in the decoder through the attention module. For instance, in the AMR graph shown in Figure 1, the `ARG1` predicate between `want-01` and `believe-01` can be interpreted as the preposition "to" in the surface form, while the `ARG1` predicate connecting `believe-01` and `boy` is realised as a pronoun. Notice that edge hidden vectors are already present in `s2s` networks that use linearised graphs: we would like our architecture to also have this benefit.

Instead of modifying the architecture, we propose to transform the input graph into its equivalent Levi graph (Levi, 1942; Gross and Yellen, 2004, p. 765). Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, L_\mathcal{V}, L_\mathcal{E}\}$, a Levi graph[3] is defined as $\mathcal{G} = \{\mathcal{V}', \mathcal{E}', L_{\mathcal{V}'}, L_{\mathcal{E}'}\}$, where $\mathcal{V}' = \mathcal{V} \cup \mathcal{E}$, $L_{\mathcal{V}'} = L_\mathcal{V} \cup L_\mathcal{E}$ and $L_{\mathcal{E}'} = \varnothing$. The new edge set $\mathcal{E}'$ contains a edge for every (node, edge) pair that is present in the original graph. By definition, the Levi graph is bipartite.

Intuitively, transforming a graph into its Levi graph equivalent turns edges into additional nodes. While simple in theory, this transformation addresses both modelling deficiencies mentioned above in an elegant way. Since the Levi graph has no labelled edges there is no risk of parameter explosion: original edge labels are represented as embeddings, in the same way as nodes. Furthermore, the encoder now naturally generates hidden states for original edges as well.

In practice, we follow the procedure in §2.3 and add reverse and self-loop edges to the Levi graph, so the practical edge label vocabulary is $L_{\mathcal{E}'} = \{\text{default}, \text{reverse}, \text{self}\}$. This still keeps the parameter space modest since we have only three labels. Figure 2 shows the transformation steps in detail, applied to the AMR graph shown in Figure 1. Notice that the transformed graphs are the ones fed into our architecture: we show the original graph in Figure 1 for simplicity.

It is important to note that this transformation can be applied to any graph and therefore is independent of the model architecture. We speculate this can be beneficial in other kinds of graph-based encoder such as GCNs and leave further investigation to future work.

4 Generation from AMR Graphs

Our first `g2s` benchmark is language generation from AMR, a semantic formalism that represents sentences as rooted DAGs (Banarescu et al., 2013). Because AMR abstracts away from syntax, graphs do not have gold-standard alignment information, so generation is not a trivial task. Therefore, we hypothesize that our proposed model is ideal for this problem.

4.1 Experimental setup

Data and preprocessing We use the latest AMR corpus release (LDC2017T10) with the default split of 36521/1368/1371 instances for training,

[3]Formally, a Levi graph is defined over any *incidence structure*, which is a general concept usually considered in a geometrical context. Graphs are an example of incidence structures but so are points and lines in the Euclidean space, for instance.

development and test sets. Each graph is preprocessed using a procedure similar to what is performed by Konstas et al. (2017), which includes entity simplification and anonymisation. This preprocessing is done before transforming the graph into its Levi graph equivalent. For the `s2s` baselines, we also add scope markers as in Konstas et al. (2017). We detail these procedures in the Appendix.

Models Our baselines are attentional `s2s` models which take linearised graphs as inputs. The architecture is similar to the one used in Konstas et al. (2017) for AMR generation, where the encoder is a BiLSTM followed by a unidirectional LSTM. All dimensionalities are fixed to 512.

For the `g2s` models, we fix the number of layers in the GGNN encoder to 8, as this gave the best results on the development set. Dimensionalities are also fixed at 512 except for the GGNN encoder which uses 576. This is to ensure all models have a comparable number of parameters and therefore similar capacity.

Training for all models uses Adam (Kingma and Ba, 2015) with 0.0003 initial learning rate and 16 as the batch size.[4] To regularise our models we perform early stopping on the dev set based on perplexity and apply 0.5 dropout (Srivastava et al., 2014) on the source embeddings. We detail additional model and training hyperparameters in the Appendix.

Evaluation Following previous work, we evaluate our models using BLEU (Papineni et al., 2001) and perform bootstrap resampling to check statistical significance. However, since recent work has questioned the effectiveness of BLEU with bootstrap resampling (Graham et al., 2014), we also report results using sentence-level CHRF++ (Popović, 2017), using the Wilcoxon signed-rank test to check significance. Evaluation is case-insensitive for both metrics.

Recent work has shown that evaluation in neural models can lead to wrong conclusions by just changing the random seed (Reimers and Gurevych, 2017). In an effort to make our conclusions more robust, we run each model 5 times using different seeds. From each pool, we report results using the median model according to performance on the dev set (simulating what is expected from a single run) and using an ensemble of the 5 models.

Finally, we also report the number of parameters used in each model. Since our encoder architectures are quite different, we try to match the number of parameters between them by changing the dimensionality of the hidden layers (as explained above). We do this to minimise the effects of model capacity as a confounder.

[4]Larger batch sizes hurt dev performance in our preliminary experiments. There is evidence that small batches can lead to better generalisation performance (Keskar et al., 2017). While this can make training time slower, it was doable in our case since the dataset is small.

	BLEU	CHRF++	#params
Single models			
`s2s`	21.7	49.1	28.4M
`s2s` (-s)	18.4	46.3	28.4M
`g2s`	23.3	50.4	28.3M
Ensembles			
`s2s`	26.6	52.5	142M
`s2s` (-s)	22.0	48.9	142M
`g2s`	**27.5**	**53.5**	141M
Previous work (early AMR treebank versions)			
KIYCZ17	22.0	–	–
Previous work (as above + unlabelled data)			
KIYCZ17	33.8	–	–
PKH16	26.9	–	–
SPZWG17	25.6	–	–
FDSC16	22.0	–	–

Table 1: Results for AMR generation on the test set. All score differences between our models and the corresponding baselines are significantly different ($p<0.05$). "(-s)" means input without scope marking. KIYCZ17, PKH16, SPZWG17 and FDSC16 are respectively the results reported in Konstas et al. (2017), Pourdamghani et al. (2016), Song et al. (2017) and Flanigan et al. (2016).

4.2 Results and analysis

Table 1 shows the results on the test set. For the `s2s` models, we also report results without the scope marking procedure of Konstas et al. (2017). Our approach significantly outperforms the `s2s` baselines both with individual models and ensembles, while using a comparable number of parameters. In particular, we obtain these results without relying on scoping heuristics.

On Figure 3 we show an example where our model outperforms the baseline. The AMR graph contains four reentrancies, predicates that refer-

Original AMR graph

```
(p / propose-01
  :ARG0 (c / country
   :wiki "Russia"
   :name (n / name
    :op1 "Russia"))
  :ARG1 (c5 / cooperate-01
   :ARG0 c
   :ARG1 (a / and
    :op1 (c2 / country
     :wiki "India"
     :name (n2 / name
      :op1 "India"))
    :op2 (c3 / country
     :wiki "China"
     :name (n3 / name
      :op1 "China"))))
  :purpose (i / increase-01
   :ARG0 c5
   :ARG1 (s / security)
   :location (a2 / around
    :op1 (c4 / country
     :wiki "Afghanistan"
     :name (n4 / name
      :op1 "Afghanistan")))
   :purpose (b / block-01
    :ARG0 (a3 / and
     :op1 c :op2 c2 :op3 c3
    :ARG1 (s2 / supply-01
     :ARG1 (d / drug)))))
```

Reference surface form

Russia proposes cooperation with **India and China** to increase security around Afghanistan to block drug supplies.

s2s output (CHRF++ 61.8)

Russia proposed cooperation with **India and China** to increase security around the Afghanistan to block security around the Afghanistan , **India and China**.

g2s output (CHRF++ 78.2)

Russia proposed cooperation with **India and China** to increase security around Afghanistan to block drug supplies.

Figure 3: Example showing overgeneration due to reentrancies. Top: original AMR graph with key reentrancies highlighted. Bottom: reference and outputs generated by the s2s and g2s models, highlighting the overgeneration phenomena.

ence previously defined concepts in the graph. In the s2s models including Konstas et al. (2017), reentrant nodes are copied in the linearised form, while this is not necessary for our g2s models. We can see that the s2s prediction overgenerates the "India and China" phrase. The g2s prediction avoids overgeneration, and almost perfectly matches the reference. While this is only a single example, it provides evidence that retaining the full graphical structure is beneficial for this task, which is corroborated by our quantitative results.

Table 1 also show BLEU scores reported in previous work. These results are not strictly comparable because they used different training set versions and/or employ additional unlabelled corpora; nonetheless some insights can be made. In particular, our g2s ensemble performs better than many previous models that combine a smaller training set with a large unlabelled corpus. It is also most informative to compare our s2s model with Konstas et al. (2017), since this baseline is very similar to theirs. We expected our single model baseline to outperform theirs since we use a larger training set but we obtained similar performance. We speculate that better results could be obtained by more careful tuning, but nevertheless we believe such tuning would also benefit our proposed g2s architecture.

The best results with unlabelled data are obtained by Konstas et al. (2017) using Gigaword sentences as additional data and a paired trained procedure with an AMR parser. It is important to note that this procedure is orthogonal to the individual models used for generation and parsing. Therefore, we hypothesise that our model can also benefit from such techniques, an avenue that we leave for future work.

5 Syntax-based Neural Machine Translation

Our second evaluation is NMT, using as graphs source language dependency syntax trees. We focus on a medium resource scenario where additional linguistic information tends to be more beneficial. Our experiments comprise two language pairs: English-German and English-Czech.

5.1 Experimental setup

Data and preprocessing We employ the same data and settings from Bastings et al. (2017),[5] which use the News Commentary V11 corpora from the WMT16 translation task.[6] English text is tokenised and parsed using SyntaxNet[7] while German and Czech texts are tokenised and split into subwords using byte-pair encodings (Sennrich et al., 2016, BPE) (8000 merge operations).

[5]We obtained the data from the original authors to ensure results are comparable without any influence from preprocessing steps.

[6]http://www.statmt.org/wmt16/translation-task.html

[7]https://github.com/tensorflow/models/tree/master/syntaxnet

We refer to Bastings et al. (2017) for further information on the preprocessing steps.

Labelled dependency trees in the source side are transformed into Levi graphs as a preprocessing step. However, unlike AMR generation, in NMT the inputs are originally surface forms that contain important sequential information. This information is lost when treating the input as dependency trees, which might explain why Bastings et al. (2017) obtain the best performance when using an initial RNN layer in their encoder. To investigate this phenomenon, we also perform experiments adding sequential connections to each word in the dependency tree, corresponding to their order in the original surface form (henceforth, `g2s+`). These connections are represented as edges with specific left and right labels, which are added after the Levi graph transformation. Figure 4 shows an example of an input graph for `g2s+`, with the additional sequential edges connecting the words (reverse and self edges are omitted for simplicity).

Models Our `s2s` and `g2s` models are almost the same as in the AMR generation experiments (§4.1). The only exception is the GGNN encoder dimensionality, where we use 512 for the experiments with dependency trees only and 448 when the inputs have additional sequential connections. As in the AMR generation setting, we do this to ensure model capacity are comparable in the number of parameters. Another key difference is that the `s2s` baselines do not use dependency trees: they are trained on the sentences only.

In addition to neural models, we also report results for Phrase-Based Statistical MT (PB-SMT), using Moses (Koehn et al., 2007). The PB-SMT models are trained using the same data conditions as `s2s` (no dependency trees) and use the standard setup in Moses, except for the language model, where we use a 5-gram LM trained on the target side of the respective parallel corpus.[8]

Evaluation We report results in terms of BLEU and CHRF++, using case-sensitive versions of both metrics. Other settings are kept the same as in the AMR generation experiments (§4.1). For PB-SMT, we also report the median result of 5 runs, obtained by tuning the model using MERT (Och and Ney, 2002) 5 times.

[8]Note that target data is segmented using BPE, which is not the usual setting for PB-SMT. We decided to keep the segmentation to ensure data conditions are the same.

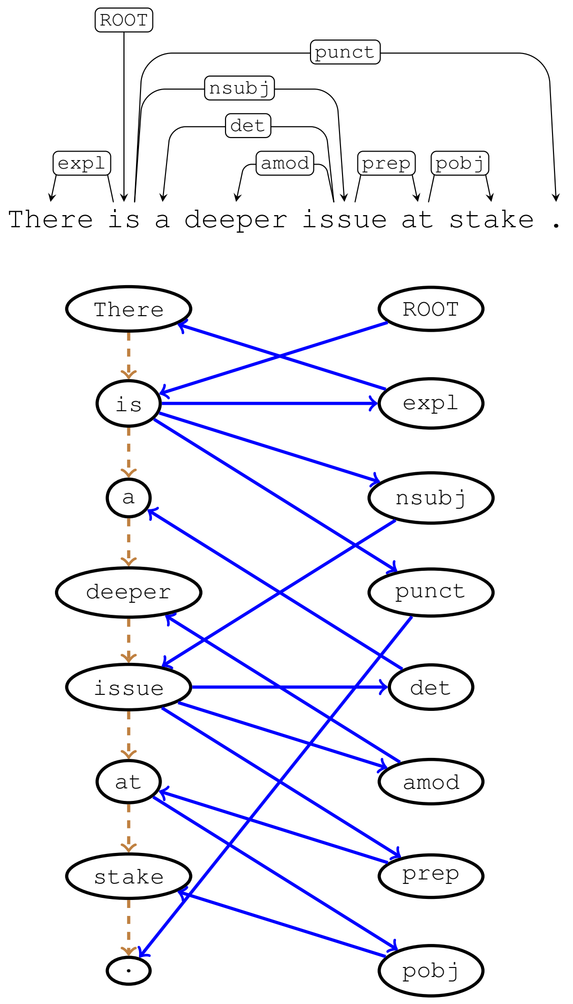

Figure 4: Top: a sentence with its corresponding dependency tree. Bottom: the transformed tree into a Levi graph with additional sequential connections between words (dashed lines). The full graph also contains reverse and self edges, which are omitted in the figure.

5.2 Results and analysis

Table 2 shows the results on the respective test set for both language pairs. The `g2s` models, which do not account for sequential information, lag behind our baselines. This is in line with the findings of Bastings et al. (2017), who found that having a BiRNN layer was key to obtain the best results. However, the `g2s+` models outperform the baselines in terms of BLEU scores under the same parameter budget, in both single model and ensemble scenarios. This result show that it is possible to incorporate sequential biases in our model without relying on RNNs or any other modification in the architecture.

English-German			
	BLEU	CHRF++	#params
Single models			
PB-SMT	12.8	43.2	–
s2s	15.5	40.8	41.4M
g2s	15.2	41.4	40.8M
g2s+	16.7	42.4	41.2M
Ensembles			
s2s	19.0	44.1	207M
g2s	17.7	43.5	204M
g2s+	**19.6**	**45.1**	206M
Results from (Bastings et al., 2017)			
BoW+GCN	12.2	–	–
BiRNN	14.9	–	–
BiRNN+GCN	16.1	–	–
English-Czech			
	BLEU	CHRF++	#params
Single models			
PB-SMT	8.6	**36.4**	–
s2s	8.9	33.8	39.1M
g2s	8.7	32.3	38.4M
g2s+	9.8	33.3	38.8M
Ensembles			
s2s	11.3	**36.4**	195M
g2s	10.4	34.7	192M
g2s+	**11.7**	35.9	194M
Results from (Bastings et al., 2017)			
BoW+GCN	7.5	–	–
BiRNN	8.9	–	–
BiRNN+GCN	9.6	–	–

Table 2: Results for syntax-based NMT on the test sets. All score differences between our models and the corresponding baselines are significantly different ($p<0.05$), including the negative CHRF++ result for En-Cs.

Interestingly, we found different trends when analysing the CHRF++ numbers. In particular, this metric favours the PB-SMT models for both language pairs, while also showing improved performance for s2s in En-Cs. CHRF++ has been shown to better correlate with human judgments compared to BLEU, both at system and sentence level for both language pairs (Bojar et al., 2017), which motivated our choice as an additional metric. We leave further investigation of this phenomena for future work.

We also show some of the results reported by Bastings et al. (2017) in Table 2. Note that their results were based on a different implementation, which may explain some variation in performance. Their BoW+GCN model is the most similar to ours, as it uses only an embedding layer and a GCN encoder. We can see that even our simpler g2s model outperforms their results. A key difference between their approach and ours is the Levi graph transformation and the resulting hidden vectors for edges. We believe their architecture would also benefit from our proposed transformation. In terms of baselines, s2s performs better than their BiRNN model for En-De and comparably for En-Cs, which corroborates that our baselines are strong ones. Finally, our g2s+ single models outperform their BiRNN+GCN results, in particular for En-De, which is further evidence that RNNs are not necessary for obtaining the best performance in this setting.

An important point about these experiments is that we did not tune the architecture: we simply employed the same model we used in the AMR generation experiments, only adjusting the dimensionality of the encoder to match the capacity of the baselines. We speculate that even better results would be obtained by tuning the architecture to this task. Nevertheless, we still obtained improved performance over our baselines and previous work, underlining the generality of our architecture.

6 Related work

Graph-to-sequence modelling Early NLP approaches for this problem were based on Hyperedge Replacement Grammars (Drewes et al., 1997, HRGs). These grammars assume the transduction problem can be split into rules that map portions of a graph to a set of tokens in the output sequence. In particular, Chiang et al. (2013) defines a parsing algorithm, followed by a complexity analysis, while Jones et al. (2012) report experiments on semantic-based machine translation using HRGs. HRGs were also used in previous work on AMR parsing (Peng et al., 2015). The main drawback of these grammar-based approaches though is the need for alignments between graph nodes and surface tokens, which are usually not available in gold-standard form.

Neural networks for graphs Recurrent networks on general graphs were first proposed under the name Graph Neural Networks (Gori et al., 2005; Scarselli et al., 2009). Our work is based on the architecture proposed by Li et al. (2016), which add gating mechanisms. The main difference between their work and ours is that they focus on problems that concern the input graph itself such as node classification or path finding while we focus on generating strings. The main alternative for neural-based graph representations is Graph Convolutional Networks (Bruna et al., 2014; Duvenaud et al., 2015; Kipf and Welling, 2017), which have been applied in a range of problems. In NLP, Marcheggiani and Titov (2017) use a similar architecture for Semantic Role Labelling. They use heuristics to mitigate the parameter explosion by grouping edge labels, while we keep the original labels through our Levi graph transformation. An interesting alternative is proposed by Schlichtkrull et al. (2017), which uses tensor factorisation to reduce the number of parameters.

Applications Early work on AMR generation employs grammars and transducers (Flanigan et al., 2016; Song et al., 2017). Linearisation approaches include (Pourdamghani et al., 2016) and (Konstas et al., 2017), which showed that graph simplification and anonymisation are key to good performance, a procedure we also employ in our work. However, compared to our approach, linearisation incurs in loss of information. MT has a long history of previous work that aims at incorporating syntax (Wu, 1997; Yamada and Knight, 2001; Galley et al., 2004; Liu et al., 2006, inter alia). This idea has also been investigated in the context of NMT. Bastings et al. (2017) is the most similar work to ours, and we benchmark against their approach in our NMT experiments. Eriguchi et al. (2016) also employs source syntax, but using constituency trees instead. Other approaches have investigated the use of syntax in the target language (Aharoni and Goldberg, 2017; Eriguchi et al., 2017). Finally, Hashimoto and Tsuruoka (2017) treats source syntax as a latent variable, which can be pretrained using annotated data.

7 Discussion and Conclusion

We proposed a novel encoder-decoder architecture for graph-to-sequence learning, outperforming baselines in two NLP tasks: generation from AMR graphs and syntax-based NMT. Our approach addresses shortcomings from previous work, including loss of information from linearisation and parameter explosion. In particular, we showed how graph transformations can solve issues with graph-based networks without changing the underlying architecture. This is the case of the proposed Levi graph transformation, which ensures the decoder can attend to edges as well as nodes, but also to the sequential connections added to the dependency trees in the case of NMT. Overall, because our architecture can work with general graphs, it is straightforward to add linguistic biases in the form of extra node and/or edge information. We believe this is an interesting research direction in terms of applications.

Our architecture nevertheless has two major limitations. The first one is that GGNNs have a fixed number of layers, even though graphs can vary in size in terms of number of nodes and edges. A better approach would be to allow the encoder to have a dynamic number of layers, possibly based on the diameter (longest path) in the input graph. The second limitation comes from the Levi graph transformation: because edge labels are represented as nodes they end up sharing the vocabulary and therefore, the same semantic space. This is not ideal, as nodes and edges are different entities. An interesting alternative is Weave Module Networks (Kearnes et al., 2016), which explicitly decouples node and edge representations without incurring in parameter explosion. Incorporating both ideas to our architecture is an research direction we plan for future work.

Acknowledgements

This work was supported by the Australian Research Council (DP160102686). The research reported in this paper was partly conducted at the 2017 Frederick Jelinek Memorial Summer Workshop on Speech and Language Technologies, hosted at Carnegie Mellon University and sponsored by Johns Hopkins University with unrestricted gifts from Amazon, Apple, Facebook, Google, and Microsoft. The authors would also like to thank Joost Bastings for sharing the data from his paper's experiments.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards String-to-Tree Neural Machine Translation. In *Proceedings of ACL*. pages 132–140.

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.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph Convolutional Encoders for Syntax-aware Neural Machine Translation. In *Proceedings of EMNLP*. pages 1947–1957.

Ondej Bojar, Yvette Graham, and Amir Kamran. 2017. Results of the WMT17 Metrics Shared Task. In *Proceedings of WMT*. volume 2, pages 293–301.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *Proceedings of ICLR*. page 14.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. In *Proceedings of the Workshop on Machine Learning Systems*. pages 1–6.

David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing Graphs with Hyperedge Replacement Grammars. In *Proceedings of ACL*. pages 924–932.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of EMNLP*. pages 1724–1734.

Frank Drewes, Hans Jörg Kreowski, and Annegret Habel. 1997. Hyperedge Replacement Graph Grammars. *Handbook of Graph Grammars and Computing by Graph Transformation* .

David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Proceedings of NIPS*. pages 2215–2223.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-Sequence Attentional Neural Machine Translation. In *Proceedings of ACL*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to Parse and Translate Improves Neural Machine Translation. In *Proceedings of ACL*.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Generation from Abstract Meaning Representation using Tree Transducers. In *Proceedings of NAACL*. pages 731–739.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah a Smith. 2014. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of ACL*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of NAACL*. pages 273–280.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. *arXiv preprint* .

Xavier Glorot and Yoshua Bengio. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of AISTATS*. volume 9, pages 249–256.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A New Model for Learning in Graph Domains. In *Proceedings of IJCNN*. volume 2, pages 729–734.

Yvette Graham, Nitika Mathur, and Timothy Baldwin. 2014. Randomized Significance Tests in Machine Translation. In *Proceedings of WMT*. pages 266–274.

Jonathan Gross and Jay Yellen, editors. 2004. *Handbook of Graph Theory*. CRC Press.

Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural Machine Translation with Source-Side Latent Graph Parsing. In *Proceedings of EMNLP*. pages 125–135.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A Toolkit for Neural Machine Translation. *arXiv preprint* pages 1–18.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-Based Machine Translation with Hyperedge Replacement Grammars. In *Proceedings of COLING*. pages 1359–1376.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. 2016. Molecular Graph Convolutions: Moving Beyond Fingerprints. *Journal of Computer-Aided Molecular Design* 30(8):595–608.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *Proceedings of ICLR*. pages 1–16.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR*. pages 1–15.

Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of ICLR*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demo Session*. pages 177–180.

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 ACL*. pages 146–157.

Friedrich Wilhelm Levi. 1942. Finite Geometrical Systems.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *Proceedings of ICLR*. 1, pages 1–20.

Yang Liu, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL - ACL '06*. pages 609–616.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of EMNLP*. pages 1412–1421.

Diego Marcheggiani and Ivan Titov. 2017. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In *Proceedings of EMNLP*.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*. page 295. https://doi.org/10.3115/1073083.1073133.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*. pages 311–318.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A Synchronous Hyperedge Replacement Grammar based approach for AMR parsing. In *Proceedings of CoNLL*. pages 32–41.

Maja Popović. 2017. chrF ++: words helping character n-grams. In *Proceedings of WMT*. pages 612–618.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English Strings with Abstract Meaning Representation Graphs. In *Proceedings of EMNLP*. pages 425–429.

Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. 2016. Generating English from Abstract Meaning Representations. In *Proceedings of INLG*. volume 0, pages 21–25.

Nils Reimers and Iryna Gurevych. 2017. Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In *Proceedings of EMNLP*. pages 338–348.

Franco Scarselli, Marco Gori, Ah Ching Tsoi, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20(1):61–80.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling Relational Data with Graph Convolutional Networks pages 1–12.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of ACL*. pages 1715–1725.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text Generation with Synchronous Node Replacement Grammar. In *Proceedings of ACL*. pages 7–13.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15:1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Proceedings of NIPS*.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics* 23(3):377–403.

Kenji Yamada and Kevin Knight. 2001. A Syntax-based Statistical Translation Model. In *Proceedings of ACL*. pages 523–530.

A Simplification and Anonymisation for AMR graphs

The procedure for graph simplification and anonymisation is similar to what is done by (Konstas et al., 2017). The main difference is that we use the alignments provided by the original LDC version of the AMR corpus, while they use a combination of the JAMR aligner (Flanigan et al., 2014) and the unsupervised aligner of (Pourdamghani et al., 2014). This preprocessing is done before transforming the graph into its bipartite equivalent.

Simplification: we remove sense information from the concepts. For instance, `believe-01` becomes `believe`. We also remove any subgraphs related to wikification (the ones starting the predicate `:wiki`).

Entity anonymisation: all subgraphs starting with the predicate `:name` are anonymised. The predicate contains one of the AMR entity names as the source node, such as `country` or `*-quantity`. These are replaced by a single anonymised node, containing the original entity concept plus an index. At training time, we use the alignment information to find the corresponding entity in the surface form and replace all aligned tokens with the same concept name. At test time, we extract a map that maps the anonymised entity to all concept names in the subgraph, in depth-first order. After prediction, if there is an anonymised token in the surface form it is replaced using that map (as long as the predicted token is present in the map).

Enitity clustering: entities are also clustered into four coarse-grained types in both graph and surface form. For instance `country_0` becomes `loc_0`. We use the list obtained from the open source implementation of (Konstas et al., 2017) for that.

Date anonymisation: if there is a `date-entity` concept, all underlying concepts are also anonymised, using separate tokens for day, month and year. At training time, the surface form is also anonymised following the alignment, but we additionally split days and months into `day_name`, `day_number`, `month_name` and `month_number`. At test time, we render the day/month according to the predicted anonymised token in the surface form and the recorded map.

B Model Hyperparameters

Our implementation is based on the Sockeye toolkit for Neural Machine Translation (Hieber et al., 2017). Besides the specific hyperparameter values mentioned in the paper, all other hyperparameters are set to the default values in Sockeye. We detail them here for completeness:

B.1 Vocabulary

- For AMR, we set the minimum frequency to 2 in both source nodes and target surface tokens. For NMT, we also use 2 as the minimum frequency in the source but use 1 in the target since we use BPE tokens.

B.2 Model

- The baseline encoder use a BiLSTM followed by a unidirectional LSTM. The decoder in all models use a 2-layer LSTM.

- The attention module uses a bilinear scoring function (*general* as in (Luong et al., 2015)).

- The max sequence length during training is 200 for AMR. In NMT, we use 100 for the `s2s` baselines and 200 for the `g2s` models. This is because we do not use dependency trees in the baseline so it naturally has half of the tokens.

- All dimensionalities are fixed to 512, which is similar to what is used by (Konstas et al., 2017). The only exceptions are the GGNN hidden state dimensionality in the `g2s` models. We use 576 for the `g2s` models used in the AMR experiments and 448 for the `g2s+` models used in the NMT experiments. As pointed out in the main paper, we change GGNN dimensionalities in order to have a similar parameter budget compared to the `s2s` baselines.

B.3 Training

These options apply for both `s2s` baselines and `g2s` `g2s+` models.

- We use 16 as the batch size. This is a lower number than most previous work we compare with: we choose this because we obtained better results in the AMR dev set. This is in line with recent evidence showing that smaller batch sizes lead to better generalisation performance (Keskar et al., 2017). The drawback is that smaller batches makes training time slower. However, this was not a problem in our experiments due to the medium size of the datasets.

- Bucketing is used to speed up training: we use 10 as the bucket size.

- Models are trained using cross-entropy as the loss.
- We save parameter checkpoints at every full epoch on the training set.
- We use early stopping by perplexity on the dev set with patience 8 (training stops if dev perplexity do not improve for 8 checkpoints).
- A maximum of 30 epochs/checkpoints is used. All our models stopped training before reaching this limit.
- We use 0.5 dropout on the input embeddings, before they are fed to the encoder.
- Weigths are initalised using Xavier initialisation (Glorot and Bengio, 2010), with except of forget biases in the LSTMs which are initialised by 0.
- We use Adam (Kingma and Ba, 2015) as the optimiser with 0.0003 as the initial learning rate.
- Learning rate is halved every time dev perplexity does not improve for 3 epochs/checkpoints.
- Gradient clipping is set to 1.0.

B.4 Decoding

- We use beam search to decode, using 5 as the beam size.
- Ensembles are created by averaging log probabilities at every step (*linear* in Sockeye). Attention scores are averaged over the 5 models at every step.
- For AMR only, we replace `<unk>` tokens in the prediction with the node with the highest attention score at that step, in the same way described by (Konstas et al., 2017). This is done before deanonymisation.