Deep Graph Convolutional Encoders for Structured Data to Text Generation

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Proceedings of the 11th International Conference on Natural Language Generation

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Deep Graph Convolutional Encoders for Structured Data to Text Generation

Diego Marcheggiani\textsuperscript{1,2} \quad Laura Perez-Beltrachini\textsuperscript{1}

\textsuperscript{1}ILCC, School of Informatics, University of Edinburgh \quad \textsuperscript{2}ILLC, University of Amsterdam

marcheggiani@uva.nl \quad lperez@inf.ed.ac.uk

Abstract

Most previous work on neural text generation from graph-structured data relies on standard sequence-to-sequence methods. These approaches linearise the input graph to be fed to a recurrent neural network. In this paper, we propose an alternative encoder based on graph convolutional networks that directly exploits the input structure. We report results on two graph-to-sequence datasets that empirically show the benefits of explicitly encoding the input graph structure.\textsuperscript{1}

1 Introduction

Data-to-text generators produce a target natural language text from a source data representation. Recent neural generation approaches (Mei et al., 2016; Lebret et al., 2016; Wiseman et al., 2017; Gardent et al., 2017b; Ferreira et al., 2017; Konstas et al., 2017) build on encoder-decoder architectures proposed for machine translation (Sutskever et al., 2014; Bahdanau et al., 2015).

The source data, differently from the machine translation task, is a structured representation of the content to be conveyed. Generally, it describes attributes and events about entities and relations among them. In this work we focus on two generation scenarios where the source data is graph structured. One is the generation of multi-sentence descriptions of Knowledge Base (KB) entities from RDF graphs (Perez-Beltrachini et al., 2016; Gardent et al., 2017a,b), namely the WebNLG task.\textsuperscript{2} The number of KB relations modelled in this scenario is potentially large and generation involves solving various subtasks (e.g. lexicalisation and aggregation). Figure (1a) shows and example of source RDF graph and target natural language description. The other is the linguistic realisation of the meaning expressed by a source dependency graph (Belz et al., 2011), namely the SR11Deep generation task. In this task, the semantic relations are linguistically motivated and their number is smaller. Figure (1b) illustrates a source dependency graph and the corresponding target text.

Most previous work casts the graph structured data to text generation task as a sequence-to-sequence problem (Gardent et al., 2017b; Ferreira et al., 2017; Konstas et al., 2017). They rely on recurrent data encoders with memory and gating mechanisms (LSTM; (Hochreiter and Schmidhuber, 1997)). Models based on these sequential encoders have shown good results although they do not directly exploit the input structure but rather rely on a separate linearisation step. In this work, we compare with a model that explicitly encodes structure and is trained end-to-end. Concretely, we use a Graph Convolutional Network (GCN; (Kipf and Welling, 2016; Marcheggiani and Titov, 2017)) as our encoder.

GCNs are a flexible architecture that allows explicit encoding of graph data into neural networks. Given their simplicity and expressiveness they have been used to encode dependency syntax and predicate-argument structures in neural machine translation (Bastings et al., 2017; Marcheggiani et al., 2018). In contrast to previous work, we do not exploit the sequential information of the input (i.e., with an LSTM), but we solely rely on a GCN for encoding the source graph structure.\textsuperscript{3}

The main contribution of this work is showing that explicitly encoding structured data with

\textsuperscript{1}Code and data available at github.com/diegma/graph-2-text.
\textsuperscript{2}Resource Description Framework https://www.w3.org/RDF/
\textsuperscript{3}Concurrently with this work, Beck et al. (2018) also encoded input structures without relying on sequential encoders.
GCNs is more effective than encoding a linearized version of the structure with LSTMs. We evaluate the GCN-based generator on two graph-to-sequence tasks, with different level of source content specification. In both cases, the results we obtain show that GCNs encoders outperforms standard LSTM encoders.

2 Graph Convolutional-based Generator

Formally, we address the task of text generation from graph-structured data considering as input a directed labeled graph \( X = (V, E) \) where \( V \) is a set of nodes and \( E \) is a set of edges between nodes in \( V \). The specific semantics of \( X \) depends on the task at hand. The output \( Y \) is a natural language text verbalising the content expressed by \( X \). Our generation model follows the standard attention-based encoder-decoder architecture (Bahdanau et al., 2015; Luong et al., 2015) and predicts \( Y \) conditioned on \( X \) as \( P(Y|X) = \prod_{t=1}^{|Y|} P(y_t|y_{1:t-1}, X) \).

Graph Convolutional Encoder In order to explicitly encode structural information we adopt graph convolutional networks (GCNs). GCNs are a variant of graph neural networks (Scarselli et al., 2009) that has been recently proposed by Kipf and Welling (2016). The goal of GCNs is to calculate the representation of each node in a graph considering the graph structure. In this paper we adopt the parametrization proposed by Marcheggiani and Titov (2017) where edge labels and directions are explicitly modeled. Formally, given a directed graph \( X = (V, E) \), where \( V \) is a set of nodes, and \( E \) is a set of edges. We represent each node \( v \in V \) with a feature vector \( x_v \in \mathbb{R}^d \). The GCN calculates the representation of each node \( h_v' \) in a graph using the following update rule:

\[
h_v' = \rho \left( \sum_{u \in \mathcal{N}(v)} g_{u,v}(W_{\text{dir}(u,v)} h_u + b_{\text{lab}(u,v)}) \right),
\]

where \( \mathcal{N}(v) \) is the set of neighbours of \( v \), \( W_{\text{dir}(u,v)} \in \mathbb{R}^{d \times d} \) is a direction-specific parameter matrix. As Marcheggiani and Titov (2017); Bastings et al. (2017) we assume there are three possible directions (\( \text{dir}(u,v) \in \{\text{in}, \text{out}, \text{loop}\} \)):

- self-loop edges ensure that the initial representation of node \( h_v \) affects the new representation \( h_v' \).
- The vector \( b_{\text{lab}(u,v)} \in \mathbb{R}^d \) is an embedding of the label of the edge \( (u,v) \).
- \( \rho \) is a non-linearity (ReLU), \( g_{u,v} \) are learned scalar gates which weight the importance of each edge. Although the main aim of gates is to down weight erroneous edges in predicted graphs, they also add flexibility when several GCN layers are stacked.

Skip Connections Between GCN layers we add skip connections. Skip connections let the gradient flows more efficiently through stacked hidden layers thus making possible the creation of deeper GCN encoders. We use two kinds of skip connections: residual connections (He et al., 2016) and dense connections (Huang et al., 2017). Residual connections consist in summing input and output representations of a GCN layer \( h_v^c = h_v' + h_v \). Whilst, dense connections consist in the concatenation of the input and output representations \( h_v^d = [h_v', h_v] \). In this way, each GCN layer is directly fed with the output of every layer before itself.

Decoder The decoder uses an LSTM and a soft attention mechanism (Luong et al., 2015) over

\[\text{(a) Above the Veil is an Australian novel and the sequel to Aenir and Castle. It was followed by Into the Battle and The Violet Keystone.}\]

\[\text{(b) Giant agreed last month to purchase the carrier.}\]

Figure 1: Source RDF graph - target description (a). Source dependency graph - target sentence (b).
3 Generation Tasks

In this section, we describe the instantiation of the input graph \( X \) for the generation tasks we address.

3.1 WebNLG Task

The WebNLG task (Gardent et al., 2017a,b) aims at the generation of entity descriptions from a set of RDF triples related to an entity of a given category (Perez-Beltrachini et al., 2016). RDF triples are of the form (subject relation object), e.g., (Aenir precededBy Castle), and form a graph in which edges are labelled with relations and vertices with subject and object entities. For instance, Figure (1a) shows a set of RDF triples related to the book Above the Veil and its verbalisation. The generation task involves several micro-planning decisions such as lexicalisation (followedBy is verbalised as sequel to), aggregation (sequel to Aenir and Castle), referring expressions (subject of the second sentence verbalised as pronoun) and segmentation (content organised in two sentences).

Reification We formulate this task as the generation of a target description \( Y \) from a source graph \( X = (\mathcal{V}, \mathcal{E}) \) where \( X \) is build from a set of RDF triples as follows. We reify the relations (Baader, 2003) from the RDF set of triples. That is, we see the relation as a concept in the KB and introduce a new relation node for each relation of each RDF triple. The new relation node is connected to the subject and object entities by two new binary relations A0 and A1 respectively. For instance, (precededBy A0 Aenir) and (precededBy A1 Castle). Thus, \( \mathcal{E} \) is a set of entities including reified relations and \( \mathcal{V} \) a set of labelled edges with labels \{A0, A1\}. The reification of relations is useful in two ways. The encoder is able to produce a hidden state for each relation in the input; and it permits to model an arbitrary number of KB relations efficiently.

3.2 SR11Deep Task

The surface realisation shared task (Belz et al., 2011) proposed two generation tasks, namely shallow and deep realisation. Here we focus on the deep task where the input is a semantic dependency graph that represents a target sentence using predicate-argument structures (NomBank; (Meyers et al., 2004), PropBank; (Palmer et al., 2005)). This task covers a more complex semantic representation of language meaning; on the other hand, the representation is closer to surface form. Nodes in the graph are lemmas of the target sentence. Only complementizers that, commas, and to infinitive nodes are removed. Edges are labelled with NomBank and PropBank labels. Each node is also associated with morphological (e.g. num=sg) and punctuation features (e.g. bracket=). The source graph \( X = (\mathcal{V}, \mathcal{E}) \) is a semantic dependency graph. We extend this representation to model morphological information, i.e. each node in \( \mathcal{V} \) is of the form (lemma, features). For this task we modify the encoder, Section 2, to represent each input node as \( h_v = [h_v; h_x] \), where each input node is the concatenation of the lemma and the sum of feature vectors.

4 Experiments

We tested our models on the WebNLG and SR11Deep datasets. The WebNLG dataset contains 18102 training and 871 development data-text pairs. The test dataset is split in two sets, test Seen (971 pairs) and a test set with new unseen categories for KB entities. As here we are interested only in the modelling aspects of the structured input data we focus on our evaluation only on the test partition with seen categories. The dataset covers 373 distinct relations from DBpedia. The SR11Deep dataset contains 39279, 1034 and 2398 examples in the training, development and test partitions, respectively. It covers 117 distinct dependency relations.

Sequential Encoders For both WebNLG and SR11Deep tasks we used a standard sequence-to-sequence model (Bahdanau et al., 2015; Luong et al., 2015) with an LSTM encoder as baseline. Both take as input a linearised version of the representation induced by the GCN encoder to generate one word \( y_t \) at the time. The prediction of word \( y_{t+1} \) is conditioned on the previously predicted words \( y_{1:t} \) encoded in the vector \( w_t \) and a context vector \( c_t \) dynamically created attending to the graph representation induced by the GCN encoder as \( P(y_{t+1}|y_{1:t}, X) = \text{softmax}(g(w_t, c_t)) \), where \( g(\cdot) \) is a neural network with one hidden layer. The model is trained to optimize negative log likelihood: \( \mathcal{L}_{NLL} = - \sum_{t=1}^{T} \log P(y_t|y_{1:t-1}, X) \)

5 There are also some cases where syntactic labels appear in the graphs, this is due to the creation process (see (Belz et al., 2011)) and done to connect graphs when there were disconnected parts.

6 In both datasets we exclude pairs with >50 target words.
the source graph. For the WebNLG baseline, we use the linearisation scripts provided by (Gardent et al., 2017b). For the SR11Deep baseline we follow a similar linearisation procedure as proposed for AMR graphs (Konstas et al., 2017). We built a linearisation based on a depth first traversal of the input graph. Siblings are traversed in random order (they are anyway shuffled in the given dataset). We repeat a child node when a node is revisited by a cycle or has more than one parent. The baseline model for the WebNLG task uses one layer bidirectional LSTM encoder and one layer LSTM decoder with embeddings and hidden units set to 256 dimensions. For the SR11Deep task we used the same architecture with 500-dimensional hidden states and embeddings. All hyperparameters tuned on the development set.

**GCN Encoders** The GCN models consist of a GCN encoder and LSTM decoder. For the WebNLG task, all encoder and decoder embeddings and hidden units use 256 dimensions. We obtained the best results with an encoder with four GCN layers with residual connections. For the SR11Deep task, we set the encoder and decoder to use 500-dimensional embeddings and hidden units of size 500. In this task, we obtained the best development performance by stacking seven GCN layers with dense connections.

We use delexicalisation for the WebNLG dataset and apply the procedure provided for the baseline in (Gardent et al., 2017b). For the SR11Deep dataset, we performed entity anonymisation. First, we compacted nodes in the tree corresponding to a single named entity (see (Belz et al., 2011) for details). Next, we used a name entity recogniser (Stanford CoreNLP; (Manning et al., 2014)) to tag entities in the input with type information (e.g. person, location, date). Two entities of the same type in a given input will be given a numerical suffix, e.g. PER_0 and PER_1.

A GCN-based Generator For the WebNLG task, we extended the GCN-based model to use pre-trained word Embeddings (GloVe (Pennington et al., 2014)) and Copy mechanism (See et al., 2017), we name this variant GCN_{EC}. To this end, we did not use delexicalisation but rather represent multi-word subject (object) entities with each word as a separate node connected with special Named Entity (NE) labelled edges. For instance, the book entity *Into Battle* is represented as *(Into

<table>
<thead>
<tr>
<th>Encoder</th>
<th>BLEU</th>
<th>METEOR</th>
<th>TER</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>.526±.010</td>
<td>.38±.00</td>
<td>.43±.01</td>
</tr>
<tr>
<td>GCN</td>
<td>.535±.004</td>
<td>.39±.00</td>
<td>.44±.02</td>
</tr>
<tr>
<td>ADAPT</td>
<td>.606</td>
<td>.44</td>
<td>.37</td>
</tr>
<tr>
<td>GCN_{EC}</td>
<td>.559±.017</td>
<td>.39±.01</td>
<td>.41±.01</td>
</tr>
<tr>
<td>MELBOURNE</td>
<td>.545</td>
<td>.41</td>
<td>.40</td>
</tr>
<tr>
<td>PKUWRITER</td>
<td>.512</td>
<td>.37</td>
<td>.45</td>
</tr>
</tbody>
</table>

**Table 1:** Test results WebNLG task.

<table>
<thead>
<tr>
<th>Encoder</th>
<th>BLEU</th>
<th>METEOR</th>
<th>TER</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>.377±.007</td>
<td>.65±.00</td>
<td>.44±.01</td>
</tr>
<tr>
<td>GCN</td>
<td>.647±.005</td>
<td>.77±.00</td>
<td>.24±.01</td>
</tr>
<tr>
<td>GCN+feat</td>
<td>.666±.027</td>
<td>.76±.01</td>
<td>.25±.01</td>
</tr>
</tbody>
</table>

**Table 2:** Test results SR11Deep task.

**Evaluation metrics** As previous works in these tasks, we evaluated our models using BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014) and TER (Snover et al., 2006) automatic metrics. During preliminary experiments we noticed considerable variance from different model initialisations; we thus run 3 experiments for each model and report average and standard deviation for each metric.

**5 Results**

**WebNLG task** In Table 1 we report results on the WebNLG test data. In this setting, the model with GCN encoder outperforms a strong baseline that employs the LSTM encoder, with .009 BLEU points. The GCN model is also more stable than the baseline with a standard deviation of .004 vs .010. We also compared the GCN_{EC} model with the neural models submitted to the WebNLG shared task. The GCN_{EC} model outperforms PKUWRITER that uses an ensemble of 7 models and a further reinforcement learning step by .047 BLEU points; and MELBOURNE by .014 BLEU points. GCN_{EC} is behind ADAPT which relies on sub-word encoding.

**SR11Deep task** In this more challenging task, the GCN encoder is able to better capture the structure of the input graph than the LSTM encoder, resulting in .647 BLEU for the GCN vs .377 BLEU of the LSTM encoder as reported in Table 2. When we add linguistic features to the GCN encoding we get .666 BLEU points. We also
5.1 Qualitative Analysis of Generated Text

We manually inspected the outputs of the LSTM and GCN models. Table 3 shows examples of source graphs and generated texts (we included more examples in Section A). Both models suffer from repeated and missing source content (i.e., source units are not verbalised in the output text (under-generation)). However, these phenomena are less evident with GCN-based models. We also observed that the LSTM output sometimes presents hallucination (over-generation) cases. Our intuition is that the strong relational inductive bias of GCNs (Battaglia et al., 2018) helps the GCN encoder to produce a more informative representation of the input; while the LSTM-based encoder has to learn to produce useful representations by going through multiple different sequences over the source data.

5.2 Ablation Study

In Table 4 (BLEU) we report an ablation study on the impact of the number of layers and the type of skip connections on the WebNLG dataset. The first thing we notice is the importance of skip connections between GCN layers. Residual and dense connections lead to similar results. Dense connections (Table 4 (SIZE)) produce models bigger, but slightly less accurate, than residual connections. The best GCN model has slightly more parameters than the baseline model (4.9M vs. 4.3M).

6 Conclusion

We compared LSTM sequential encoders with a structured data encoder based on GCNs on the task of structured data to text generation. On two different tasks, WebNLG and SR11Deep, we show that explicitly encoding structural information with GCNs is beneficial with respect to sequential encoding. In future work, we plan to apply the approach to other input graph representations like Abstract Meaning Representations (AMR; Banerescu et al., 2013) and scoped semantic representations (Van Noord et al., 2018).

Acknowledgments

We want to thank Ivan Titov and Mirella Lapata for their help and suggestions. We also gratefully acknowledge the financial support of the European Research Council (award number 681760) and the Dutch National Science Foundation (NWO VIDI 639.022.518). We thank NVIDIA for donating the GPU used for this research.
References


A Supplemental Material

A.1 Training details
We implemented all our models using OpenNMT-py (Klein et al., 2017). For all experiments we used a batch size of 64 and Adam (Kingma and Ba, 2015) as the optimizer with an initial learning rate of 0.001. For GCN models and baselines we used a one-layer LSTM decoder, we used dropout (Srivastava et al., 2014) in both encoder and decoder with a rate of 0.3. We adopt early stopping on the development set using BLEU scores and we trained for a maximum of 30 epochs.

A.2 More example outputs
Table 5 shows additional examples of generated texts for source WebNLG and SR11Deep graphs.
Table 5: Examples of system output.