Modeling Global and Local Node Contexts for Text Generation from Knowledge Graphs

Leonardo F. R. Ribeiro[†], **Yue Zhang**[‡], **Claire Gardent**[§] **and Iryna Gurevych**[†]

[†]Research Training Group AIPHES and UKP Lab, Technische Universität Darmstadt
[‡]School of Engineering, Westlake University, [§]CNRS/LORIA, Nancy, France
ribeiro@aiphes.tu-darmstadt.de, yue.zhang@wias.org.cn
claire.gardent@loria.fr, gurevych@ukp.informatik.tu-darmstadt.de

Abstract

Recent graph-to-text models generate text from graph-based data using either global or local aggregation to learn node representations. *Global node encoding* allows explicit communication between two distant nodes, thereby neglecting graph topology as all nodes are directly connected. In contrast, *local node encoding* considers the relations between neighbor nodes capturing the graph structure, but it can fail to capture long-range relations. In this work, we gather both encoding strategies, proposing novel neural models that encode an input graph combining both global and local node contexts, in order to learn better contextualized node embeddings. In our experiments, we demonstrate that our approaches lead to significant improvements on two graph-to-text datasets achieving BLEU scores of 18.01 on the AGENDA dataset, and 63.69 on the WebNLG dataset for seen categories, outperforming state-of-the-art models by 3.7 and 3.1 points, respectively.[1]

1 Introduction

Graph-to-text generation refers to the task of generating natural language text from input graph structures, which can be semantic representations (Konstas et al., 2017) or knowledge graphs (KGs) (Gardent et al., 2017; Koncel-Kedziorski et al., 2019). Whereas most recent work (Song et al., 2018; Ribeiro et al., 2019; Guo et al., 2019) focuses on generating sentences, a more challenging and interesting scenario emerges when the goal is to generate multisentence texts. In this context, in addition to sentence generation, document planning needs to be handled: The input needs to be mapped into several sentences; sentences need to be ordered and connected using appropriate discourse markers; and inter-sentential anaphora and ellipsis may need to be generated to avoid repetition. In this paper, we focus on generating texts rather than sentences where the output are short texts (Gardent et al., 2017) or paragraphs (Koncel-Kedziorski et al., 2019).

A key issue in neural graph-to-text generation is how to encode the input graphs. The basic idea is to incrementally compute node representations by aggregating structural context information. To this end, two main approaches have been proposed: (i) models based on *local node aggregation*, usually built upon Graph Neural Networks (GNNs) (Kipf and Welling, 2017; Hamilton et al., 2017) and (ii) models that leverage *global node aggregation*. Systems that adopt global encoding strategies are typically based on Transformers (Vaswani et al. 2017), using self-attention to compute a node representation based on all nodes in the graph. This approach enjoys the advantage of a large node context range, but neglects the graph topology by effectively treating every node as being connected to all the others in the graph. In contrast, models based on local aggregation learn the representation of each node based on its adjacent nodes as defined in the input graph. This approach effectively exploits the graph topology, and the graph structure has a strong impact on the node representation (Xu et al., 2018). However, encoding relations between distant nodes can be challenging by requiring more graph encoding layers, which can also propagate noise (Li et al., 2018).

For example, Figure 1a presents a KG, for which a corresponding text is shown in Figure 1b. Note that there is a mismatch between how entities are connected in the graph and how their natural language descriptions are related in the text. Some entities syntactically related in the text are not connected in the graph. For instance, in the

[1]Code is available at https://github.com/UKPLab/kg2text.

Transactions of the Association for Computational Linguistics, vol. 8, pp. 589–604, 2020. https://doi.org/10.1162/tacl_a_00332
Action Editor: Alessandro Moschitti. Submission batch: 2/2019; Revision batch: 5/2020; Published 9/2020.

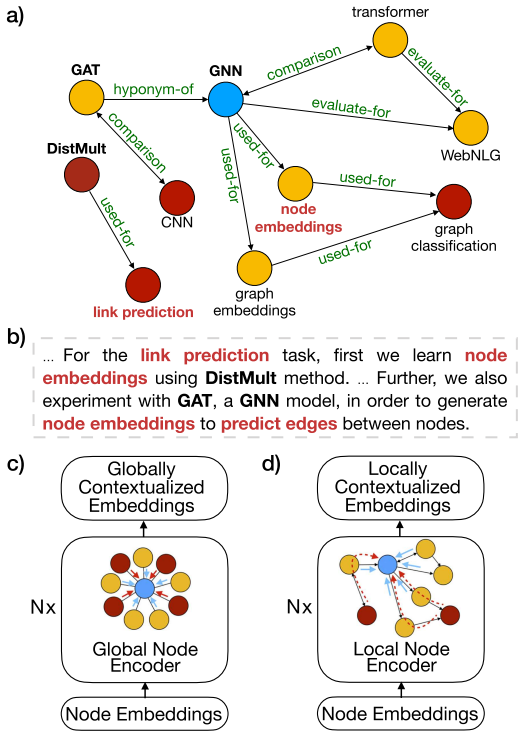


Figure 1: A graphical representation (a) of a scientific text (b). (c) A global encoder directly captures longer dependencies between any pair of nodes (blue and red arrows), but fails in capturing the graph structure. (d) A local encoder explicitly accesses information from the adjacent nodes (blue arrows) and implicitly captures distant information (dashed red arrows).

sentence ''For *the link prediction task*, first we learn *node embeddings* using *DistMult method*.'', although the entity mentions are dependent of the same verb, in the graph, the *node embeddings* node has no explicit connection with *link prediction* and *DistMult* nodes, which are in a different connected component. This example illustrates the importance of encoding distant information in the input graph. As shown in Figure 1c, a global encoder is able to learn a node representation for *node embeddings* which captures information from non-connected entities such as *DistMult*. By modeling distant connections between all nodes, we allow for these missing links to be captured, as KGs are known to be highly incomplete (Dong et al., 2014; Schlichtkrull et al., 2018).

In contrast, the local strategy refines the node representation with richer neighborhood information, as nodes that share the same neighborhood exhibit a strong *homophily*: Two similar entities are much more likely to be connected than at random. Consequently, the local context enriches the node representation with local information from KG triples. For example, in Figure 1a, *GAT* reaches *node embeddings* through the *GNN*. This transitive relation can be captured by a local encoder, as shown in Figure 1d. Capturing this form of relationship also can support text generation at the sentence level.

In this paper, we investigate novel graph-to-text architectures that combine both *global* and *local* node aggregations, gathering the benefits from both strategies. In particular, we propose a unified graph-to-text framework based on Graph Attention Networks (GATs) (Veličković et al., 2018). As part of this framework, we empirically compare two main architectures: a *cascaded architecture* that performs global node aggregation before performing local node aggregation, and a *parallel architecture* that performs global and local aggregations simultaneously. The cascaded architecture allows the local encoder to leverage global encoding features, and the parallel architecture allows more independent features to complement each other. To further consider fine-grained integration, we additionally consider *layer-wise integration* of the global and local encoders.

Extensive experiments show that our approaches consistently outperform recent models on two benchmarks for text generation from KGs. To the best of our knowledge, we are the first to consider integrating global and local context aggregation in graph-to-text generation, and the first to propose a unified GAT structure for combining global and local node contexts.

2 Related Work

Early efforts for graph-to-text generation used statistical methods (Flanigan et al., 2016; Pourdamghani et al., 2016; Song et al., 2017). Recently, several neural graph-to-text models have exhibited success by leveraging encoder mechanisms based on LSTMs, GNNs, and Transformers.

AMR-to-Text Generation. Various neural models have been proposed to generate sentences from Abstract Meaning Representation (AMR) graphs. Konstas et al. (2017) provide the first neural approach for this task, by linearizing the input graph as a sequence of nodes and edges. Song et al. (2018) propose the graph recurrent network to directly encode the AMR nodes, whereas Beck et al. (2018) develop a model based on gated GNNs.

However, both approaches only use local node aggregation strategies. Damonte and Cohen (2019) combine graph convolutional networks and LSTMs in order to learn complementary node contexts. However, differently from Transformers and GNNs, LSTMs generate node representations that are influenced by the node order. Ribeiro et al. (2019) develop a model based on different GNNs that learns node representations which simultaneously encode a top–down and a bottom–up views of the AMR graphs, whereas Guo et al. (2019) leverage dense connectivity in GNNs. Recently, Wang et al. (2020) propose a local graph encoder based on Transformers using separated attentions for incoming and outgoing neighbors. Recent methods (Zhu et al., 2019; Cai and Lam, 2020) also use Transformers, but learn globalized node representations, modeling graph paths in order to capture structural relations.

KG-to-Text Generation. In this work, we focus on generating text from KGs. In comparison to AMRs, which are rooted and connected graphs, KGs do not have a defined topology, which may vary widely among different datasets, making the generation process more demanding. KGs are sparse structures that potentially contain a large number of relations. Moreover, we are typically interested in generating multisentence texts from KGs, and this involves solving document planning issues (Konstas and Lapata, 2013).

Recent neural approaches for KG-to-text generation simply linearize the KG triples, thereby loosing graph structure information. For instance, Colin and Gardent (2018), Moryossef et al. (2019), and Adapt (Gardent et al., 2017) utilize LSTM/GRU to encode WebNLG graphs. Castro Ferreira et al. (2019) systematically compare pipeline and end-to-end models for text generation from WebNLG graphs. Trisedya et al. (2018) develop a graph encoder based on LSTMs that captures relationships within and between triples. Previous work has also studied how to explicitly encode the graph structure using GNNs or Transformers. Marcheggiani and Perez Beltrachini (2018) propose an encoder based on graph convolutional networks, that consider explicitly local node contexts, and show superior performance compared with LSTMs. Recently, Koncel-Kedziorski et al. (2019) proposed a Transformer-based approach that computes the node representations by attending over node neighborhoods following a self-attention strategy. In contrast, our models focus on distinct global and local message passing mechanisms, capturing complementary graph contexts.

Integrating Global Information. There has been recent work that attempts to integrate global context in order to learn better node representations in graph-to-text generation. To this end, existing methods use an artificial global node for message exchange with the other nodes. This strategy can be regarded as extending the graph structure but using similar message passing mechanisms. In particular, Koncel-Kedziorski et al. (2019) add a global node to the graph and use its representation to initialize the decoder. Recently, Guo et al. (2019) and Cai and Lam (2020) also utilized an artificial global node with direct edges to all other nodes to allow global message exchange for AMR-to-text generation. Similarly, Zhang et al. (2018) use a global node to a graph recurrent network model for sentence representation. Different from the above methods, we consider integrating global and local contexts at the *node* level, rather than the *graph* level, by investigating *model* alternatives rather than *graph structure* changes. In addition, we integrate GAT and Transformer architectures into a unified global-local model.

3 Graph-to-Text Model

This section first describes (i) the graph transformation adopted to create a relational graph from the input (Section 3.1), and (ii) the graph encoders of our framework based on GAT (Veličković et al., 2018), for dealing with both global (Section 3.3) and local (Section 3.4) node contexts. We adopt GAT because it is closely related to the Transformer architecture (Vaswani et al., 2017), which provides a convenient prototype for modeling global node context. Then, (iii) we proposed strategies to combined the global and local graph enoders (Section 3.5). Finally, (iv) we describe the decoding and training procedures (Section 3.6).

3.1 Graph Preparation

We represent a KG as a multi-relational graph[2] $\mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e, \mathcal{R})$ with entity nodes $e \in \mathcal{V}_e$ and labeled edges $(e_h, r, e_t) \in \mathcal{E}_e$, where $r \in \mathcal{R}$

[2]In this paper, multi-relational graphs refer to directed graphs with labeled edges.

denotes the relation existing from the entity e_h to e_t.[3]

Unlike other current approaches (Koncel-Kedziorski et al., 2019; Moryossef et al., 2019), we represent an entity as a set of nodes. For instance, the KG node "node embedding" in Figure 1 will be represented by two nodes, one for the token "node" and the other for the token "embedding". Formally, we transform each $\mathcal{G}_e$ into a new graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where each token of an entity $e \in \mathcal{V}_e$ becomes a node $v \in \mathcal{V}$. We convert each edge $(e_h, r, e_t) \in \mathcal{E}_e$ into a set of edges (with the same relation r) and connect every token of e_h to every token of e_t. That is, an edge (u, r, v) will belong to $\mathcal{E}$ if and only if there exists an edge $(e_h, r, e_t) \in \mathcal{E}_e$ such that $u \in e_h$ and $v \in e_t$, where e_h and e_t are seen as sets of tokens. We represent each node $v \in \mathcal{V}$ with an embedding $h_v^0 \in \mathbb{R}^{d_v}$, generated from its corresponding token.

The new graph $\mathcal{G}$ increases the representational power of the models because it allows learning node embeddings at a token level, instead of entity level. This is particularly important for text generation as it permits the model to be more flexible, capturing richer relationships between entity tokens. This also allows the model to learn relations and attention functions between source and target tokens. However, it has the side effect of removing the natural sequential order of multi-word entities. To preserve this information, we use position embeddings (Vaswani et al., 2017), that is, h_v^0 becomes the sum of the corresponding token embedding and the positional embedding for v.

3.2 Graph Neural Networks (GNN)

Multilayer GNNs work by iteratively learning a representation vector h_v of a node v based on both its context node neighbors and edge features, through an information propagation scheme. More formally, the l-th layer aggregates the representations of v's context nodes:

$$h_{\mathcal{N}(v)}^{(l)} = \text{AGGR}^{(l)}\left(\left\{\left(h_u^{(l-1)}, r_{uv}\right) : u \in \mathcal{N}(v)\right\}\right),$$

where $\text{AGGR}^{(l)}(.)$ is an aggregation function, shared by all nodes on the l-th layer. r_{uv} represents the relation between u and v. $\mathcal{N}(v)$ is a set of context nodes for v. In most GNNs, the context nodes are those adjacent to v. $h_{\mathcal{N}(v)}^{(l)}$ is the aggregated context representation of $\mathcal{N}(v)$ at layer l. $h_{\mathcal{N}(v)}^{(l)}$ is used to update the representation of v:

$$h_v^{(l)} = \text{COMBINE}^{(l)}\left(h_v^{(l-1)}, h_{\mathcal{N}(v)}^{(l)}\right).$$

After L iterations, a node's representation encodes the structural information within its L-hop neighborhood. The choices of $\text{AGGR}^{(l)}(.)$ and $\text{COMBINE}^{(l)}(.)$ differ by the specific GNN model. An example of $\text{AGGR}^{(l)}(.)$ is the sum of the representations of $\mathcal{N}(v)$. An example of $\text{COMBINE}^{(l)}(.)$ is a concatenation after the feature transformation.

3.3 Global Graph Encoder

A global graph encoder aggregates the *global context* for updating each node based on all nodes of the graph (see Figure 1c). We use the attention mechanism as the message passing scheme, extending the self-attention network structure of Transformer to a GAT structure. In particular, we compute a layer of the *global convolution* for a node $v \in \mathcal{V}$, which takes the input feature representations h_v as input, adopting $\text{AGGR}^{(l)}(.)$ as:

$$h_{\mathcal{N}(v)} = \sum\nolimits_{u \in \mathcal{V}} \alpha_{vu} W_g \, h_u, \tag{1}$$

where $W_g \in \mathbb{R}^{d_v \times d_z}$ is a model parameter. The attention weight α_{vu} is calculated as:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in \mathcal{V}} \exp(e_{vk})}, \tag{2}$$

where

$$e_{vu} = \left(\left(W_q h_v\right)^\top \left(W_k h_u\right)\right) / d_z \tag{3}$$

is the attention function which measures the *global importance* of node u's features to node v. $W_q, W_k \in \mathbb{R}^{d_v \times d_z}$ are model parameters and d_z is a scaling factor. To capture distinct relations between nodes, K independent global convolutions are calculated and concatenated:

$$\hat{h}_{\mathcal{N}(v)} = \Big\|_{k=1}^{K} h_{\mathcal{N}(v)}^{(k)}. \tag{4}$$

Finally, we define $\text{COMBINE}^{(l)}(.)$ using layer normalization (LayerNorm) and a fully connected

[3] $\mathcal{R}$ contains relations both in canonical direction (e.g., used-for) and in inverse direction (e.g., used-for-inv), so that the models consider the differences in the incoming and outgoing relations.

feed-forward network (FFN), in a similar way as the transformer architecture:

$$\hat{h}_v = \text{LayerNorm}(\hat{h}_{\mathcal{N}(v)} + h_v), \quad (5)$$

$$h_v^{global} = \text{FFN}(\hat{h}_v) + \hat{h}_{\mathcal{N}(v)} + h_v. \quad (6)$$

Note that the global encoder creates an artificial complete graph with $\mathcal{O}(n^2)$ edges and does not consider the edge relations. In particular, if the labeled edges were considered, the self-attention space complexity would increase to $\Theta(|\mathcal{R}|n^2)$.

3.4 Local Graph Encoder

The representation h_v^{global} captures macro relationships from v to all other nodes in the graph. However, this representation lacks both structural information regarding the local neighborhood of v and the graph topology. Also, it does not capture labeled edges (relations) between nodes (see Equations 1 and 3). In order to capture these crucial graph properties and impose a strong relational inductive bias, we build a graph encoder to aggregate the *local context* by utilizing a modified version of GAT augmented with relational weights. In particular, we compute a layer of the *local convolution* for a node $v \in \mathcal{V}$, adopting $\text{AGGR}^{(l)}(.)$ as:

$$h_{\mathcal{N}(v)} = \sum\nolimits_{u \in \mathcal{N}(v)} \alpha_{vu} W_r h_u, \quad (7)$$

where $W_r \in \mathbb{R}^{d_v \times d_z}$ encodes the relation $r \in \mathcal{R}$ between u and v. $\mathcal{N}(v)$ is a set of nodes adjacent to v and v itself. The attention coefficient α_{vu} is computed as:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in \mathcal{N}(v)} \exp(e_{vk})}, \quad (8)$$

where

$$e_{vu} = \sigma\left(a^\top [W_v h_v \parallel W_r h_u]\right) \quad (9)$$

is the attention function which calculates the *local importance* of adjacent nodes, considering the edge labels. σ is an activation function, $\parallel$ denotes concatenation and $W_v \in \mathbb{R}^{d_v \times d_z}$ and $a \in \mathbb{R}^{2d_z}$ are model parameters.

We use multihead attentions to learn local relations in different perspectives, as in Equation 4, generating $\hat{h}_{\mathcal{N}(v)}$. Finally, we define $\text{COMBINE}^{(l)}(.)$ as:

$$h_v^{local} = \text{RNN}(h_v, \hat{h}_{\mathcal{N}(v)}), \quad (10)$$

where we use as RNN a Gated Recurrent Unit (GRU) (Cho et al., 2014). GRU facilitates information propagation between local layers. This choice is motivated by recent work (Xu et al., 2018; Dehmamy et al., 2019) that theoretically demonstrates that sharing information between layers helps the structural signals propagate. In a similar direction, AMR-to-text generation models use LSTMs (Song et al., 2017) and dense connections (Guo et al., 2019) between GNN layers.

3.5 Combining Global and Local Encodings

Our goal is to implement a graph encoder capable of encoding global and local aspects of the input graph. We hypothesize that these two sources of information are complementary, and a combination of both enriches node representations for text generation. In order to test this hypothesis, we investigate different combined architectures.

Intuitively, there are two general methods for integrating two types of representation. The first is to concatenate vectors of global and local contexts, which we call a *parallel* representation. The second is to form a pipeline, where a global representation is first obtained, which is then used as a input for calculating refined representations based on the local node context. We call this approach a *cascaded* representation.

Parallel and cascaded integration can be performed at the model level, considering the global and local graph encoders as two representation learning units disregarding internal structures. However, because our model takes a multilayer architecture, where each layer makes a level of abstraction in representation, we can alternatively consider integration on the layer level, so that more interaction between global and local contexts may be captured. As a result, we present four architectures for integration, as shown in Figure 2. All models serve the same purpose, and their relative strengths should be evaluated empirically.

Parallel Graph Encoding (PGE). In this setup, we compose global and local graph encoders in a fully parallel structure (Figure 2a). Note that each graph encoder can have different numbers of layers and attention heads. The final node

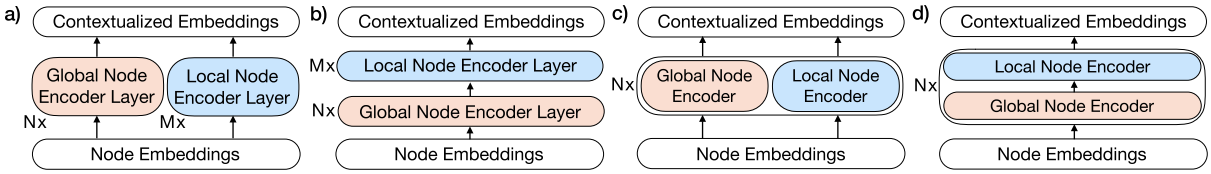


Figure 2: Overview of the proposed encoder architectures. (a) Parallel Graph Encoder (PGE) with separated parallel global and local node encoders. (b) Cascaded Graph Encoder (CGE) with separated cascaded encoders. (c) PGE−LW: global and local node representations are concatenated layer-wise. (d) CGE−LW: Both node representations are cascaded layer-wise.

representation is the concatenation of the local and global node representations of the last layers of both graph encoders:

$$\begin{aligned} h_v^{global} &= \text{GE}(h_v^0, \{h_u^0 : u \in \mathcal{V}\}) \\ h_v^{local} &= \text{LE}(h_v^0, \{h_u^0 : u \in \mathcal{N}(v)\}) \\ h_v &= [\, h_v^{global} \parallel h_v^{local} \,], \end{aligned} \tag{11}$$

where GE and LE denote the global and local graph encoders, respectively. h_v^0 is the initial node embedding used in the first layer of both encoders.

Cascaded Graph Encoding (CGE). We cascade local and global graph encoders as shown in Figure 2b. We first compute a globally contextualized node embedding, and then refine it with the local node context. h_v^0 is the initial input for the global encoder and h_v^{global} is the initial input for the local encoder. In particular, the final node representation is calculated as follows:

$$\begin{aligned} h_v^{global} &= \text{GE}(h_v^0, \{h_u^0 : u \in \mathcal{V}\}) \\ h_v &= \text{LE}(h_v^{global}, \{h_u^{global} : u \in \mathcal{N}(v)\}). \end{aligned} \tag{12}$$

Layer-wise Parallel Graph Encoding. To allow fine-grained interaction between the two types of graph contextual information, we also combine the encoders in a layer-wise (LW) fashion. As shown in Figure 2c, for each graph layer, we use both global and local encoders in a parallel structure (PGE−LW). More precisely, each encoder layer is calculates as follows:

$$\begin{aligned} h_v^{global} &= \text{GE}_l(h_v^{l-1}, \{h_u^{l-1} : u \in \mathcal{V}\}) \\ h_v^{local} &= \text{LE}_l(h_v^{l-1}, \{h_u^{l-1} : u \in \mathcal{N}(v)\}) \\ h_v^l &= [\, h_v^{global} \parallel h_v^{local} \,], \end{aligned} \tag{13}$$

where GE_l and LE_l refer to the l-th layers of the global and local graph encoders, respectively.

Layer-wise Cascaded Graph Encoding. We also propose cascading the graph encoders layer-wise (CGE−LW, Figure 2d). In particular, we compute each encoder layer as follows:

$$\begin{aligned} h_v^{global} &= \text{GE}_l(h_v^{l-1}, \{h_u^{l-1} : u \in \mathcal{V}\}) \\ h_v^l &= \text{LE}_l(h_v^{global}, \{h_u^{global} : u \in \mathcal{N}(v)\}). \end{aligned} \tag{14}$$

3.6 Decoder and Training

Our decoder follows the core architecture of a Transformer decoder (Vaswani et al., 2017). Each time step t is updated by performing multihead attentions over the output of the encoder (node embeddings h_v) and over previously generated tokens (token embeddings). An additional challenge in our setup is to generate multisentence outputs. In order to encourage the model to generate longer texts, we implement a length penalty (Wu et al., 2016) to refine the pure max-probability beam search.

The model is trained to optimize the negative log-likelihood of each gold-standard output text. We use label smoothing regularization to prevent the model from predicting the tokens too confidently during training and generalizing poorly.

4 Data and Preprocessing

We attest the effectiveness of our models on two datasets: AGENDA (Koncel-Kedziorski et al., 2019) and WebNLG (Gardent et al., 2017). Table 1 shows the statistics for both datasets.

AGENDA. In this dataset, KGs are paired with scientific abstracts extracted from proceedings of 12 top AI conferences. Each instance consists of the paper title, a KG, and the paper abstract. Entities correspond to scientific terms that are often multiword expressions (co-referential entities are merged). We treat each token in the title as a node, creating a unique graph with title and KG

	#train	#dev	#test	#relations	avg #entities	avg #nodes	avg #edges	avg #CC	avg length
AGENDA	38,720	1,000	1,000	7	12.4	44.3	68.6	19.1	140.3
WebNLG	18,102	872	971	373	4.0	34.9	101.0	1.5	24.2

Table 1: Data statistics. Nodes, edges, and CC values are calculated after the graph transformation. The average values are calculated for all splits (training, dev, and test sets). CC refers to the number of connected components.

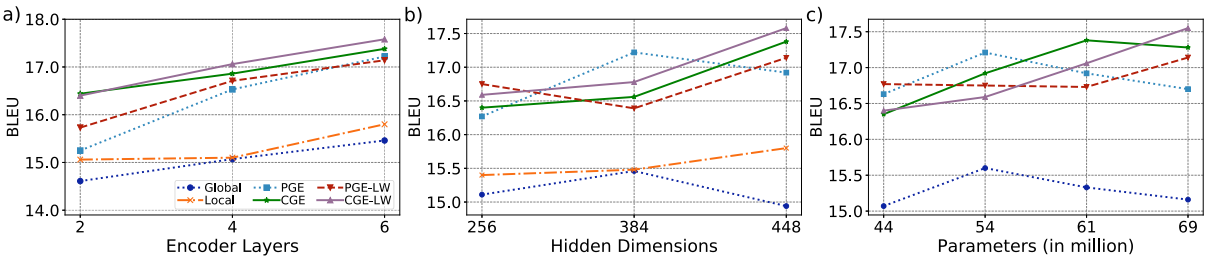


Figure 3: BLEU scores for the AGENDA dev set, with respect to (a) the encoder layers, (b) the encoder hidden dimensions, and (c) the number of parameters.

tokens as nodes. As shown in Table 1, the average output length is considerably large, as the target outputs are multisentence abstracts.

WebNLG. In this dataset, each instance contains a KG extracted from DBPedia. The target text consists of sentences that verbalize the graph. We evaluate the models on the test set with seen categories. Note that this dataset has a considerable number of edge relations (see Table 1). In order to avoid parameter explosion, we use regularization based on the basis function decomposition to define the model relation weights (Schlichtkrull et al., 2018). Also, as an alternative, we use the Levi Transformation to create nodes from relational edges between entities (Beck et al., 2018). That is, we create a new relation node for each edge relation between two nodes. The new relation node is connected to the subject and object token entities by two binary relations, respectively.

5 Experiments

We implemented all our models using PyTorch Geometric (PyG) (Fey and Lenssen, 2019) and OpenNMT-py (Klein et al., 2017). We use the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.98$. Our learning rate schedule follows Vaswani et al. (2017), with 8,000 and 16,000 warming-up steps for WebNLG and AGENDA, respectively. The vocabulary is shared between the node and target tokens. In order to mitigate the effects of random seeds, for the test sets, we report the averages over 4 training runs along with their standard deviation. We use byte pair encoding (Sennrich et al., 2016) to split entity words into smaller more frequent pieces. Therefore some nodes in the graph can be sub-words. We also obtain sub-words on the target side. Following previous works, we evaluate the results with BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), and CHRF++ (Popović, 2015) automatic metrics and also perform a human evaluation (Section 5.6). For layer-wise models, the number of encoder layers are chosen from $\{2, 4, 6\}$, and for `PGE` and `CGE`, the global and local layers are chosen from and $\{2, 4, 6\}$ and $\{1, 2, 3\}$, respectively. The hidden encoder dimensions are chosen from $\{256, 384, 448\}$ (see Figure 3). Hyperparameters are tuned on the development set of both datasets. We report the test results when the BLEU score on dev set is optimal.

5.1 Results on AGENDA

Table 2 shows the results, where we report the number of layers and attention heads utilized. We train models with only global or local encoders as baselines. Each model has the respective parameter size that gives the best results on the dev set. First, the local encoder, which requires fewer encoder layers and parameters, has a better performance compared with the global encoder. This shows that explicitly encoding the graph structure

Model	#L	#H	BLEU	METEOR	CHRF++	#P
Koncel-Kedziorski et al. (2019)	6	8	14.30 ± 1.01	18.80 ± 0.28	–	–
Global Encoder	6	8	15.44 ± 0.25	20.76 ± 0.194	43.95 ± 0.40	54.4
Local Encoder	3	8	16.03 ± 0.19	21.12 ± 0.32	44.70 ± 0.29	54.0
PGE	6, 3	8, 8	17.55 ± 0.154	22.02 ± 0.07	**46.41** ± 0.07	56.1
CGE	6, 3	8, 8	**17.82** ± 0.134	**22.23** ± 0.09	**46.47** ± 0.10	61.5
PGE-LW	6	8, 8	17.42 ± 0.25	21.78 ± 0.20	45.79 ± 0.32	69.0
CGE-LW	6	8, 8	**18.01** ± 0.14	**22.34** ± 0.07	**46.69** ± 0.17	69.8

Table 2: Results on the AGENDA test set. #L and #H are the numbers of layers and the attention heads in each layer, respectively. When more than one, the values are for the global and local encoders, respectively. #P stands for the number of parameters in millions (node embeddings included).

Model	BLEU	METEOR	CHRF++	#P
UPF-FORGe (Gardent et al., 2017)	40.88	40.00	–	–
Melbourne (Gardent et al., 2017)	54.52	41.00	70.72	–
Adapt (Gardent et al., 2017)	60.59	44.00	76.01	–
Marcheggiani and Perez Beltrachini (2018)	55.90	39.00	–	4.9
Trisedya et al. (2018)	58.60	40.60	–	–
Castro Ferreira et al. (2019)	57.20	41.00	–	–
CGE	62.30 ± 0.27	43.51 ± 0.18	75.49 ± 0.34	13.9
CGE (Levi Graph)	63.10 ± 0.13	44.11 ± 0.09	76.33 ± 0.10	12.8
CGE-LW	62.85 ± 0.07	43.75 ± 0.21	75.73 ± 0.31	11.2
CGE-LW (Levi Graph)	**63.69** ± 0.10	**44.47** ± 0.12	**76.66** ± 0.10	10.4

Table 3: Results on the WebNLG test set with seen categories.

is important to improve the node representations. Second, our approaches substantially outperform both baselines. CGE−LW outperforms Koncel-Kedziorski et al. (2019), a transformer model that focuses on the relations between adjacent nodes, by a large margin, achieving the new state-of-the-art BLEU score of 18.01, 25.9% higher. We also note that KGs are highly incomplete in this dataset, with an average number of connected components of 19.1 (see Table 1). For this reason, the global encoder plays an important role in our models as it enables learning node representations based on all connected components. The results indicate that combining the local node context, leveraging the graph topology, and the global node context, capturing macro-level node relations, leads to better performance. We find that, even though CGE has a small number of parameters compared to CGE−LW, it achieves comparable performance. PGE−LW has the worse performance among the proposed models. Finally, note that cascaded architectures are more effective according to different metrics.

5.2 Results on WebNLG

We compare the performance of our more effective models (CGE, CGE−LW) with six state-of-the-art results reported on this dataset. Three systems are the best competitors in the WebNLG challenge for seen categories: UPF-FORGe, Melbourne, and Adapt. UPF-FORGe follows a rule-based approach, whereas the others use neural encoder-decoder models with linearized triple sets as input.

Table 3 presents the results. CGE achieves a BLEU score of 62.30, 8.9% better than the best model of Castro Ferreira et al. (2019), who use an end-to-end architecture based on GRUs. CGE using Levi graphs outperforms Trisedya et al. (2018), an approach that encodes both intra-triple and inter-triple relationships, by 4.5 BLEU points. Interestingly, their intra-triple and inter-triple mechanisms are closely related with the local and global encodings. However, they rely on encoding entities based on sequences generated by traversal graph algorithms, whereas we explicitly

exploit the graph structure, throughout the local neighborhood aggregation.

CGE-LW with Levi graphs as inputs has the best performance, achieving 63.69 BLEU points, even thought it uses fewer parameters. Note that this approach allows the model to handle new relations, as they are treated as nodes. Moreover, the relations become part of the shared vocabulary, making this information directly usable during the decoding phase. We outperform an approach based on GNNs (Marcheggiani and Perez Beltrachini, 2018) by a large margin of 7.7 BLEU points, showing that our combined graph encoding strategies lead to better text generation. We also outperform Adapt, a strong competitor that utilizes subword encodings, by 3.1 BLEU points.

5.3 Development Experiments

We report several development experiments in Figure 3. Figure 3a shows the effect of the number of encoder layers in the four encoding methods.[4] In general, the performance increases when we gradually enlarge the number of layers, achieving the best performance with 6 encoder layers. Figure 3b shows the choices of hidden sizes for the encoders. The best performances for global and PGE are achieved with 384 dimensions, whereas the other models have the better performance with 448 dimensions. In Figure 3c, we evaluate the performance employing different number of parameters.[5] When the models are smaller, parallel encoders obtain better results than the cascaded ones. When the models are larger, cascaded models perform better. We speculate that for some models, the performance can be further improved with more parameters and layers. However, we do not attempt this owing to hardware limitations.

5.4 Ablation Study

In Table 4, we report an ablation study on the impact of each module used in CGE model on the dev set of AGENDA. We also report the number of parameters used in each configuration.

Global Graph Encoder. We start by an ablation on the global encoder. After removing the global attention coefficients, the performance of the model drops by 1.79 BLEU and 1.97 CHRF++ scores. Results also show that using FFN in the global COMBINE(.) function is important to the model but less effective than the global attention. However, when we remove FNN, the number of parameters drops considerably (around 18%) from 61.5 to 50.4 million. Finally, without the entire global encoder, the result drops substantially by 2.21 BLEU points. This indicates that enriching node embeddings with a global context allows learning more expressive graph representations.

Model	BLEU	CHRF++	#P
CGE	17.38	45.68	61.5
Global Encoder			
-Global Attention	15.59	43.71	59.0
-FFN	16.33	44.86	50.4
-Global Encoder	15.17	43.30	45.6
Local Encoder			
-Local Attention	16.92	45.97	61.5
-Weight Relations	16.88	45.61	53.6
-GRU	16.38	44.71	60.2
-Local Encoder	14.68	42.98	51.8
-Shared Vocab.	16.92	46.16	81.8
Decoder			
–Length Penalty	16.68	44.68	61.5

Table 4: Ablation study for modules used in the encoder and decoder of the CGE model.

Local Graph Encoder. We first remove the local graph attention and the BLEU score drops to 16.92, showing that the neighborhood attention improves the performance. After removing the relation types, encoded as model weights, the performance drops by 0.5 BLEU points. However, the number of parameters is reduced by around 7.9 million. This indicates that we can have a more efficient model, in terms of the number of parameters, with a slight drop in performance. Removing the GRU used on the COMBINE(.) function decreases the performance considerably. The worse performance occurs if we remove the entire local encoder, with a BLEU score of 14.68, essentially making the encoder similar to the global baseline.

Finally, we find that vocabulary sharing improves the performance, and the length penalty is beneficial as we generate multisentence outputs.

[4]For CGE and PGE the values refer to the global layers and the number of local layers is fixed to 3.

[5]It was not possible to execute the local model with larger number of parameters because of memory limitations.

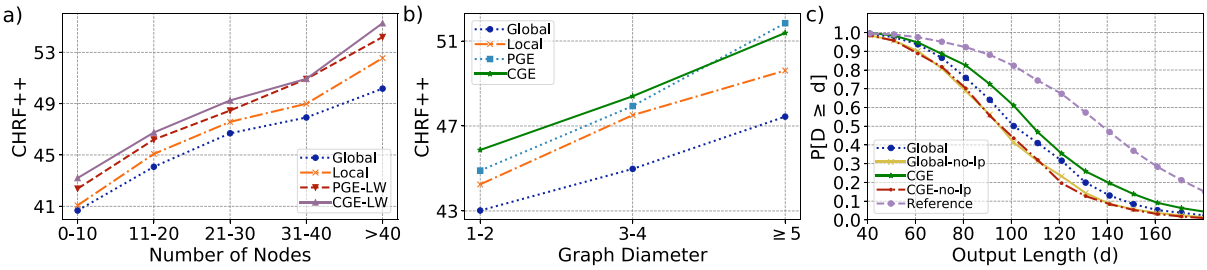


Figure 4: CHRF++ scores for the AGENDA test set, with respect to (a) the number of nodes, and (b) the graph diameter. (c) Distribution of length of the gold references and models' outputs for the AGENDA test set.

5.5 Impact of the Graph Structure and Output Length

The overall performance on both datasets suggests the strength of combining global and local node representations. However, we are also interested in estimating the models' performance concerning different data properties.

Graph Size. Figure 4a shows the effect of the graph size, measured in number of nodes, on the performance, measured using CHRF++ scores,[6] for the AGENDA. We evaluate global and local graph encoders, `PGE-LW` and `CGE-LW`. We find that the score increases as the graph size increases. Interestingly, the gap between the local and global encoders increases when the graph size increases. This suggests that, because larger graphs may have very different topologies, modeling the relations between nodes based on the graph structure is more beneficial than allowing direct communication between nodes, overlooking the graph structure. Also note that the the cascaded model (`CGE-LW`) is consistently better than the parallel model (`PGE-LW`) over all graph sizes.

Table 5 shows the effect of the graph size, measured in number of triples, on the performance for the WebNLG. Our model obtains better scores over all partitions. In contrast to AGENDA, the performance decreases as the graph size increases. This behavior highlights a crucial difference between AGENDA and AMR and WebNLG datasets, in which the models' general performance decreases as the graph size increases (Gardent et al., 2017; Cai and Lam, 2020). In WebNLG, the graph and sentence sizes are correlated, and longer sentences are more challenging to generate than the smaller ones. Differently, AGENDA contains similar text lengths[7] and when the input is a larger graph, the model has more information to be leveraged during the generation.

Graph Diameter. Figure 4b shows the impact of the graph diameter[8] on the performance for the AGENDA. Similarly to the graph size, the score increases as the diameter increases. As the global encoder is not aware of the graph structure, this module has the worst scores, even though it enables direct node communication over long distance. In contrast, the local encoder can propagate precise node information throughout the graph structure for k-hop distances, making the relative performance better. Table 5 shows the models' performances with respect to the graph diameter for WebNLG. Similarly to the graph size, the score decreases as the diameter increases.

Output Length. One interesting phenomenon to analyze is the length distribution (in number of words) of the generated outputs. We expect that our models generate texts with similar output lengths as the reference texts. As shown in Figure 4c, the references usually are bigger than the texts generated by all models for AGENDA. The texts generated by `CGE-no-pl`, a `CGE` model without length penalty, are consistently shorter than the texts from the global and `CGE` models. We increase the length of the texts when we use the length penalty (see Section 3.6). However, there is still a gap between the reference and the generated text lengths. We leave further investigation of this aspect for future work.

[6] CHRF++ score is used as it is a sentence-level metric.

[7] As shown on Figure 4c, 82% of the reference abstracts have more than 100 words.

[8] The diameter of a graph is defined as the length of the longest shortest path between two nodes. We convert the graphs into undirected graphs to calculate the diameters.

#T	#DP	Melbourne	Adapt	CGE-LW
1-2	396	78.74	83.10	84.35
3-4	386	66.84	72.02	72.27
5-7	189	61.85	69.28	70.25
#D	**#DP**	**Melbourne**	**Adapt**	**CGE-LW**
1	222	82.27	87.54	88.04
2	469	69.94	74.54	75.90
≥ 3	280	62.87	69.30	69.41
#S	**#DP**	**Melbourne**	**Adapt**	**CGE-LW**
1	388	77.19	81.66	82.03
2	306	67.29	73.29	73.78
3	151	66.30	72.46	73.21
4	66	66.73	71.26	75.16
≥ 5	60	61.93	67.57	69.20

Table 5: CHRF++ scores with respect to the number of triples (#T), graph diameters (#D), and number of sentences (#S) on the WebNLG test set. #DP refers to the number of datapoints.

Table 5 shows the models' performances with respect to the number of sentences for WebNLG. In general, increasing the number of sentences reduces the performance of all models. Note that when the number of sentences increases, the gap between CGE-LW and the baselines becomes larger. This suggests that our approach is able to better handle complex graph inputs in order to generate multisentence texts.

Effect of the Number of Nodes on the Output Length. Figure 5 shows the effect of the size of a graph, defined as the number of nodes, on the quality (measured in CHRF++ scores) and length of the generated text (in number of words) in the AGENDA dev set. We bin both the graph size and the output length in 4 classes. CGE consistently outperforms the global model, in some cases by a large margin. When handling smaller graphs (with ≤ 35 nodes), both models have difficulties generating good summaries. However, for these smaller graphs, our model achieves a score 12.2% better when generating texts with length ≤ 75. Interestingly, when generating longer texts (>140) from smaller graphs, our model outperforms the global encoder by an impressive 21.7%, indicating that our model is more effective in capturing semantic signals from graphs with scarce information. Our approach also performs better when the graph size is large (> 55) but the generation output is small (≤ 75), beating the global encoder by 9 points.

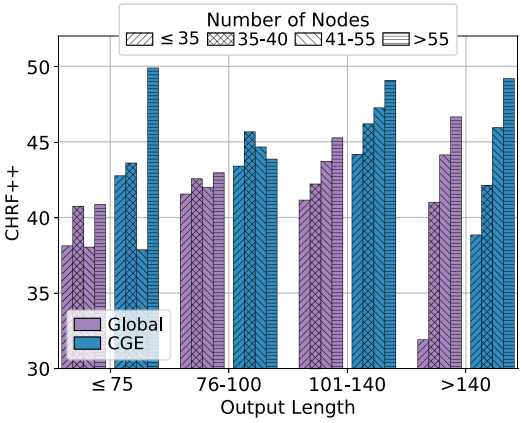


Figure 5: Relation between the number of nodes and the length of the generated text, in number of words.

5.6 Human Evaluation

To further assess the quality of the generated text, we conduct a human evaluation on the WebNLG dataset.[9] Following previous work (Gardent et al., 2017; Castro Ferreira et al., 2019), we assess two quality criteria: (i) *Fluency* (i.e., does the text flow in a natural, easy to read manner?) and (ii) *Adequacy* (i.e., does the text clearly express the data?). We divide the datapoints into seven different sets by the number of triples. For each set, we randomly select 20 texts generated by Adapt, CGE with Levi graphs, and their corresponding human reference (420 texts in total). Because the number of datapoints for each set is not balanced (see Table 5), this sampling strategy ensures that we have the same number of samples for the different triple sets. Moreover, having human references may serve as an indicator of the sanity of the human evaluation experiment. We recruited human workers from Amazon Mechanical Turk to rate the text outputs on a 1–5 Likert scale. For each text, we collect scores from 4 workers and average them. Table 6 shows the results. We first note a similar trend as in the automatic evaluation, with CGE outperforming Adapt on both fluency and adequacy. In sets with the number of triples smaller than 5, CGE was the highest rated system in fluency. Similarly to the automatic evaluation, both systems are better in generating text from

[9]Because AGENDA is scientific in nature, we choose to crowd-source human evaluations only for WebNLG.

#T	Adapt		CGE		Reference	
	F	A	F	A	F	A
All	3.96^{C}	4.44^{C}	4.12^{B}	4.54^{B}	4.24^{A}	4.63^{A}
1–2	3.94^{C}	4.59^{B}	4.18^{B}	4.72^{A}	4.30^{A}	4.69^{A}
3–4	3.79^{C}	4.45^{B}	3.96^{B}	4.50^{AB}	4.14^{A}	4.66^{A}
5–7	4.08^{B}	4.35^{B}	4.18^{B}	4.45^{B}	4.28^{A}	4.59^{A}
#D	**Adapt**		**CGE**		**Reference**	
	F	A	F	A	F	A
1–2	3.98^{C}	4.50^{B}	4.16^{B}	4.61^{A}	4.28^{A}	4.66^{A}
≥ 3	3.91^{C}	4.33^{B}	4.03^{B}	4.43^{B}	4.17^{A}	4.60^{A}

Table 6: Fluency (F) and Adequacy (A) obtained in the human evaluation. #T refers to the number of input triples and #D to graph diameters. The ranking was determined by pair-wise Mann-Whitney tests with $p < 0.05$, and the difference between systems that have a letter in common is not statistically significant.

AGENDA		
Model	**BLEU**	**CHRF++**
CGE-LW	18.17	46.80
-Shared Vocab	17.88	47.12
-Length Penalty	17.46	45.76
-Both	17.24	46.14
WebNLG		
Model	**BLEU**	**CHRF++**
CGE-LW	63.86	76.80
-Shared Vocab	63.07	76.17
-Length Penalty	63.28	76.51
-Both	62.60	75.80

Table 7: Effects of the vocabulary sharing and length penalty on the test sets of AGENDA and WebNLG.

graphs with smaller diameters. Note that bigger diameters pose difficulties to the models, which achieve their worst performance for diameters ≥ 3.

5.7 Additional Experiments

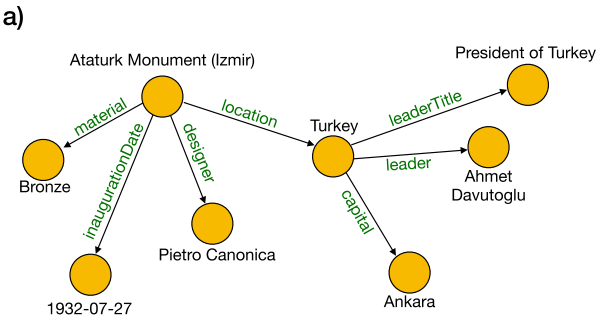
Impact of the Vocabulary Sharing and Length Penalty. During the ablation studies, we note that the vocabulary sharing and length penalty are beneficial for the performance. To better estimate their impact, we evaluate `CGE-LW` model with its variations without using vocabulary sharing, length penalty and without both mechanisms, on the test set of both datasets. Table 7 shows the results. We observe that sharing vocabulary is more important to WebNLG than AGENDA. This suggests that sharing vocabulary is beneficial when the training data is small, as in WebNLG. On the other hand, length penalty is more effective for AGENDA, as it has longer texts than WebNLG,[10] improving the BLEU score by 0.71 points.

How Far Does the Global Attention Look? Following previous work (Voita et al., 2019; Cai and Lam, 2020), we investigate the attention distribution of each graph encoder global layer of `CGE-LW` on the AGENDA dev set. In particular, for each node, we verify its global neighbor that receives the maximum attention weight and record the distance between them.[11] Figure 7 shows the averaged distances for each global layer. We observe that the global encoder mainly focuses on distant nodes, instead of the neighbors and closest nodes. This is very interesting and agrees with our intuition: Whereas the local encoder is concerned about the local neighborhood, the global encoder focuses on the information from long-distance nodes.

Case Study. Figure 6 shows examples of generated texts when the WebNLG graph is complex (7 triples). While `CGE` generates a factually correct text (it correctly verbalises all triples), the Adapt's output is repetitive. The example also illustrates how the text generated by `CGE` closely follows the graph structure whereby the first sentence verbalises the right-most subgraph, the second the left-most one and the linking node *Turkey* makes the transition (using hyperonymy and a definite description, i.e., *The country*). The text created by `CGE` is also more coherent than the reference. As noted above, the input graph includes two subgraphs linked by *Turkey*. In natural language, such a meaning representation corresponds to a topic shift with the first part of the text describing an entity from one subgraph, the second part an entity from the other subgraph, and the linking entity (*Turkey*) marking the topic shift. Typically,

[10] As shown in Table 1, AGENDA has texts 5.8 times longer than WebNLG on average.

[11] The distance between two nodes is defined as the number of edges in a shortest path connecting them.



b) Adapt: Ahmet Davutoglu is the president of Turkey. The capital city is Ankara, but it is in Izmir that the bronze Ataturk monument designed by Pietro Canonica and inaugurated on 27 July 1932 is located. The monument was designed in bronze by Pietro Canonica and inaugurated on 27 July 1932.

c) CGE: President Ahmet Davutoglu is the leader of Turkey where the capital city is Ankara. The country is the location of the bronze Ataturk monument designed by Pietro Canonica and inaugurated on 27 July 1932 in Izmir.

d) Reference: President Ahmet Davutoglu is the Turkish leader where the capital city is Ankara. The bronze Ataturk Monument which was designed by Pietro Canonica is located in Izmir and was inaugurated on 27 July 1932.

Figure 6: (a) A WebNLG input graph and the outputs for (b) Adapt and (c) CGE. The colored text indicates repetition.

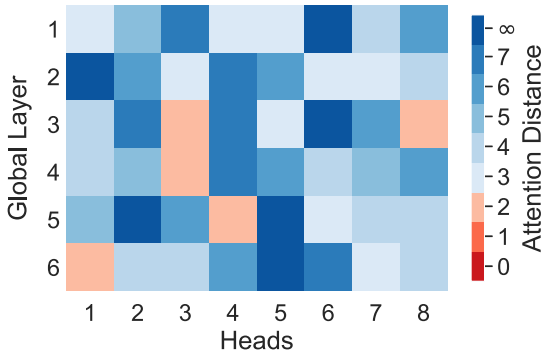


Figure 7: The average distance between nodes for the maximum attention for each head. ∞ indicates no path between two nodes, that is, they belong to distinct connected components.

in English, a topic shift is marked by a definite noun phrase in the subject position. Although this is precisely the discourse structure generated by CGE (*Turkey* is realized in the second sentence by the definite description *The country* in the subject position), the reference fails to mark the topic shift, resulting in a text with weaker discourse coherence.

6 Conclusion

In this work, we introduced a unified graph attention network structure for investigating graph-to-text models that combines global and local graph encoders in order to improve text generation. An extensive evaluation of our models demonstrated that the global and local contexts are empirically complementary, and a combination can achieve state-of-the-art results on two datasets. In addition, cascaded architectures give better results compared with parallel ones.

We point out some directions for future work. First, it is interesting to study different fusion strategies to assemble the global and local encodings. Second, a promising direction is incorporating pre-trained contextualized word embeddings in graphs. Third, as discussed in Section 5.5, it is worth studying ways to diminish the gap between the reference and the generated text lengths.

Acknowledgments

We would like to thank Pedro Savarese, Markus Zopf, Mohsen Mesgar, Prasetya Ajie Utama, Ji-Ung Lee, and Kevin Stowe for their feedback on this work, as well as the anonymous reviewers for detailed comments that improved this paper. This work has been supported by the German Research Foundation as part of the Research Training Group Adaptive Preparation of Information from Heterogeneous Sources (AIPHES) under grant No. GRK 1994/1.

References

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

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 552–562. Hong Kong, China. Association for Computational Linguistics.

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 the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Doha, Qatar. Association for Computational Linguistics.

Emilie Colin and Claire Gardent. 2018. Generating syntactic paraphrases. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 937–943, Brussels, Belgium. Association for Computational Linguistics.

Marco Damonte and Shay B. Cohen. 2019. Structural neural encoders for AMR-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3649–3658. Minneapolis, Minnesota. Association for Computational Linguistics.

Nima Dehmamy, Albert-Laszlo Barabasi, and Rose Yu. 2019. Understanding the representation power of graph neural networks in learning graph topology, In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 15387–15397. Curran Associates, Inc.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 376–380, Baltimore, Maryland, USA. Association for Computational Linguistics.

Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,z KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610.

Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

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

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

Zhijiang Guo, Yan Zhang, Zhiyang Teng, and Wei Lu. 2019. Densely connected graph convolutional networks for graph-to-sequence learning. *Transactions of the Association for Computational Linguistics*, 7:297–312.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs, In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1024–1034. Curran Associates, Inc.

Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th*

International Conference on Learning Representations, ICLR '17.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72. Vancouver, Canada. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text Generation from Knowledge Graphs with Graph Transformers. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2284–2293, Minneapolis, Minnesota. Association for Computational Linguistics.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada. Association for Computational Linguistics.

Ioannis Konstas and Mirella Lapata. 2013. Inducing document plans for concept-to-text generation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1503–1514, Seattle, Washington, USA. Association for Computational Linguistics.

Q. Li, Z. Han, and X.-M. Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI.

Diego Marcheggiani and Laura Perez Beltrachini. 2018. Deep graph convolutional encoders for structured data to text generation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 1–9, Tilburg University, The Netherlands. Association for Computational Linguistics.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2267–2277, Minneapolis, Minnesota. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318. Stroudsburg, PA, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram f-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. 2016. Generating English from abstract meaning representations. In *Proceedings of the 9th International Natural Language Generation conference*, pages 21–25, Edinburgh, UK. Association for Computational Linguistics.

Leonardo F. R. Ribeiro, Claire Gardent, and Iryna Gurevych. 2019. Enhancing AMR-to-text generation with dual graph representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3181–3192, Hong Kong, China. Association for Computational Linguistics.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, pages 593–607.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of*

the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

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

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

Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang, and Wei Wang. 2018. GTR-LSTM: A triple encoder for sentence generation from RDF data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1627–1637, Melbourne, Australia. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. Vancouver, Canada.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Tianming Wang, Xiaojun Wan, and Hanqi Jin. 2020. AMR-to-text generation with graph transformer. *Transactions of the Association for Computational Linguistics*, 8:19–33.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.

Yue Zhang, Qi Liu, and Linfeng Song. 2018. Sentence-state LSTM for text representation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 317–327, Melbourne, Australia. Association for Computational Linguistics.

Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. 2019. Modeling graph structure in transformer for better AMR-to-text generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5458–5467, Hong Kong, China. Association for Computational Linguistics.