Bridging the Structural Gap Between Encoding and Decoding for Data-To-Text Generation

Chao Zhao[†], **Marilyn Walker**[‡] **and Snigdha Chaturvedi**[†]
[†] Department of Computer Science, University of North Carolina at Chapel Hill
[‡] Natural Language and Dialog Systems Lab, University of California, Santa Cruz
{zhaochao, snigdha}@cs.unc.edu mawalker@ucsc.edu

Abstract

Generating sequential natural language descriptions from graph-structured data (e.g., knowledge graph) is challenging, partly because of the structural differences between the input graph and the output text. Hence, popular sequence-to-sequence models, which require serialized input, are not a natural fit for this task. Graph neural networks, on the other hand, can better encode the input graph but broaden the structural gap between the encoder and decoder, making faithful generation difficult. To narrow this gap, we propose DUALENC, a dual encoding model that can not only incorporate the graph structure, but can also cater to the linear structure of the output text. Empirical comparisons with strong single-encoder baselines demonstrate that dual encoding can significantly improve the quality of the generated text.

1 Introduction

Data-to-text generation aims to create natural language text to describe the input data (Reiter and Dale, 2000). Here we focus on structured text input in a particular form such as a tree or a graph. Figure 1 shows an example where the input data is a mini knowledge graph, and the output text is its corresponding natural language description. Generating text from such data is helpful for many NLP tasks, such as question answering and dialogue (He et al., 2017; Liu et al., 2018; Moon et al., 2019).

During generation, the structure of the data as well as the content inside the structure jointly determine the generated text. For example, the direction of the edge "capital" in Figure 1 determines that "London is the capital of U.K." is an accurate description, but not vice versa. Current generation methods are based on sequence-to-sequence (Seq2Seq) encoder-decoder architecture (Sutskever et al., 2014), which requires the input data to be serialized as a sequence, resulting in a loss of structural information.

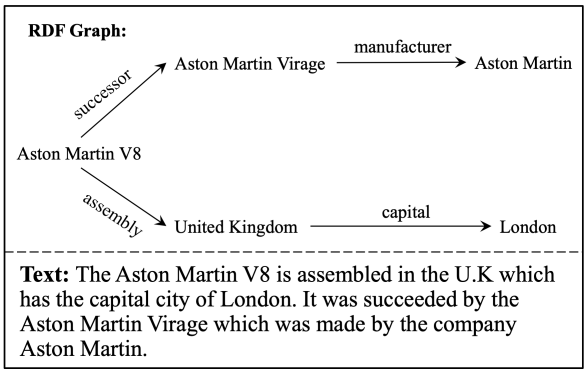

Figure 1: Illustration of the WebNLG challenge: the source data is an RDF graph and the target output is a text description of the graph.

Recent research has shown the utility of incorporating structural information during generation. By replacing the sequential encoder with a structure-aware graph encoder, such as a graph convolutional network (GCNs) (Kipf and Welling, 2017) or graph-state LSTMs (Song et al., 2018), the resulting graph-to-sequence (Graph2Seq) methods can encode the structural information of the input and thus outperform Seq2Seq models on certain tasks. However, these architectures broaden the structural gap between the encoder and decoder. That is, while the encoder receives the input data as a graph, the decoder has to create the output text as a linear chain structure.

This structural gap increases the difficulty of establishing alignments between source and target, which is believed to play a key role in text generation. For example, in machine translation, pre-reordering the source words into a word order that is close to that of the target sentence can yield significant improvements in translation quality (Bisazza and Federico, 2016). This suggests a need for an intermediate "planning" stage (Reiter

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2481–2491
July 5 - 10, 2020.

and Dale, 2000; Puduppully et al., 2019) to help with organizing the output.

In this work, we present a dual encoding model that is not only aware of the input graph structure but also incorporates a content planning stage. To encode the structural information in the input graph, we use a GCN based **graph encoder**. To narrow the ensuing structural gap, we use another GCN-based neural planner to create a sequential content plan of this graph, which is represented as a re-ordered sequence of its nodes. The plan is then encoded by an LSTM based **sequential encoder**. During generation, an LSTM based **decoder** simultaneously conditions on the two encoders, which helps it in capturing both the graph structure of the input data and the linear structure of the plan. We expect such a dual encoding (DUALENC) structure can integrate the advantages of both graph and sequential encoders while narrowing the structural gap present in single-encoder methods.

We evaluate the proposed planning and generation models on the WebNLG dataset (Colin et al., 2016; Gardent et al., 2017) – a widely used benchmark for data-to-text generation. Experimental results show that our neural planner achieves a 15% absolute improvement on accuracy compared to the previous best planning method. Furthermore, DUALENC significantly outperforms the previous start-of-the-art on the generation task. The human evaluation confirms that the texts generated by our model are preferred over strong baselines.

The contributions of this paper are three-fold:

- We propose a dual encoding method to narrow the structural gap between data encoder and text decoder for data-to-text generation;
- We propose a neural planner, which is more efficient and effective than previous methods;
- Experiments show that our method outperforms all baselines on a variety of measures.

2 Related Work

This work is inspired by two lines of research: Seq2Seq generation and Graph2Seq generation.

2.1 Seq2Seq Generation

Traditional data-to-text generation follows a planning and realization pipeline (Reiter and Dale, 2000; Stent et al., 2004). More recent methods use Seq2Seq architecture (Sutskever et al., 2014) to combine planning and realization into an end-to-end network and have achieved the state-of-the-art on a variety of generation tasks (Lebret et al., 2016; Trisedya et al., 2018; Juraska et al., 2018; Reed et al., 2018). Despite the fair fluency and grammatical correctness, the generated text suffers from several problems such as repetition, omission, and unfaithfulness, which are less likely to happen in traditional planning-and-realization frameworks.

Recent work has shown that neural models can also benefit from an explicit planning step to alleviate the above-mentioned problems. The input of these planners ranges from unstructured keyphrases (Hua and Wang, 2019) to structured tables (Puduppully et al., 2019) and graphs (Ferreira et al., 2019; Moryossef et al., 2019a). Our work also focuses on planning from graph data. Compared with previous methods, we show that our neural planning method is more feasible and accurate. More importantly, rather than serializing the planning and realization stages in a pipeline, our dual encoding method simultaneously captures information from the original data and the corresponding plan.

2.2 Graph2Seq Generation

Graph neural networks (GNN) (Scarselli et al., 2009) aim to learn a latent state representation for each node in a graph by aggregating local information from its neighbors and the connected edges. Previous work has explored different ways of aggregating this local information, such as in GCNs (Kipf and Welling, 2017), gated graph neural networks (GGNNs) (Li et al., 2016), and Graph attention networks (GANs) (Veličković et al., 2018)

Several works have applied GNNs instead of Seq2Seq models for text generation (Beck et al., 2018; Marcheggiani and Perez-Beltrachini, 2018; Guo et al., 2019; Li et al., 2019), and some of them outperform Seq2Seq models. However, Damonte and Cohen (2019) use both types of encoders and show that GCN can help LSTM capture reentrant structures and long-range dependencies, albeit on a different problem than ours. Our method also uses the two types of encoders but instead of using one to assist the other, it combines them simultaneously to capture their complementary effects.

3 Problem Statement

In this work we focus on text generation from RDF data.[1] The input for this task is a set of RDF triples, where each triple (s, p, o) contains a subject, a predicate, and an object. For example, ("U.K.", "cap-

[1] https://www.w3.org/TR/rdf-concepts/

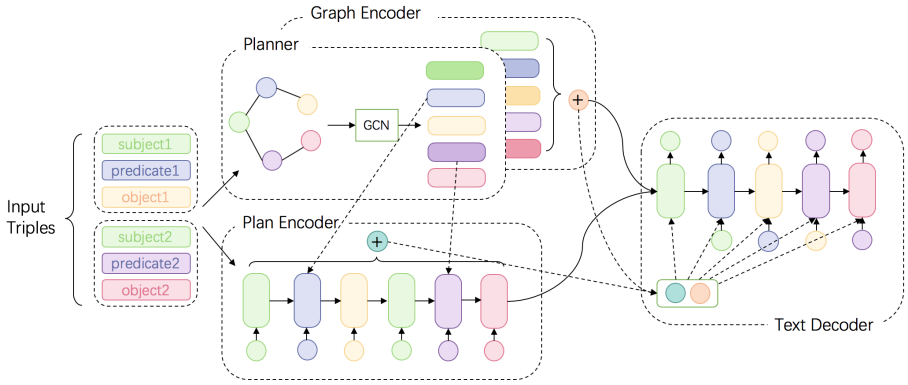

Figure 2: The architecture of the proposed DUALENC model. The input triples are converted as a graph and then fed to two GCN encoders for plan and text generation (Planner and Graph Encoder, top center). The plan is then encoded by an LSTM network (Plan Encoder, bottom center). Finally an LSTM decoder combines the hidden states from both the encoders to generate the text (Text Decoder, middle right).

ital", "London") is a RDF triple. The output is a natural language text with one or more sentences to describe the facts represented by this graph. Figure 1 shows an example of this task.

4 Dual Encoding Model

For a given input RDF graph, the aim of our method is not only to capture its structural information, but also to facilitate the information alignment between the input and output. The first goal can be achieved by employing a **GCN encoder**. To achieve the second goal, we first serialize and re-order the nodes of the graph as an intermediate plan using another GCN, and then feed the plan into an **LSTM encoder**. Finally, an **LSTM decoder** is used to generate the output by incorporating the context representations of both encoders. Notice that the graph and the plan are dual representations of the same input data. We encode them with two independent encoders, which can provide complementary information for decoding. The architecture of our dual encoding method is shown in Figure 2. We describe the two encoders and the decoder in the following three subsections.

4.1 Graph Representation and Encoding

To make it easier for GCNs to encode information from both entities and predicates, we reconstruct the input graph by regarding both entities and predicates as nodes, which is different from Figure 1.

Formally, for each RDF triple (s, p, o), we regard the s, p, and o as three kinds of nodes. s and o are identified by their entity mentions, and p is identified by a unique ID. That is, two entities from different triples that have the same mentions will be regarded as the same node. However, since we want to use predicates to distinguish between different triples, two predicates with the same mentions will be regarded as separate nodes.[2]

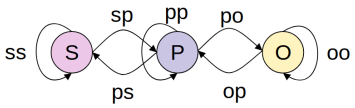

Figure 3: The graph obtained from an RDF triple.

We use the same edge structure as Beck et al. (2018). As Figure 3 shows, a triple contains four directed edges to connect its nodes: $s \rightarrow p$, $p \rightarrow s$, $o \rightarrow p$, and $p \rightarrow o$. These edges help in information exchange between arbitrary neighbor pairs. There is also a special self-loop edge $n \rightarrow n$ for each node n to enable information flow between adjacent iterations during feature aggregation.

After building the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from the RDF data, we use a relational GCN (R-GCN) (Schlichtkrull et al., 2018) to encode the graph and learn a state representation $\mathbf{h}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$ using the following iterative method:

$$\mathbf{h}_v^t = \rho\left(\sum_{r\in\mathcal{R}}\sum_{u\in\mathcal{N}_v^r}\frac{1}{c_{v,r}}\mathbf{W}_r\mathbf{h}_u^{(t-1)} + \mathbf{b}_r\right) \quad (1)$$

where $\mathbf{h}_v^0 = \mathbf{x}_v$ is the input embedding of the node v, and $\mathbf{h}_v^t$ is its hidden state at time-step t. We use the average embedding of the node mentions as $\mathbf{x}_v$. $\mathcal{R}$ is the set of all possible edge types, and $\mathcal{N}_v^r$ is the set of in-neighbors of node v with the edge

[2]For example, 'capital's in (U.K., capital, London) and (U.S., capital, Washington D.C.) are different nodes.

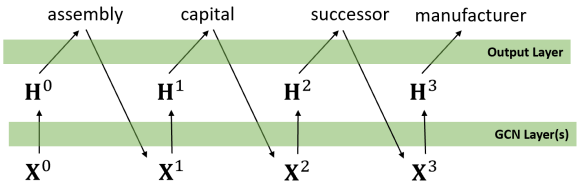

Figure 4: The sequential decision-making process of the planning stage.

type as r. $\mathbf{W}_r$ and $\mathbf{b}_r$ are parameters for each edge type, which allow transformations of message to become relational-specific. $c_{v,r} = 1/|\mathcal{N}_v^r|$ is a normalization term and $\rho()$ is an activation function.

4.2 Planning Creation and Encoding

In the planning stage, we determine the *content plan* or order of triples (identified by their predicates) for text realization. For example, the content plan for the text in Figure 1 is: "assembly → capital → successor → manufacturer ".[3]

Learning a plan can be naturally regarded as a sequential decision-making process. That is, given a set of triples, we first determine which triple to mention/visit first, and then select the second triple from the remaining triples that have not been visited so far. This process continues until all the triples have been visited. During each decision step, the selection of the next triple can be regarded as a classification task, where the output space is all the remaining unvisited triples.

Figure 4 shows how our model implements this process. We first utilize the GCN encoder described in Section 4.1 to get the state representation of each node. However, while obtaining a predicate's representation, we concatenate two extra bits to the input feature $\mathbf{X}^t$. One is to indicate whether or not the predicate has been visited, the other to indicate the last predicate that has been visited. After the encoding, we get the final hidden state $\mathbf{h}_{r_i} = \mathbf{h}_{r_i}^{(T)}$ for each predicate $r_i \in \mathcal{R}$ as its representation, and calculate its probability of being selected as

$$P(r_i) = \text{softmax}(\mathbf{h}_{r_i}^T \mathbf{W} \bar{\mathbf{h}}_{\mathcal{R}}) \quad (2)$$

where $\bar{\mathbf{h}}_{\mathcal{R}}$ is the average pooling of all the predicate embeddings. For obtaining a plan, we select the predicate with the highest probability, append it onto the plan sequence, and then repeat the above process until all the predicates have been visited.

[3]Here we only consider the order of triples. Future plans could explore ordering of subjects and/or objects.

After determining an order of input predicates, we complete the plan's triples by adding the corresponding subjects and objects. To better help the plan encoder (described below) capture the semantic roles of each entity and predicate, we add special tokens before **S**ubjects, **P**redicates, and **O**bjects as delimiters. For example, the plan of the example in Figure 1 will be:

<S> Aston Martin V8 <P> assembly <O> United Kingdom <S> United Kingdom <P> capital <O> London <S> Aston Martin V8 <P> successor <O> Aston Martin Virage <S> Aston Martin Virage <P> manufacturer <O> Aston Martin

Finally, we use an LSTM to encode the plan obtained above. We choose LSTM because it excels at capturing sequential information.

4.3 Decoding

During decoding, we adopt an LSTM-based decoder with an attention and copy mechanism. Since we have two representations of the input triple-set: the original graph and the serialized plan, we adopt two strategies for inputting context to the decoder.

The first strategy is to only use hidden states of the plan encoder as context. We refer to this strategy as PLANENC.

While the serialized plan may contain some structural information, it cannot preserve all the information of the original graph. We therefore propose a second strategy, DUALENC, to incorporate the information from both the graph and the plan. More concretely, when calculating the context state $\mathbf{m}_t$ of the LSTM decoder at time step t, we concatenate the previous hidden state $\mathbf{z}_{t-1}$ and the two context vectors $\mathbf{c}_t^1$ and $\mathbf{c}_t^2$, and then update the current hidden state, $\mathbf{z}_t$ as:

$$\mathbf{m}_t = \text{MLP}([\mathbf{z}_{t-1}; \mathbf{c}_t^1; \mathbf{c}_t^2]), \quad (3)$$

$$\mathbf{z}_t = \text{LSTM}\left(\mathbf{z}_{t-1}, [(\mathbf{y}_{t-1}; \mathbf{m}_t]\right), \quad (4)$$

where $\mathbf{c}_t^1$ and $\mathbf{c}_t^2$ are the attention-based weighted sum of the context memories from GCN and RNN encoders, respectively, and $\mathbf{y}_{t-1}$ is the embedding of the previously generated token. The initial hidden state $\mathbf{z}_0$ is the summation of the final states from the two encoders. For the plan encoder, we use the final state $\mathbf{H}^T$ of LSTM as the context representation. For the graph encoder, we use an average of all the hidden states following a two-layer perceptron to produce the final state.

5 Experiments

We conduct experiments to evaluate our Planner (Section 5.2) and the overall generation system (Section 5.3). [4]

5.1 Dataset

We conduct experiments on the WebNLG dataset (Gardent et al., 2017; Castro Ferreira et al., 2018) used in the WebNLG challenge.[5] For each instance, the input is a set of up to 7 RDF triples from DBPedia, and the output is their text descriptions. Each triple-set is paired with a set of (up to three) human-generated reference texts. Each reference is also paired with the order of triples it realized. We use them to train and evaluate our Planner. Overall, the dataset contains $9,674$ unique triple-sets and $25,298$ text references, and is divided into training, development, and test set. The test set contains two subsets, the SEEN part where the instances belong to one of the nine domains that are seen in the training and development set (such as *Astronaut* and *Food*), and the UNSEEN part where the instances are from the other five unseen domains. The UNSEEN part is designed to evaluate models' generalizability to out-of-domain instances.

5.2 Experiments on Plan Generation

As previous work suggests, planning plays a crucial role in text generation. We, therefore, first investigate the performance of our planner.

5.2.1 Setup

During the graph encoding, we initialize the node embeddings with 100-dimensional random vectors. Our GCN model has two layers, with the hidden size of each layer as 100. The activation function is ReLU (Nair and Hinton, 2010). We optimize the training objective using Adam (Kingma and Ba, 2015) with a learning rate of 0.001 and an early stopping on the development set. The batch size is 100. We compare our results with the following six baseline planners:

- Random: returns a random permutation of the input triples as a plan;
- Structure-Random: returns a random traversal over the input graph. We report the highest score among three random strategies: random walk, random BFS, and random DFS;
- Step-By-Step (Moryossef et al., 2019a): a transition-based statistical ranking method;
- Step-By-Step II (Moryossef et al., 2019b): a DFS-based method with a neural controller;
- GRU & Transformer (Ferreira et al., 2019): two neural Seq2Seq methods with attention;

We report the performance on three test sets: SEEN, UNSEEN, and ALL (SEEN & UNSEEN). We remove all one-triple instances for planner's evaluation since the planning for these instances is trivial. Results are evaluated with accuracy and BLEU-n (Papineni et al., 2002). For accuracy, we regard a plan as correct only if it exactly matches one of the human-generated plans. BLEU-n is more forgiving than accuracy. It is also adopted in Yao et al. (2019) for plan evaluation. Here we choose $n = 2$.

5.2.2 Results

Table 1 shows results of the planning experiments. Our GCN method significantly outperforms all the baselines (approximate randomization (Noreen, 1989; Chinchor, 1992), $p < 0.05$) by a large margin on all the test sets and both measures, indicating the effectiveness of our planner. The most competitive baseline on ALL and UNSEEN sets is Step-By-Step, but our method is more time-efficient. For example, Step-By-Step needs 250 seconds to solve one 7-triple instance, but our method solves all 4928 instances in less than 10 seconds. For the SEEN set, the most competitive models are GRU and Transformer. However, while their accuracies drop by 0.46 on UNSEEN test set, our method drops only slightly by 0.02, indicating our method's better generalization power.

We believe that this superior generalization capacity comes from the modeling of the graph structure. While the surface forms of triples in UNSEEN set do not overlap with those in the training data, the graph-level structural features are still shared, making it a key factor for generalization. GRU and Transformer linearize the graph as a sequential input, making them miss the structural information and resulting in poorer generalization capacity. Step-By-Step II also considers graph structure, but our model achieves better performance because we use GCN to encode the node representation, which can aggregate richer information from both the graph structure and the surface information.

We also investigated the effect of the graph size on the plan quality. In Figure 5, we separate the ALL test set into six subsets according to the size of input triple-sets, to reflect the model's capacity

[4] Code is available on https://github.com/zhaochaocs/DualEnc

[5] http://webnlg.loria.fr/pages/index.html

	Accuracy			BLEU-2		
	SEEN	UNSEEN	ALL	SEEN	UNSEEN	ALL
Random	0.28	0.34	0.31	54.1	62.1	57.9
Structure-random	0.32	0.38	0.34	56.6	62.9	59.5
Transformer (Ferreira et al., 2019)	0.56	0.09	0.34	74.3	20.9	49.3
GRU (Ferreira et al., 2019)	0.56	0.10	0.35	75.8	25.4	52.2
Step-By-Step II (Moryossef et al., 2019b)	0.45	0.44	0.44	67.7	67.3	67.5
Step-By-Step (Moryossef et al., 2019a)	0.49	0.44	0.47	73.2	68.0	70.8
GCN	**0.63**	**0.61**	**0.62**	**80.8**	**79.3**	**80.1**

Table 1: Planning results of three test sets evaluated by accuracy and BLEU-2.

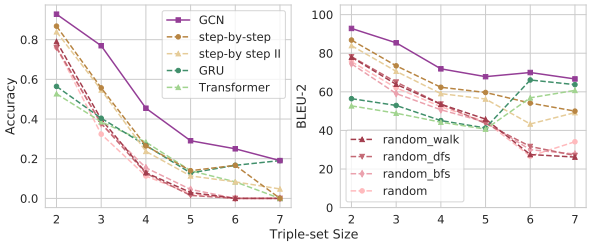

Figure 5: Fine-grained planning results for the ALL test set. Our method outperforms all the baselines regardless of the triple size.

at a fine-grained level. Fewer input triples make the planning task easier, while the 7-triple case is the most difficult one. The accuracy of seven out of eight baselines drops to around 0 in this case, while our method achieves an accuracy of 0.19. Besides this, our method consistently outperforms all the baselines for all the triple-set sizes.

5.3 Experiments on Text Generation

This section investigates the ability of our models to improve the generation quality.

5.3.1 Setup

We implement the generator based on the OpenNMT toolkit.[6] For the graph encoder, we use a similar setting as above. Since the generation task is more complicated than planning, we increase the dimension of the input and the hidden states to 256. The plan encoder is a 2-layer bidirectional LSTM with the same dimension setting of the GCN to ease the information fusion. During encoding, for UNSEEN test set, we adopt delexicalization (Gardent et al., 2017) to enhance the model's generalizability to unseen domains.

We use Adam with a batch size of 64. The initial learning rate is set to 0.001 and is decayed with a rate of 0.7 after the eighth epoch. We continue the training until the perplexity of the development set does not decrease. We also apply dropout on the decoding output layer with a rate of 0.3.

The quality of the generated text (as well as those of the baselines) is evaluated through a variety of automatic measures, such as BLEU, METEOR, and TER, which are strictly the same as those applied in the official challenge.[7] Following Marcheggiani and Perez-Beltrachini (2018), we report averaged performances over ten runs of the models.

We compare our method with the top systems of the WebNLG challenge and published state-of-the-art systems. The WebNLG systems are:

- ADAPT: a neural system with sub-word representations to deal with rare words and sparsity.
- TILB-SMT: a statistical machine translation method using Moses and delexicalization.
- MELBOURNE: a Seq2Seq model with enriched delexicalization from DBPedia.

The published research models are:

- GTR-LSTM (Trisedya et al., 2018): a graph-based triple encoder;
- GCN-EC (Marcheggiani and Perez-Beltrachini, 2018): a GCN-based triple encoder with glove embedding and copy;
- GRU & Transformer (Ferreira et al., 2019): two pipeline methods with 5 sequential steps and GRU or Transformer as the encoder;
- STEP-BY-STEP (Moryossef et al., 2019a): a pipeline method that generates the text from plans with OpenNMT and a copy mechanism.

5.3.2 Qualitative Results

Table 2 shows the results of the automatic evaluation on the generation task. Our PLANENC achieves the best performance on BLEU and TER, while DUALENC performs best under METEOR. Both PLANENC and DUALENC significantly out-

[6] https://github.com/OpenNMT/OpenNMT-py

[7] That is why some of the numbers in our table are not exactly the same as those in the cited works.

	BLEU (↑)			METEOR (↑)			TER (↓)		
	SEEN	UNSEEN	ALL	SEEN	UNSEEN	ALL	SEEN	UNSEEN	ALL
TILB-SMT	54.29	29.88	44.28	0.42	0.33	0.38	0.47	0.61	0.53
ADAPT	60.59	10.53	31.06	0.44	0.19	0.31	0.37	1.40	0.84
MELBOURNE	54.52	33.27	45.13	0.41	0.33	0.37	0.40	0.55	0.47
GTR-LSTM (2018)	54.00	29.20	37.10	0.37	0.28	0.31	0.45	0.60	0.55
GCN-EC (2018)	55.90	-	-	0.39	-	-	0.41	-	-
GRU (2019)	56.09	25.12	42.73	0.42	0.22	0.33	0.39	0.64	0.51
Transformer (2019)	56.28	23.04	42.41	0.42	0.21	0.32	0.39	0.63	0.50
Step-By-Step (2019a)	53.30	34.41	47.24	0.44	0.34	0.39	0.47	0.56	0.51
PLANENC	**64.42**	**38.23**	**52.78**	0.45	0.37	0.41	**0.33**	**0.53**	**0.42**
DUALENC	**63.45**	36.73	51.42	**0.46**	**0.37**	**0.41**	**0.34**	0.55	**0.44**

Table 2: Generation results evaluated by BLEU, METEOR, and TER. We compare our methods with different generation systems (SMT, Sequential NMT, Graph NMT, Pipeline). Both of our methods outperform all the baselines on all three measures. We highlight both results if there is no significant difference.

perform the previous state-of-the-art (bootstrapping (Koehn and Monz, 2006), $p < 0.05$). For the SEEN part, while no existing published work performed better than ADAPT, our PLANENC achieves a 3.83 performance gain on BLEU. It also outperforms the single GCN encoder by 8.52 BLEU, which confirms the advantage of the planning stage for bridging the structural gap between the encoder and decoder. For the UNSEEN part, PLANENC and DUALENC improve BLEU by 3.82 and 2.32 compared with the previous state-of-the-art. While it is difficult to distinguish the performance of DUALENC and PLANENC by automatic measures, our human experiments (see Section 5.3.4) show that dual encoding generates better text compared with PLANENC.

When comparing with the pipeline methods, one difference from the data perspective is how to obtain the plans of each instance to train the planner. While Step-By-Step uses heuristic string matching to extract plans from the referenced sentences, other methods (GRU and transformer), as well as ours, use plans provided in the enriched WebNLG dataset (Castro Ferreira et al., 2018). However, Step-By-Step reported worse BLEU results on these plans.

5.3.3 Ablation Study

To further analyze what factors contribute to the performance gain, we conduct an ablation study by removing the following components:

- Copy mechanism: The text is generated without copying from the source;
- Triple planning: The input triples are shuffled before feeding into RNN, but the (s, p, o) inside a triple are not shuffled.
- Entity mentions: We join the words in a node mention with underlines (e.g., *Aston_Martin* instead of *Aston Martin*).
- Plan delimiter: We concatenate the (s, p, o) without separating them with role delimiters.

Methods	BLEU (↑)	METEOR (↑)	TER (↓)
PLANENC	**64.42** $\pm$ 0.17	**0.45** $\pm$ 0.00	**0.33** $\pm$ 0.00
-plan	57.81 $\pm$ 0.82	0.40 $\pm$ 0.00	0.40 $\pm$ 0.01
-copy	61.64 $\pm$ 0.53	0.43 $\pm$ 0.01	0.36 $\pm$ 0.01
-mention	61.49 $\pm$ 0.35	0.43 $\pm$ 0.00	0.36 $\pm$ 0.00
-delimiter	63.26 $\pm$ 0.33	0.44 $\pm$ 0.00	0.34 $\pm$ 0.00

Table 3: Results of the ablation study.

We conduct the ablation study on the SEEN testset using our PLANENC. Table 3 shows the average performance and standard deviations. Compared with PLANENC, replacing plans with a random sequence of triples hurts the BLEU score by 6.61 points, indicating that the accuracy of planning is essential for the quality of generation. Our planning also makes the model more stable to random seeds (by decreasing the standard deviation from 0.82 to 0.17). Removing the copy mechanism also decreases the BLEU score by 2.78 points. It demonstrates the effectiveness of copying words from the source triples rather than generating them from the vocabulary set. Removing the mention information, decreases the BLEU score by 2.93. It reflects two benefits of word mentions: to alleviate data sparsity and to coordinate with the copy mechanism. However, removing delimiters does not affect the BLEU much. Intuitively, we expected the delimiters to

	Absolute(%)		Pairwise(%)			
	CVGE	FAITH	CVGE	FAITH	FLCY	ALL
MELBOURNE	83.0	75.2	-35.0	-42.5	-38.8	-68.8
STEP	**96.1**	**89.3**	**5.0**	**-3.7**	-45.0	-55.0
E2E-TRANS	85.5	78.0	-21.2	-32.5	-21.2	-46.3
GCN	79.8	76.8	-48.7	-50.0	-26.3	-67.5
PLANENC	**92.3**	**88.2**	**-7.5**	**-12.5**	**-7.5**	-21.2
DUALENC	**94.5**	**91.8**	–	–	–	–

Table 4: Results of human evaluation. DUALENC outperforms most of the baselines on all measures.

help the LSTM capture the boundaries and semantic roles of each node, but the ablation study does not support it. We provide an example in Table 5 to show that the LSTM indeed has trouble learning such semantic roles.

5.3.4 Human Evaluation

Automatic measures are based on lexical similarities and are not good measures of text quality in general. We therefore further conduct a human evaluation on Amazon Mechanical Turk to better access the quality of the generated texts. We evaluate the results for MELBOURNE, Step-By-Step, Transformer, GCN, as well as our PLANENC and DUALENC. We randomly select 80 test instances (440 triples in total) with the size of tripleset between 4 to 7, since they are more challenging than those with fewer triples. Then we evaluate the generation quality of each system with the following three measures:

- Coverage: the percentage of triples that are covered by the generated text (all $< s, p, o >$ values in the triples are realized);
- Faithfulness: the percentage of triples that are faithfully described by the text (the text correctly expresses the predicate and also the subject and object as its arguments. No substitutions or hallucinations);
- Fluency: a measure of the fluency or naturalness of the generated text.

For coverage and faithfulness, workers are asked to check each triple of an instance, and judge whether the triple is covered and faithfully described by the generated text. For fluency, we ask another group of workers to compare between two outputs of the same instance and identify which one is more fluent. Table 5 shows examples where these qualities are compromised.

In Table 4, we report the absolute scores of coverage and faithfulness, which range from 0 to 100%. We also provide pairwise scores of all three measures by comparing the outputs of DUALENC with each of the other five systems. We report the percentage of instances that were judged to be worse/better/same than those of DUALENC, yielding a score ranging from -100% (unanimously worse) to 100% (unanimously better). For example, MELBOURNE performs better/worse/same than DUALENC for 10%/45%/45% of the instances, yielding a pairwise score as 10%-45%=-0.35%. We also report an overall pairwise score combining all three measures. For each instance, the overall score of one output is higher than the other iff it outperforms the other on at least one of the three measures and has a better or equal vote on the other two.

Our PLANENC and DUALENC outperform most of the baselines on all of the measures by a large margin (approximate randomization, $p < 0.05$.), which is consistent with the automatic results. The only exception is Step-By-Step, which has high Coverage and Faithfulness (not significant). It first separates the input triples into smaller subsets and then realizes them separately. This greatly reduces the difficulty of long-term generation but at the expense of Fluency (worst among all the baselines). GCN does not perform well on Coverage, which demonstrates that the structural gap between encoding and decoding indeed makes generation more difficult. However, it has the smallest difference between Coverage and Faithfulness among all the baselines, indicating that the fidelity of generation can benefit from the encoding of graph-level structural information. By combining GCN and PLANENC, our DUALENC incorporates the advantages of both encoders while ameliorating their weaknesses, and therefore achieves the best OVERALL performance on human evaluation.

5.4 Qualitative Analysis

Table 5 shows examples of generated texts by various systems for an input of six triples. Colored fonts represent missing, unfaithful, and unfluent information. For example, PLANENC misses "Buzz Aldrin" and also wrongly expresses the subject of "retirement" as "Frank Borman", indicating that LSTM is less powerful at capturing the semantic roles of entities. This disadvantage can be well complemented by GCN, which is designed to capture the graph structure and the relations between entities. Hence, by incorporating information from

Tripleset	(William Anders | birthPlace | British Hong Kong), (William Anders | was a crew member of | Apollo 8), (Apollo 8 | crewMembers | Frank Borman), (Apollo 8 | backup pilot | Buzz Aldrin), (Apollo 8 | operator | NASA), (William Anders | dateOfRetirement | 1969-09-01)
MELBOURNE	william anders (born in british hong kong) was a crew member of **apollo 8' s apollo 8 8** mission along with buzz aldrin as backup pilot and **buzz aldrin on 1969-09-01 . [Frank Borman, NASA]**
Step-by-Step	william anders was a crew member of apollo 8 operated by nasa. apollo 8' s backup pilot was buzz aldrin and **frank borman**. william anders was born in british hong kong. william anders retired on september 01st, 1969.
PLANENC	william anders was born in british hong kong and was a crew member of nasa' s apollo 8. frank borman was a crew members of apollo 8 and **he** retired on september 1st, 1969 . **[Buzz Aldrin]**
DUALENC	william anders was born in british hong kong and served as a crew member of nasa' s apollo 8 along with frank borman and backup pilot buzz aldrin. he retired on september 1st, 1969 .
Reference	william anders was born in british hong kong and served as a crew member on apollo 8 along with frank borman. nasa operated apollo 8, where buzz aldrin was a back up pilot. anders retired on sept 1, 1969 .

Table 5: Sample texts generated by our methods and baselines, compared with a human-provided reference. We highlight in different color the **[missing]**, **unfaithful**, and **unfluent** parts of each text. Only the results of our DUALENC correctly mention all the input triples.

both GCN and LSTM, DUALENC correctly expresses the subject argument of "retirement".

6 Conclusion

This paper proposes DUALENC, a dual encoding method to bridge the structural gap between encoder and decoder for data-to-text generation. We use GCN encoders to capture the structural information of the data, which is essential for accurate planning and faithful generation. We also introduce an intermediate content planning stage to serialize the data and then encode it with an LSTM network. This serialized plan is more compatible with the output sequence, making the information alignment between the input and output easier. Experiments on WebNLG dataset demonstrate the effectiveness of our planner and generator by outperforming the previous state-of-the-art by a large margin. Future work will validate the effectiveness of this method on more varied data-to-text generation tasks.

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.

Arianna Bisazza and Marcello Federico. 2016. A survey of word reordering in statistical machine translation: Computational models and language phenomena. *Computational Linguistics*, 42(2):163–205.

Thiago Castro Ferreira, Diego Moussallem, Sander Wubben, and Emiel Krahmer. 2018. Enriching the webnlg corpus. In *Proceedings of the 11th International Conference on Natural Language Generation*, INLG'18, Tilburg, The Netherlands. Association for Computational Linguistics.

Nancy Chinchor. 1992. The statistical significance of the muc-4 results. In *Proceedings of the 4th conference on Message understanding*, pages 30–50. Association for Computational Linguistics.

Emilie Colin, Claire Gardent, Yassine M'rabet, Shashi Narayan, and Laura Perez-Beltrachini. 2016. The webnlg challenge: Generating text from dbpedia data. In *Proceedings of the 9th International Natural Language Generation conference*, pages 163–167.

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.

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.

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.

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.

Shizhu He, Cao Liu, Kang Liu, and Jun Zhao. 2017. Generating natural answers by incorporating copying and retrieving mechanisms in sequence-to-sequence learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 199–208.

Xinyu Hua and Lu Wang. 2019. Sentence-level content planning and style specification for neural 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 591–602.

Juraj Juraska, Panagiotis Karagiannis, Kevin Bowden, and Marilyn Walker. 2018. A deep ensemble model with slot alignment for sequence-to-sequence natural language generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 152–162.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Philipp Koehn and Christof Monz. 2006. Manual and automatic evaluation of machine translation between european languages. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 102–121.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213.

Wei Li, Jingjing Xu, Yancheng He, Shengli Yan, Yunfang Wu, et al. 2019. Coherent comment generation for chinese articles with a graph-to-sequence model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4843–4852.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Shuman Liu, Hongshen Chen, Zhaochun Ren, Yang Feng, Qun Liu, and Dawei Yin. 2018. Knowledge diffusion for neural dialogue generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1489–1498.

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.

Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. 2019. Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 845–854.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019a. 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.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019b. improving quality and efficiency in plan-based neural data-to-text generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 377–382.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Eric W Noreen. 1989. *Computer-intensive methods for testing hypotheses*. Wiley New York.

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*, pages 311–318. Association for Computational Linguistics.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6908–6915.

Lena Reed, Shereen Oraby, and Marilyn Walker. 2018. Can neural generators for dialogue learn sentence planning and discourse structuring? *INLG 2018*, page 284.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge university press.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, 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. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

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.

Amanda Stent, Rashmi Prassad, and Marilyn Walker. 2004. Trainable sentence planning for complex information presentations in spoken dialog systems. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 79–86.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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)*, volume 1, pages 1627–1637.

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

Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-and-write: Towards better automatic storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7378–7385.