A Novel GCN Architecture for Text Generation from Knowledge Graphs: Full Node Embedded Strategy and Context Gate with Copy and Penalty Mechanism

Zhongqiang Hu\textsuperscript{a}, Weiwen Zhang\textsuperscript{b,}\textsuperscript{*}, Depei Wang\textsuperscript{a}, Weicai Niu\textsuperscript{b}, Fei Mo\textsuperscript{b}, Jianwen Ma\textsuperscript{a}, Guoheng Huang\textsuperscript{b,} and Lianglun Cheng\textsuperscript{b}

\textsuperscript{a}School of Automation, Guangdong University of Technology, Guangzhou 510006, China
\textsuperscript{b}School of Computers, Guangdong University of Technology, Guangzhou 510006, China

Abstract. Text generation from knowledge graphs is a fundamental task, which aims to map triplets to description text. Previous research mostly adopts standard sequence-to-sequence methods, which would inevitably fail to capture graph structure information. In this paper, we propose a novel neural network architecture called GCN-FCCP, which is based on Graph Convolutional Network enabled by a Full node embedded strategy and Context gates with Copy and Penalty mechanism. The full node embedded strategy embeds each word in the input triplets as a new node to enhance the graph information, while a stacked multi-layer graph convolutional network is used as the encoder to directly exploit the input structure. For the decoder, we integrate a context gate into the LSTM network to retain the information of contexts during the hidden state updating process, which ensures the faithfulness to the original meaning. Meanwhile, we add copy attention and penalty mechanism to the decoder to solve the Out-of-vocabulary (OOV) problem and improve the quality of the generated sentences. Extensive experiments on the WebNLG dataset show that GCN-FCCP can effectively generate high-quality text from graph-structured input, which obtains high scores in four automatic metrics.

Keywords: Knowledge Graph; Text Generation; Graph convolutional network; Encoder-decoder architecture; Penalty mechanism

1. Introduction

The task of text generation from Knowledge Graph (KG) \cite{1} originates from the WebNLG challenge \cite{2-3}, which aims to generate descriptions of entities from a set of Resource Description Framework (RDF) triplets. Such a text generation brings great significance to knowledge question and answer \cite{4}, data-to-document generation \cite{5}, and recommender system \cite{6}, which enables the information to be more available for end-users. However, the input graphs are diverse, representing knowledge from different domains and in different ways. Moreover, document planning issues, such as order, coherence, and discourse markers, should be considered for generating a concise and faithful description text \cite{2-3}. Therefore, it still remains challenging for the KG-to-Text task.

The mainstream methods for tackling the KG-to-Text task are based on the neural network method \cite{7-9,10,11}.
which leverages the encoder and decoder architecture [11-12]. It regards the task as the sequence-to-sequence problem, using LSTM [13] encoder and gating mechanism to linearize the input triplets. Although some of these encoders can achieve good results, information about internal graph structure is lost because of linearizing the RDF data directly. Moreover, the Out-of-vocabulary (OOV) problem in text generation of the encoder and decoder architecture is remained to be solved. In addition, generating the faithful output is challenging due to the structural gap of data between the knowledge graph and the generated text [14]. Although the triplets can be well-documented for some pipeline methods, the fidelity and fluency of the generated sentences of triplets still need to be improved.

In this paper, we have made improvements on the general Graph Convolutional Network (GCN) encoder architecture and propose a new model GCN-FCCP based on Full node embedded strategy and Context gates with Copy and Penalty mechanism. First, to process the graph data more efficiently, we present a full node embedded strategy to explore the information of nodes in the graph, which splits the nodes of the input graph and builds a new graph structure. To obtain a better word feature representation, we fine-tune the pre-trained word embedding model BERT to generate embedding pre-trained word vectors for the nodes. Following that, a stacked multi-layer GCN is employed as an encoder to encode each node in the graph. We add dense connection to stack a single GCN layer in multiple layers to make full use of the information between the further nodes of the graph while ensuring the conduction of the gradient. To improve the faithfulness of text generation from knowledge graphs, we integrate a context gate in the design of the decoder network. In addition, we utilize the copy mechanism and penalty mechanism to the GCN architecture to generate the target text. The main contributions of our work are summarized as follows:

- We propose a full node embedded strategy to construct a new graph and enhance graph information processing of the model that permits to model an arbitrary number of KGs efficiently.
- We add a context gate in the decoder to enable the output to be more faithful to the original meaning while ensuring its smoothness.
- To tackle the OOV problem and further improve the quality of the generated sentences in the architecture, we adopt the copy mechanism and penalty mechanism into the decoder to control the length of a generated sentence.

Extensive experiments have shown that our method GCN-FCCP achieves good performance in the WebNLG benchmark dataset for text generation and improves the BLEU score by 2.02 points over the GTR-LSTM architecture [8]. The ROUGE$_{	ext{L}}$ score of GCN-FCCP is 8.2 points higher than that of the Melbourne model. Compared with directly embedding graph nodes, the adoption of the full node embedded strategy and BERT pre-training model for embedding allows the generation to be more adaptable for obtaining more graph information. By stacking multi-layer graph convolutional networks with the copy and penalty mechanism, GCN-FCCP can obtain higher scores than other baseline models among the four automatic evaluation metrics, including BLEU, METEOR, TER and ROUGE$_{	ext{L}}$.

The rest of the paper is organized as follows. In Section 2, we present the previous works of KG-to-Text task. Section 3 explains in details the architecture of GCN-FCCP. Section 4 introduces the parameter setting of GCN-FCCP and the performance comparison with other models as well as ablation studies. In Section 5, we summarize the paper and the future work.

2. Related Work

As a key technology of artificial intelligence, Knowledge Graph [1] has laid a solid foundation for knowledge interconnection and intelligent applications due to its rich semantic expression and open interconnection capabilities. KG-to-Text, one of the important tasks in natural language generation (NLG) [15], aims to generate text from the sub-graph structure data of the knowledge graph. At present, the methods for KG-to-Text mainly include pipelines and end-to-end methods based on neural networks. However, both of them still have the problem of generating word out-of-vocabulary to a certain extent, and the quality of sentences generated by triplets still needs to be improved.

In previous research work, the task of KG-to-Text was regarded as a sequence-to-sequence problem. Gardent et al. [3] used LSTM as an encoder to linearize the input triplets and added a gating mechanism so that the network can store long-distance information. In addition, the RDF2Vec method is proposed to learn the potential numerical representations of entities in RDF graphs [16]. Nevertheless, the direct linearization strategy renders the relation infor-
mation between the triplets unable to be preserved, resulting in a reduction of the coherence between the generated sentences. Distilawat et al. [8] developed a variant model of LSTM-based encoder called GTR-LSTM. It exploited a topological-sort and breadth-first search strategy to linearize the input graph to save the information of relations within and between triplets. GCN is a variant of graph neural network (GNN) [17], recently proposed by Kipf and Welling [18]. With the development on GCN, some researchers began to adopt it to the task of graph-structured input [7, 19]. Marcheggian and Perez Beltrachini [9] proposed an encoder based on GCN to consider the information between the input graphs, which directly encodes graph data and stacks multiple layers of GCN in the network. Its performance is better than using the LSTM [20]. More recently, Ribeiro and Zhang [10] designed the characteristics of GCN coding between non-contact nodes. Encoder architectures were proposed to combine local node coding and global node coding. Our work is similar to Marcheggian et al. [9], but we employ another word embedding method and improve the decoding layer.

The pipeline methods for the KG-to-Text task generally adopt specially-designed strategies. Some researchers divide the model into multiple components [21-22]. Moryossef et al. [21] proposed to split the generation task into two steps: planning and generation. First, it arranges and plans the input triplets, and then generates the text by the neural translation model that only pays attention to the generation. This strategy improved the authenticity and fluency of sentences. Moryossef et al. [23] improved the framework of step-by-step [21] and designed a trainable neural planner to automatically plan the input to speed up the inference of the model. Geng et al. [24] also divided the framework into two steps: an attentive ZSL learner and an explanation generator, and improved the precision of the model. On another note, copy mechanism in the pointer network of summary generation has been found to be beneficial in the generation task. However, only a small number of works used the pointer network, which is based on the graph transformers architecture [25-26]. It focuses on the relationship between directly connected nodes. Castro Ferreira et al. [20] compared the pipeline with the neural end-to-end method of text generation from Knowledge Graphs, which showed that the usage of explicit intermediate steps in the generation process would yield better results than the end-to-end generation method. Besides, the pipeline model can be better generalized to invisible inputs.

Thus, we adopt a full node embedded strategy to process the input graph before encodings, and try to make full use of the advantages of the pipeline in our model.

When developing a neural network model for the KG-to-Text task, it is crucial for the model to learn how to get a better hidden state representation of a word or a sentence. The most commonly used models of word embedding in various NLP tasks are word2vec and Glove [27-28], both of which are unsupervised methods based on the distribution hypothesis. In the research of word embedding, these unsupervised methods have made remarkable progress. Based on word2vec and Glove, FastText [29] and ELMo [30] are proposed respectively. FastText mainly improves the n-gram model based on character-level embedding, which allows the calculation of word representations that do not appear in the training datasets, while ELMo is a sentence-level relation embedding of deeper semantic relations. Subsequently, BERT [31] model uses the Transformer as the main framework of the algorithm. It can capture the two-way relation in the sentences more thoroughly. The BERT model has greatly improved the accuracy of 11 tasks in the NLP field.

In this work, we present a novel GCN architecture that is powered by the reconstruction of input graph structure and a decoder with a context gate and penalty mechanism. To resolve the problem of structural information loss caused by linearization in the encoder-decoder architecture, we adopt a novel stacked multi-layer GCN encoder. The encoder directly encodes the input graph-structured data. Meanwhile, based on the comparative study of the pipeline methods and the neural end-to-end methods, the appropriate addition of the pipeline can improve the performance of the model. Therefore, we propose a method of reconstructing graph embedding to process the input information more efficiently. Besides, we adopt a BERT pre-training model to generate word vectors for obtaining better word embeddings. From the research in sequence to sequence, it can be found that the context information is beneficial to the performance of the model [32-33]. Thus, we add a context gate in the decoder to obtain the context information. To deal with the problem of improving sentence quality, we adopt the copy mechanism in pointer network and penalty mechanism in the domain of neural machine translation. The proposed model can thus effectively deal with the OOV and fidelity problems and generate high-quality sentence descriptions for the complex graph-structured input.
3. Model

We address the task of text generation from graph-structured data of RDF triplets by designing a novel neural network GCN-FCCP. Formally, we consider a labeled graph \(X = \{E, V\}\), where \(E\) is a set of nodes in the graph and \(V\) is a set of edges between nodes in \(E\). The nodes represent the entities in the knowledge graph and the attributes of the edges represent the relation between two entities. The output natural language \(\{y_1, y_2, \ldots, y_n\}\) with \(n\) tokens is expressed by \(X\). Our proposed model follows the standard attention-based encoder-decoder architecture to learn the mapping between source encoding and target decoding. The overall architecture of our GCN-FCCP model is shown in Fig. 1.

It consists of four modules: full node process and word embedding before encoding, stacked multiple layers of GCN encoder with dense connection, two layers of LSTM and copy mechanism decoding layer, and the target text generation. We will introduce each module in details.

3.1. Full Node Embedded Strategy for Processing Triplets Input

For processing triplets input, we propose a full node embedded strategy to determine entity boundaries and relationship boundaries for the correct encoding and generation of information. Generally, delexicalisation is used in the graph-structured data processing of the previous method [9, 21], where the slot-value in the sentence is replaced with a specific predefined category token. Then lexicalisation is used to replace the token with the original value at the generation stage. Instead of using delexicalisation in data processing, we embed each word in the input graph as a separate node and mark the connecting edges of different nodes with specific labels. The full node process is shown in Fig. 2.

Given a set of input triplets \(\{[e_1^1, r_1, e_1^2], [e_2^1, r_2, e_2^2], \ldots, [e_r^1, r_r, e_r^2]\}\), we extract the entities and relations while retaining their direction information. We tokenize the entities and relations and obtain the token sequence of each part, such as \(\{[e_{11}, e_{12}, \ldots, e_{1j}], [r_{11}, r_{12}, \ldots, r_{1j}], [e_{21}, e_{22}, \ldots, e_{2j}], \ldots, [e_{r1}, e_{r2}, \ldots, e_{rj}]\}\), where \(j\) is the length of the entity \(e_i^j\). Then, we obtain three sequences: nodes, labels, and indexes based on the word tokenization results.
We extract the first token in each token sequence which belongs to the same triplet. Following that, we add the remaining parts of the triplet to the end of the sequence subsequently, which structures the nodes and incorporates the relation between nodes into the node encoder. We design the new labels \{A_i, A_i, NE\} instead of the original relation tags in the graph. We denote A, A, to represent the head token in each triplet, as shown in Fig. 2. The direction between the entity node and the relation node is \( A \rightarrow A \). The rest of this triplet is marked as NE according to the original order of relation. Finally, the index sequence maps the node to the label. The same node shares the same index. For example, the entity \( e_i \) can be divided into \([e_{i_1}, e_{i_2}, \ldots, e_{i_j}]\), all of which have the same index of 0. To this end, we complete the full-node graph processing from the input triplets and thus generate a new input graph.

Fig. 3 shows an example of full node embedded strategy processing on the original input triplets. The input graph-structured data is \{\{Amatriciana_sauce, country, Italy\}, \{Italy, language, Italian_language\}\} which has two sets of triplets. The new graph structure generated after processing is shown on the right-hand side in Fig. 3. The relations in this example are all single words. The relations information can be more clearly modeled under the processing when there are multiple words of relations in the triplets.

We adopt the BERT pre-training model to obtain the embedding word vectors. BERT leverages Transformer as the main framework, which can capture the two-way relationship in the sentence. In addition, it is trained on a massive corpus and shown to improve the accuracy in multiple tasks in NLP domain. Thus, to enhance the representation of the words, we fine-tune a BERT pre-training model to generate embedding pre-training word vectors.

### 3.2. Encoder Architecture by Stacked GCN

For the input of graph structure data, direct linearization will result in loss of graph structure information. In this paper, we adopt a stacked GCN encoder for graph-structured data to obtain the representation of each node in the graph. We adopt this encoder instead of RNN, directly taking the knowledge graph subgraph data \( X = \{E, V\} \) as input.
We denote each entity $e \in E$ as a $d$-dimensional feature vector $x_e \in \mathbb{R}^d$. With the full node embedded processing of the graph data, we update the node representation using adjacent nodes in the new graph layer by layer. Then, the stacked GCN encoder calculates and updates the feature representation $\tilde{h}_v$ of node $v$ as

$$
\tilde{h}_v = \text{RELU} \left( \sum_{w \in N(v)} g_{w,v}(W_{(w,v)}h_u + b_{(w,v)}) \right),
$$

where $u \in N(v)$ are all adjacent nodes of the node $v$. $W_{(u,v)} \in \mathbb{R}^{d \times d}$ is a weight matrix with node direction features of edge $(u,v)$. The vector $h_{(u,v)} \in \mathbb{R}^d$ denotes the label embedding of each edge between nodes. $g_{u,v}$ is a gate unit that performs weighted scalar learning on the importance of each edge $(u,v)$, and when multi-layer GCN networks are stacked, $g_{u,v}$ can increase the flexibility of the network. Thus, we adopt this gate unit into our network. Finally, the feature representation $h_v$ of the node $v$ is calculated and updated by the feature representation $\tilde{h}_u$ of the adjacent node $u \in N(v)$.

We stack the GCN network in multiple layers such that non-adjacent nodes can obtain the information about further graph nodes for more hidden information. Currently, two methods have been proposed to make the gradient flow through the stacked network layers more fluent and assist the hidden state transfer. One is a residual connections method, which directly sums the hidden layer states in the multilayer stacked network to obtain the state representation of the next layer $h'_t = h_t + h_{t-1}$ [34]. The other is dense connection that connects all the output feature of previous layer in the network as the input of the next layer $h'_t = [h'_t; h_t]$ [35]. It has been reported that using the residual connection in graph-to-text generation task may affect the feature representation of word embedding [9]. Therefore, we adopt the dense connection method for hidden layer state transfer of multi-layer GCN in our architecture.

3.3. Decoder Architecture with Context Gate and Copy Mechanism

The decoder architecture of GCN-FCCP consists of a global attention [36] and a two-layer LSTM. During the decoding process, the generator generates a corresponding word $y_t$ at each time step $t$. In general, the word generation $y_{t+1}$ of the next time step is calculated from the feature representation matrix $w$ of the previous $t$-th generated words and a context vector $c_t$ that is dynamically generated in GCN encoding

$$
P(y_t | y_{t-1}, X) = \text{softmax}(g(w_t, c_t)),
$$

where $g$ is the neural network layer.

We integrate a context gate into the LSTM network to ensure the decoder to be more faithful to the original meaning during the hidden layer status update process. That can ensure the decoding process to be more faithful to the original meaning and balance the fluency and sufficiency of the decoded values. Besides, the performance of NMT and BLEU scores can be improved by using an encoder-decoder framework that combines gates and attention mechanisms according to the report of the NMT research [37-38]. The gate structure of GCN-FCCP is shown in Fig. 4.

![Fig. 4. Context gate unit structure used in GCN-FCCP architecture.](image)

We introduce a weight variable

$$
q_t = \text{sigmoid} \left( O_q \cdot E(y_{t-1}) + U_q h_{t-1} + C_q h'_t \right),
$$

where $O_q, U_q, C_q \in \mathbb{R}^{d \times d}$ are the trainable weight matrices, $E(y_{t-1})$ is the word vector of the word generated at time step $t$, and the final output result is $q_t \in \mathbb{R}^d$. Similar to the forget gate in LSTM, when the result output is 1, it indicates that this part of the information is completely reserved; when the result output is 0, it indicates that this part of the information is forgotten.

We follow the context gate strategy [37], and the updated hidden layer state is given by
\[ h_t = f \left( (1 - q_t) \odot (O_q \cdot E(y_{t-1}) + U_q h_{t-1}) + q_t \odot C_q h_t \right) , \]

where \( \odot \) is an element-wise multiplication and \( f \) is our network layer. When the partial generation has been obtained, the gate will focus on the context and decide to rely more on the information of contexts. Therefore, the gate will assign more weight to the source context, but less weight to the target context before feeding them into the decoding activation layer. By the context gate, we can obtain smooth output to ensure the fidelity of the sentences.

Meanwhile, we adopt the copy mechanism in the pointer network summary generation [39] in the decoding process. By copy attention, the model can choose to copy words from the source sequence or look up in the vocabulary when generating words. This strategy can ensure obtaining better sentence expressions and solving some OOV problems. Besides, we introduce a variable \( z_t \in [0,1] \) at each time step \( t \), which indicates whether the model should copy words from the original input. Then, the generation probability is given by

\[
P(y_t, z_t | y_{1:t-1}, X) = \sum_{z \in \{0,1\}} P(y_t, z_t = z | y_{1:t-1}, X).
\]

We adopt a trainable parameter \( v \in \mathbb{R}^d \) and the hidden state \( h_t \) of the decoding layer as input in the generation to obtain the copy probability \( P(z_t) = \sigma(h_t^T v) \). In each time step, \( P(y_t, z_t = 0) \) means that the word is generated from the vocabulary; while \( P(y_t, z_t = 1) \) means that the model chooses to copy the word in the original input as the output. We utilize the negative log-likelihood as the loss function \( L \) to optimize the model during the training process

\[
L = \sum_{t=1}^{l} \log P(y_t | y_{1:t-1}, X).
\]

### 3.4. Score Function with Penalty Mechanism

We exploit beam search in the generator to obtain the best sequence \( Y \) corresponding to the score function \( s(Y, X) \). We add the coverage penalty and length normalization commonly used in the NMT system [40] to improve the quality of the generated sentences. In addition, we add the penalty mechanism to enable the model to generate longer sentences and solve the problem of the repeated generation or missing information. The results of conventional beam search are more inclined to generate short sentences because the log probability results of long sentences are relatively lower. Thus, the original score item is divided by the sentence length penalty item for normalization. We define the length penalty term \( p_l \) as

\[
p_l(Y) = \frac{(5 + |Y|)^\alpha}{(5 + 1)^\alpha}
\]

where \( \alpha \in [0,1] \) is a parameter that controls the strength of the length penalty [38].

Similarly, the definition of the coverage penalty term \( p_c \) is the same as the research of Wu et al. [38]

\[
p_c(X, Y) = \beta \cdot \sum_{i=1}^{l} \log(\min\{\sum_{i'=1}^{y} a_{i'}, 1.0\}),
\]

where \( \beta \in [0,1] \) is a penalty item parameter, which controls the degree of penalty. \( a_{i'} \) represents the attention item of the target word \( y_t \) at the \( t \)-th time step to the \( i \)-th source word \( x_t \). When we generate a target word containing multiple identical words for a certain set of input data, this penalty term will be aggravated. Hence, the score function for beam search is defined as

\[
s(X, Y, z) = \frac{\log p(y_t, z_i | x) + p_l(Y)}{p_c(Y)} + p_c(X, Y),
\]

where \( z \) is the copy item parameter. When \( \alpha = 0 \) and \( \beta = 0 \), we have \( p_l(Y) = 1 \) and \( p_c(X, Y) = 0 \), which means that the penalty mechanism is not activated.

### 4. Experiment

#### 4.1. Datasets

We train and test our generative model on the WebNLG dataset. The WebNLG dataset was released in the WebNLG Challenge 2017. There are several different natural language generation tasks in the challenge, including RDF data to text generation. For each instance, the input is a set of up to 7 RDF triplets, and the output is their text descriptions. The dataset covering 10 kinds of DBpedia categories (such as Politician, City, and Astronaut), includes 16,095 data inputs, 42,873 data-text pairs, and 373 types of relation. The dataset is divided into three parts: training set, dev set, and test set. The test set also contains two subsets of Seen and Unseen. The seen test data set is a category that has appeared in the training set and the unseen test data set has five other invisible categories. Here we only focus on the test part of seen.
4.2. Setup

We use a BERT pre-training model with Cased to learn the embedding of node words. The pre-training model network has 12 layers with 110M parameters, and the embedding dimension of hidden layer is 300. Cased means that the true case and accent markers are preserved, which has been shown to be beneficial in some tasks that emphasize the case. Meanwhile, we adopt the shared embedding to share the word embedding of the source input and target text to expand our vocabulary. We set the vocabulary size of words to 8000. The learning rate is set to 0.001 and then automatically attenuated with the gradient transmission of the training process. Besides, we add and set label smoothing as 0.2 and dropout rate as 0.3 to mitigate the problem of model overfitting. We adopt four kinds of automatic metrics: BLEU [41], METEOR [42], TER [43], and ROUGEL [44] to evaluate the performance of the model.

We conduct parameter tuning to explore the effect of the number of GCN stacked. We keep other parameter settings the same and test the effect of stacking GCN at different layers on performance. The results are shown in Fig. 5. The dense connection plays an important role in the stacking of multiple layers in GCN experiments. We observe that as the number of stacked layers in a sentence increases, the performance of None decreases.

Nevertheless, when the dense connection is added, BLEU is gradually increasing and the model achieves the best performance when the number of layers is 6. When the number of layers is greater than 6, the score will gradually decrease. Thus, GCN-FCCP adopts the dense connection method to stack 6 layers of GCN.

We also tune penalty parameters in GCN-FCCP. It is demonstrated that the model achieves better performance by setting the length penalty term parameter $\alpha$ and coverage penalty term parameter $\beta$ as $[0.2, 0.6]$ [38]. Thus, we only experiment with different penalty parameters for GCN-FCCP within this range. Specifically, we only adjust the penalty parameters without changing other model settings and the results are shown in Fig. 6. It can be observed from Fig. 6(a) that, the model obtains the best performance with 60.62 BLEU points when the penalty parameters are set to 0.2. When the penalty parameters are greater than 0.2, the BLEU points will gradually decrease. Fig. 6(b) shows that when the penalty parameters in a sentence increase from 0.0 to 0.2, the METEOR score will gradually increase to 0.428. When the parameters are larger than 0.2, the METEOR score saturates and fluctuates around 0.43. Fig. 6(c) shows the ROUGEL score under different setting of $\alpha$ and $\beta$, which has a similar trend of Fig. 6(a). Setting $\alpha$ and $\beta$ too high or too low can have a negative effect on the
model. Consequently, the length penalty parameter $\alpha$ and the coverage penalty parameter $\beta$ are set to $[0.2, 0.2]$ in GCN-F CCP.

4.3. Comparison with Baseline Models

For comparison, we apply the baseline models as follows:

- LSTM (2017) [2] is the baseline model in the WebNLG Challenge;
- UPF-FORGe [45] is one of the top systems in the WebNLG Challenge which follows a rule-based method;
- Melbourne is a Seq2Seq model with enriched delexicalization from DBPedia in the WebNLG Challenge;
- Marcheggiani and Perez (2018) [9] is a model that uses GCN encoder and Glove pre-trained word embedding;

In Table 1, we report the results of GCN-F CCP on the WebNLG test data in comparison with other models.

It can be observed from the experimental results listed in Table 1 that in this setting, our GCN-F CCP model achieves the best performance on the four automatic metrics. The GCN-F CCP with stacked GCN encoder outperforms the strong baseline that uses the LSTM encoder, with 6.41 BLEU points. This confirms the advantages of the GCN encoder that combines with the planning strategy of full node embedded in the graph-structured data. GCN-F CCP is also better than the baseline with a rule-based and a step by step strategy. Also, GCN-F CCP with context gate and penalty mechanism decoder architecture outperforms Marcheggiani and Perez (2018) that utilizes general decoder architecture, with 4.54 BLEU points improvement. Compared with the GTR-LSTM linear coding model, GCN-F CCP can effectively use structural information and outperform the GTR-LSTM model with 2.02 BLEU points. It demonstrates that the context gate increases the faithfulness of the sentence. In addition, the penalty mechanism increases the information coverage of the source sentences and length of sentences to obtain a higher BLEU score, which is related to the common n-gram between the sentences [41]. Furthermore, in the ROUGE evaluation method, GCN-F CCP achieves 8.2 points higher than Melbourne that ranks the second.

Table 2 shows some examples of the generated text on the WebNLG dataset. Lex is the target description text in the dataset, and the GCN-F CCP part corresponds to the text generated by our model. No Full means the full node embedded strategy is not used and adopt the traditional graph node embedded rules. It can be clearly observed that GCN-F CCP generates fluent sentences with complete information. Most of the multi-word entities in the generated texts have a subpar expression after removing the full node embedded strategy in the stacked multi-layer GCN encoder. We also observe that the generated sentences miss information and present false facts in Table 2. In addition, due to the mechanism of graph reconstruction, the GCN-F CCP model can generate more expressions for the entity in the sentence, but not just the one that is exactly the same entity in the input triplet. When the input triplets are more complex, the gap is more obvious. Therefore, the full node embedded strategy can resolve this issue well from the input and improve the ability of the model to generate high-quality sentences.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>METEOR</th>
<th>TER</th>
<th>ROUGE$_{L}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM (2017)</td>
<td>54.03</td>
<td>0.39</td>
<td>0.40</td>
<td>-</td>
</tr>
<tr>
<td>UPF-FORGe</td>
<td>40.88</td>
<td>0.40</td>
<td>-</td>
<td>60.9</td>
</tr>
<tr>
<td>Melbourne</td>
<td>54.52</td>
<td>0.41</td>
<td>0.40</td>
<td>63.5</td>
</tr>
<tr>
<td>Step-by-Step (2019)</td>
<td>47.40</td>
<td>0.391</td>
<td>-</td>
<td>63.1</td>
</tr>
<tr>
<td>Marcheggiani and Perez (2018)</td>
<td>55.90</td>
<td>0.39</td>
<td>0.41</td>
<td>-</td>
</tr>
<tr>
<td>Distiawan et al. (2018)</td>
<td>58.60</td>
<td>0.406</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GCN-F CCP (Ours)</td>
<td>60.44±0.18</td>
<td>0.428</td>
<td>0.41</td>
<td>71.79±0.11</td>
</tr>
</tbody>
</table>
Some examples of texts generated by the model. The “T” means the number of input triplets. Some of the entities in input triplets are marked in different colors.

| WebNLG (T=3) | (AC_Hotel_Bella_Sky_Copenhagen | tenant | Marriott_International) (AC_Hotel_Bella_Sky_Copenhagen | architect | 3XN) (AC_Hotel_Bella_Sky_Copenhagen | floorCount | 23) |
| Lex | Marriott International is the tenant of AC Hotel Bella Sky Copenhagen, that was designed by the architects of the 3XN firm and has 23 floors. |
| No Full | 3XN was the architect of the grounds of AC_Hotel_Bella_Sky_Copenhagen which has the Marriott_International tenant. |
| GCN-FCCP | Marriott International is the tenant of AC Hotel Bella Sky in Copenhagen which has 23 floors and was designed by the architect 3XN. |

| WebNLG (T=4) | (Ashgabat_International_Airport | operatingOrganisation | Turkmenistan_Airlines) (Turkmenistan_Airlines | headquarter | Turkmenistan) (Turkmenistan_Airlines | hubAirport | Turkmenbashli_International_Airport) (Turkmenistan_Airlines | headquarter | Ashgabat) |
| Lex | Turkmenistan Airlines have their HQ in Ashgabat, Turkmenistan and their hub airport at Turkmenbashli International airport. They are the operating organisation for Ashgabat International airport. |
| No Full | The hub airport for Turkmenistan_Airlines is Turkmenbashli_International_Airport. The hub airport for Turkmenistan_Airlines is Turkmenbashli_International_Airport. |
| GCN-FCCP | Ashgabat International Airport is operated by Turkmenistan Airlines which has its headquarters in Turkmenistan. The hub airport for Turkmenistan Airlines is Turkmenbashli_International_Airport. |

| WebNLG (T=6) | (Buzz_Aldrin | birthPlace | Glen_Ridge,_New_Jersey) (Buzz_Aldrin | was a crew member of | Apollo_11) (Buzz_Aldrin | birthDate | "1930-01-20") (Buzz_Aldrin | almaMater | "Massachusetts Institute of Technology, Sc.D. 1963") (Apollo_11 | backup pilot | William_Anders) (Apollo_11 | operator | NASA) |
| Lex | Buzz Aldrin was born on 20th January 1930 in Glen Ridge New Jersey. He graduated from MIT in 1963 and was a member of the Apollo 11 crew, operated by NASA. The backup pilot was William Anders. |
| No Full | Buzz_Aldrin was born in Glen_Ridge,_New_Jersey and graduated from Massachusetts_Institute_of_Technology, Sc. | D. | 1963 with an MA in the National Register of Historic Places. |
| GCN-FCCP | Buzz Aldrin was born in Glen_Ridge, New Jersey on January 20th, 1930. He graduated from Massachusetts Institute of Technology in 1963 with a Sc. D. He was a crew member on the NASA operated Apollo 11 mission along with backup pilot William Anders. |

4.4. Ablation Study

We conduct an ablation experiment on the GCN-FCCP model to investigate the influence of each module of the model on the performance and summarize the experimental results in Table 3. We start with an ablation on the encoder of the model, by removing the BERT pre-trained word embedding. It turns out that the BLEU score decreases by 2.44 points, which indicates that the pre-trained word embedding improves the performance. Besides, we notice that when the GCN network is stacked in multiple layers, directly embedding the graph without using full node embedded has a significant impact on the results of the model. The full node embedded can identify node and entity boundary more accurately, and thus capture relationship information as much as possible, which is beneficial to model encoding, decoding, and generation. For the decoder of the model, separately removing the penalty mechanism and the pre-trained word vectors embedding degrade the performance of GCN-FCCP.

Compared with the GCN-FCCP, removing the context gate and penalty mechanism hurt the BLEU score by 0.51 points and 1.57 points, indicating that the context gate and penalty mechanism are essential for the quality of generation. It can be observed that, when the copy attention is removed, the ROUGE$_L$ score reduces sharply by 10.6 points, which is calculated by the length of the longest common subsequence between the generated text and the reference text [44]. This indicates that the copy mechanism can effectively copy the word from the source and bring a great improvement to the model. Meanwhile, the experiment shows that the introduction of shared embedding can increase the size of the vocabulary and have a positive effect on the generation of the model.
In the future, we will consider adopting the word segmentation embedding method to divide the graph in more details. To further solve the OOV problem in the text generation, we will extend our method to other types of input graph data.

Acknowledgments

This research is supported by Program of Marine Economy Development (Six Marine Industries) Special Foundation of Department of Natural Resources of Guangdong Province under Grant GDNRG [2020]056, Key Program of NSFC-Guangdong Joint Funds under Grant U1701262 and U1801263, National Natural Science Foundation of China under Grant 62002071, Top Youth Talent Project of Zhujiang Talent Program under Grant 2019QTN01X516, R & D projects in key areas of Guangdong Province under Grant 2019B010153002, National Key R & D project under Grant 2019YFB1705503, R & D projects in key areas of Guangdong Province under Grant 2018B010109007 and Guangdong Provincial Key Laboratory of Cyber-Physical System under Grant 2016B030301008.

References
