GTR-LSTM: A Triple Encoder for Sentence Generation from RDF Data

Bayu Distiawan Trisedya[1], **Jianzhong Qi**[1], **Rui Zhang**[1*], **Wei Wang**[2]
[1] The University of Melbourne
[2] University of New South Wales
btrisedya@student.unimelb.edu.au
{jianzhong.qi,rui.zhang}@unimelb.edu.au
weiw@cse.unsw.edu.au

Abstract

A knowledge base is a large repository of facts that are mainly represented as RDF triples, each of which consists of a subject, a predicate (relationship), and an object. The RDF triple representation offers a simple interface for applications to access the facts. However, this representation is not in a natural language form, which is difficult for humans to understand. We address this problem by proposing a system to translate a set of RDF triples into natural sentences based on an encoder-decoder framework. To preserve as much information from RDF triples as possible, we propose a novel graph-based triple encoder. The proposed encoder encodes not only the elements of the triples but also the relationships both within a triple and between the triples. Experimental results show that the proposed encoder achieves a consistent improvement over the baseline models by up to 17.6%, 6.0%, and 16.4% in three common metrics BLEU, METEOR, and TER, respectively.

1 Introduction

Knowledge bases (KBs) are becoming an enabling resource for many applications including Q&A systems, recommender systems, and summarization tools. KBs are designed based on a W3C standard called the *Resource Description Framework* (RDF)[1]. An RDF triple consists of three elements in the form of $\langle subject, predicate\ (relationship), object\rangle$. It describes a relationship between an entity (the subject) and another entity or literal (the object) via the predicate. This representation allows easy data share between KBs. However, usually the elements of a triple are stored as *Uniform Resource Identifiers* (URIs), and many predicates (words or phrases) are not intuitive; this representation is difficult to comprehend by humans.

RDF triples	⟨John Doe,birth place,London⟩ ⟨John Doe,birth date,1967-01-10⟩ ⟨London,capital of,England⟩
Target sentence	John Doe was born on 1967-01-10 in London, the capital of England.

Table 1: RDF based sentence generation.

Translating RDF triples into natural sentences helps humans to comprehend the knowledge embedded in the triples, and building a natural language based user interface is an important task in user interaction studies (Damljanovic et al., 2010). This task has many applications, such as question answering (Bordes et al., 2014; Fader et al., 2014), profile summarizing (Lebret et al., 2016; Chisholm et al., 2017), and automatic weather forecasting (Mei et al., 2016). For example, the SPARQL inference of a Q&A system (Unger et al., 2012) returns a set of RDF triples which need to be translated into natural sentences to provide a more readable answer for the users. Table 1 illustrates such an example. Suppose a user is asking a question about "John Doe". By querying a KB, a Q&A system retrieves three triples "⟨John Doe,birth place,London⟩", "⟨John Doe,birth date,1967-01-10⟩", and "⟨London,capital of,England⟩." We aim to generate a natural sentence that incorporates the information of the triples and is easier to be understood by the user. In this example, the generated sentence is "John Doe was born on 1967-01-10 in London,

*Corresponding author
[1]https://www.w3.org/RDF/

the capital of England."

Most existing studies for this task use domain specific rules. Bontcheva and Wilks (2004) create rules to generate sentences in the medical domain, while Cimiano et al. (2013) create rules to generate step by step cooking instructions. The problem of rule-based methods is that they need a lot of human efforts to create the rules, which mostly cannot deal with complex or novel cases.

Recent studies propose neural language generation systems. Lebret et al. (2016) generate the first sentence of a biography by a conditional neural language model. Mei et al. (2016) propose an encoder-aligner-decoder architecture to generate weather forecasts. The model does not need predefined rules and hence generalizes better to open domain data.

A straightforward adaptation of neural language generation system is to use the encoder-decoder model by first concatenating the elements of the RDF triples into a linear sequence and then feeding the sequence as the model input to learn the corresponding natural sentence. We implemented such a model (detailed in Section 3.2) that ranked top in the WebNLG Challenge 2017[2]. This Challenge has a primary objective of generating syntactically correct natural sentences from a set of RDF triples. Our model achieves the highest global scores on the automatic evaluation, outperforming competitors that use rule-based methods, statistical machine translation, and neural machine translation (Gardent et al., 2017b).

While our previous model achieves a good result, simply concatenating the elements in the RDF triples may lose the relationship between entities that affects the semantics of the resulting sentence (cf. Table 3). To address this issue, in this paper, we propose a novel graph-based triple encoder model that maintain the structure of RDF triples as a small knowledge graph named the *GTR-LSTM* model. This model computes the hidden state of each entity in a graph to preserve the relationships between entities in a triple (*intra-triple relationships*) and the relationships between entities in related triples (*inter-triple relationships*) that helps to achieve even more accurate sentences. This leads to two problems of preserving the relationships in a knowledge graph: (1) how to deal with a cycle in a knowledge graph; (2) how to deal with multiple non-predefined relationships between two entities in a knowledge graph. The proposed model differs from existing non-linear LSTM models such as Tree LSTM (Tai et al., 2015) and Graph LSTM (Liang et al., 2016) in addressing the mentioned problem. In particular, Tree LSTM does not allow cycles, while the proposed model handles cycles by first using a combination of topological sort and breadth-first traversal over a graph, and then using an attention model to capture the global information of the knowledge graph. Meanwhile, Graph LSTM only allows a predefined set of relationships between entities, while the proposed model allows any relationships by treating them as part of the input for the hidden state computation.

To further enhance the capability of our model to handle unseen entities, we propose to use entity masking, which maps the entities in the model training pairs to their types, e.g., we map an entity (literal) "1967-01-10" to a type symbol "DATE" in the training pairs. This way, our model can learn to handle any date entities rather than just "1967-01-10". This is particularly helpful when there is a limited training dataset.

Our contributions are:

- We propose an end-to-end encoder-decoder based framework for the problem of translating RDF triples into natural sentences.
- We further propose a graph-based triple encoder to optimize the amount of information preserved in the input of the framework. The proposed model can handle cycles to capture the global information of a knowledge graph. The proposed model also handles non-predefined relationships between entities.
- We evaluate the proposed framework and model over two real datasets. The results show that our model outperforms the state-of-the-art models consistently.

The rest of this paper is organized as follows. Section 2 summarizes previous studies on sentence generation. Section 3 details the proposed model. Section 4 presents the experimental results. Section 5 concludes the paper.

2 Related Work

The studied problem falls in the area of *Natural Language Generation* (NLG) (Reiter and Dale, 2000). Bontcheva and Wilks (2004) follow a

[2]http://talc1.loria.fr/webnlg/stories/challenge.html

traditional NLG approach to generate sentences from RDF data in the medical domain. They start with filtering repetitive RDF data (document planning) and then group coherent triples (microplanning). After that, they aggregate the sentences generated for coherent triples to produce the final sentences (aggregation and realization). Cimiano et al. (2013) generate cooking recipes from semantic web data. They focus on using a large corpus to extract lexicon in the cooking domain. The lexicon is then used with a traditional NLG approach to generate cooking recipes. Duma and Klein (2013) learn a sentence template from a parallel RDF data and text corpora. They first align entities in RDF triples with entities mentioned in sentences. Then, they extract templates from the aligned sentences by replacing the entity mention with a unique token. This method works well on RDF triples in a seen domain but fails on RDF triples in a previously unseen domain.

Recently, several methods using neural networks are proposed. Lebret et al. (2016) generate the first sentence of a biography using a conditional neural language model. This model is trained to predict the next word of a sentence not only based on previous words, but also by using features captured from Wikipedia infoboxes. Mei et al. (2016) propose an encoder-aligner-decoder model to generate weather forecasts. The aligner is used to filter the most relevant data to be used to predict the weather forecast. Both studies experiment on cross-domain datasets. The result shows that the neural language generation approach is more flexible to work in an open domain since it is not limited to handcrafted rules. This motivates us to use a neural network based framework.

The most similar system to ours is Neural Wikipedian (Vougiouklis et al., 2017), which generates a summary from RDF triples. It uses feed-forward neural networks to encode RDF triples and concatenate them as the input of the decoder. The decoder uses LSTM to predict a sequence of words as a summary. There are differences from our work. First, Neural Wikipedian only works with a set of RDF triples with a single entity point of view (i.e., the entity of interest must be in either the subject or the object of every triple). Our system does not have this constraint. Second, Neural Wikipedian uses standard feed-forward neural networks in the encoder. We design new triple encoder models to accommodate specific features of RDF triples. Experimental results show that our framework outperforms Neural Wikipedian.

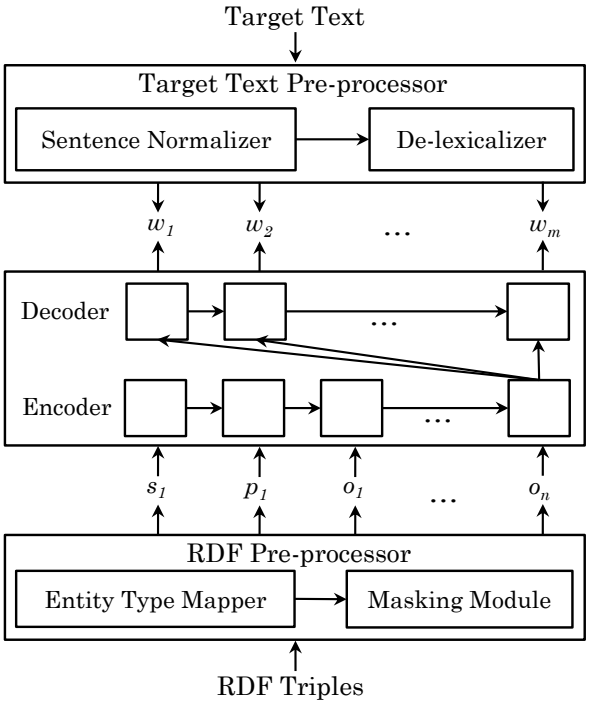

Figure 1: RDF sentence generation based on an encoder-decoder architecture.

3 Proposed Model

We start with the problem definition. We consider a set of RDF triples as the input, which is denoted by $T = [t_1, t_2, ..., t_n]$ where a triple t_i consists of three elements (subject s_i, predicate p_i, and object o_i), $t_i = \langle s_i, p_i, o_i \rangle$. Every element can contain multiple words. We aim to generate a set of sentences that consist of a sequence of words $S = \langle w_1, w_2, ..., w_m \rangle$, such that the relationships in the input triples are correctly represented in S while the sentences have a high quality. We use BLEU, METEOR, and TER to assess the quality of the sentence (detailed in Section 4). Table 1 illustrates our problem input and the target output.

This section is organized as follows. First we describe the overall framework (Section 3.1). Next, we describe three triple encoder models including the *adapted standard BLSTM* model (Section 3.2), the *adapted standard triple encoder* model (Section 3.3), and the *proposed GTR-LSTM* model (Section 3.4). The decoder which is used for all encoder models is described in Section 3.5. The entity masking is described in Section 3.6

3.1 Solution Framework

Our solution framework uses an encoder-decoder architecture as illustrated in Fig. 1. The framework

consists of three components including an *RDF pre-processor*, a *target text pre-processor*, and an *encoder-decoder module*.

The RDF pre-processor consists of an entity type mapper and a masking module. The entity type mapper maps the subjects and objects in the triples to their types, such that the sentence patterns learned are based on entity types rather than entities. For example, the input entities in Table 1, "`John Doe`", "`London`", "`England`", and "`1967-01-10`" can be mapped to "`PERSON`", "`CITY`", "`COUNTRY`", and "`DATE`", respectively. The mapping has been shown in our experiments to be highly effective in improving the model output quality. The masking module converts each entity into an *entity identifier* (eid). The target text pre-processor consists of a text normalizer and a de-lexicalizer. The text normalizer converts abbreviations and dates into the same format as the corresponding entities in the triples. The de-lexicalizer replaces all entities in the target sentences by their eids. The RDF and target text pre-processors are detailed in Section 3.6. The replaced target sentences are combined with the original target sentences and the English Wikipedia articles is used as a corpus to learn the word embeddings of the vocabulary.

To accommodate the RDF data, in the encoder side, we consider three triple encoder models: (1) the adapted standard BLSTM encoder; (2) the adapted standard triple encoder; and (3) the proposed GTR-LSTM triple encoder. The adapted standard BLSTM encoder concatenates the tokens in RDF triples as an input sequence, while the standard triple encoder first encodes each RDF triple into a vector representation and then concatenates the vectors of different triples. The latter model better captures intra-triple relationships but suffers in capturing inter-triple relationships. Considering the native representation of RDF triples as a small knowledge graph, our graph-based GTR-LSTM triple encoder captures both intra-triple and inter-triple entity relationships.

3.2 Adapted Standard BLSTM Encoder

The standard encoder-decoder model with a BLSTM encoder is a sequence to sequence learning model (Cho et al., 2014). To adapt such a model for our problem, we transform a set of RDF triples input T into a sequence of elements (i.e., $T = [w_{1,1}, w_{1,2}, ..., w_{1,j}, ..., w_{n,j}]$), where $w_{n,j}$ is

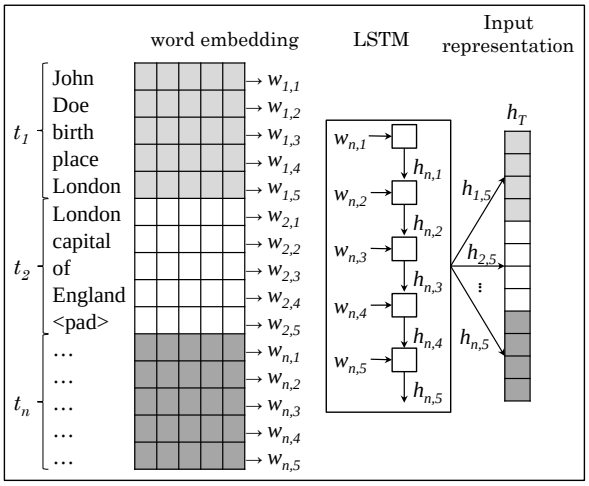

Figure 2: LSTM-based standard triple encoder.

the word embedding of a word in the n-th triple. For example, following the triples in Table 1, $w_{1,1}$ is the word embedding of "John", $w_{1,2}$ is the word embedding of "Doe", etc. This sequence forms an input for the encoder. We use zero padding to ensure that each input has the same representation size. The rest of the model is the same as the standard encoder-decoder model with an attention mechanism (Bahdanau et al., 2015). We call this model the *adapted standard BLSTM encoder*.

3.3 Adapted Standard Triple Encoder

The standard BLSTM encoder suffers in capturing the element relationships as the elements are simply concatenated together. Next, we adapt the standard BLSTM encoder to aggregate the word embeddings of the elements of the same triple to retain the intra-triple relationship. We call this the *adapted standard triple encoder*.

The adaptation is done by grouping the elements of each triple, so the input is represented as $T = [\langle w_{1,1}, ..., w_{1,j}\rangle, ..., \langle w_{n,1}, ...w_{n,j}\rangle]$, where $w_{n,j}$ is the word embedding of a word in the n-th triple. We use zero padding to ensure that each triple has the same representation size. An LSTM network of the encoder computes a hidden state of each triple and concatenates them together to be the input for the decoder:

$$h_T = [f(t_1); f(t_2); ...; f(t_n)] \quad (1)$$

where h_T is the input vector representation for the decoder and f is an LSTM network (cf. Fig. 2).

3.4 GTR-LSTM Triple Encoder

The adapted standard triple encoder has an advantage in preserving the intra-triple relationship. However, it has not considered the structural rela-

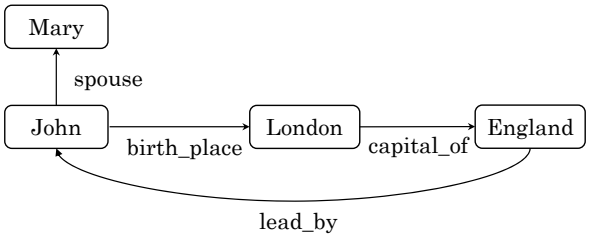

Figure 3: A small knowledge graph formed by a set of RDF triples.

tionships between the entities in different triples. To overcome this limitation, we propose a graph-based triple encoder. We call it the *GTR-LSTM triple encoder*. This encoder takes the input triples in the form of a graph, which preserves the natural structure of the triples (cf. Fig. 3).

GTR-LSTM differs from existing Graph LSTM (Liang et al., 2016) and Tree LSTM (Tai et al., 2015) models in the following aspects. Graph LSTM is proposed for image data. It constructs the graph based on the spatial relationships among super-pixels of an image. Tree LSTM uses the dependency tree as the structure of a sentence. Both models have a predefined relationship between the vertices (Graph LSTM uses spatial relationships: top, bottom, left, or right between super-pixels; Tree LSTM uses dependencies between words in a sentence as the relationship). In contrast, a KB has an open set of relationships between the vertices (i.e., the predicate defines the relationship between entities/vertices) which make our problem more difficult to model.

Our GTR-LSTM triple encoder overcomes the difficulty as follows. It receives a directed graph $G = \langle V, E \rangle$ as the input, where V is a set of vertices that represent entities or literals, and E is a set of directed edges that represent predicates. Since the graph can contain cycles, we use a combination of *topological sort* and *breadth-first traversal* algorithms to traverse the graph. The traversal is used to create an ordering of feeding the vertices into a GTR-LSTM unit to compute their hidden states. We start with running a topological sort to establish an order of the vertices until no further vertex has a zero in-degree. For the remaining vertices, they must be in strongly connected component(s). Then, we run a breadth-first traversal over the remaining vertices with a random starting vertex, since every vertex can be reached from all vertices of a strongly connected component. When a vertex v_i is visited, the hidden states of all adjacent vertices of v_i are computed (or updated if the hidden state of the vertex

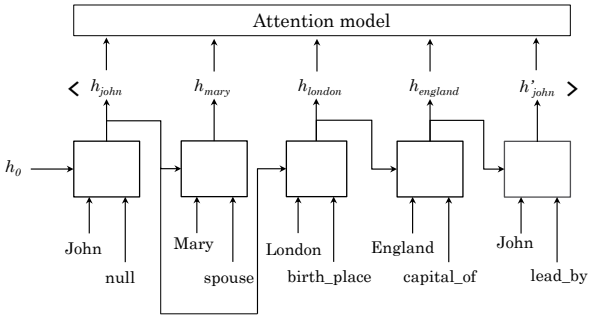

Figure 4: GTR-LSTM triple encoder.

is already computed in the previous step).

Following the graph in Fig. 3, the order of hidden state computation is as follows. The process starts with a vertex with zero in-degree. Because there is no such vertex, a vertex is randomly selected as the starting vertex. Assume we pick "`John`" as the starting vertex, then we compute h_{john} using h_0 as the previous hidden state. Next, following the *breadth-first traversal* algorithm, we visit vertex "`John`" and compute h_{mary} and h_{london} by passing h_{john} as the previous hidden state. Next step, vertex "`Mary`" is visited, but no hidden states are computed or updated since it does not have any adjacent vertices. In the last step, vertex "`England`" is visited and h_{john} is updated. Fig. 4 illustrates the overall process.

Different from the Graph LSTM, our GTR-LSTM model computes a hidden state by taking into account the processed entity and its edge (the edge pointing to the current entity from the previous entity) to handle non-predefined relationships (any relationships between entities in a knowledge graph). Thus, our GTR-LSTM unit (cf. Fig. 4) receives two inputs, i.e., the entity and its relationship. We propose the following model to compute the hidden state of each GTR-LSTM unit.

$$i_t = \sigma\left(\sum_e \left(U^{ie} x_{te} + W^{ie} h_{t-1}\right)\right) \quad (2)$$

$$f_{te} = \sigma\left(U^f x_{te} + W^f h_{t-1}\right) \quad (3)$$

$$o_t = \sigma\left(\sum_e \left(U^{oe} x_{te} + W^{oe} h_{t-1}\right)\right) \quad (4)$$

$$g_t = \tanh\left(\sum_e \left(U^{ge} x_{te} + W^{ge} h_{t-1}\right)\right) \quad (5)$$

$$c_t = \left(c_{t-1} * \sum_e f_{te}\right) + (g_t * i_t) \quad (6)$$

$$h_t = \tanh(c_t) * o_t \quad (7)$$

Here, U and W are learned parameter matrices, σ denotes the sigmoid function, $*$ denotes element-

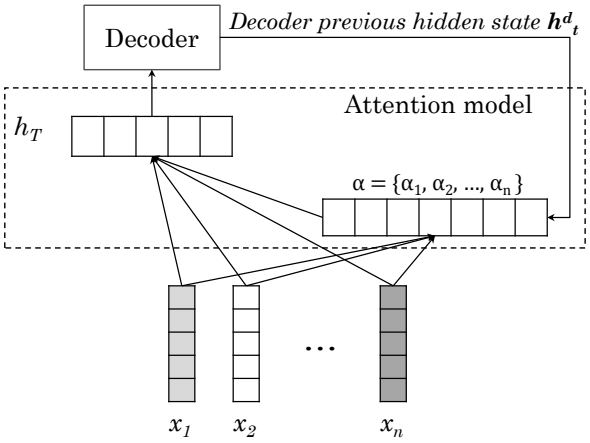

Figure 5: Attention model of GTR-LSTM.

wise multiplication, and x is the input at the current time-step. The input gate i determines the weight of the current input. The forget gate f determines the weight of the previous state. The output gate o determines the weight of the cell state forwarded to the next time-step. The state g is the candidate hidden state used to compute the internal memory unit c based on the current input and the previous state. The subscript t is the time-step. The subscript/superscript e is the input element (an entity or a predicate). Following Tree LSTM (Tai et al., 2015) and Graph LSTM (Liang et al., 2016), we also use a separate forget gate for each input that allows the GTR-LSTM unit to incorporate information from each input selectively.

From Fig. 4, we can see that the traversal creates two branches, one ended in h_{mary} and the other ended in h'_{john}. After the encoder computes the hidden states of each vertex, h'_{john} does not include the information of h_{mary} and vice versa. Moreover, the graph can contain cycles that cause difficulty in determining the starting and ending vertices. Our traversal procedure ensures that the hidden states of all vertices are updated based on their adjacent vertices (local neighbors). To further capture the global information of the graph, we apply an attention model on the GTR-LSTM triple encoder. The attention model takes the hidden states of all vertices computed by the encoder and the previous hidden state of the decoder to compute the final input vector of each decoder time-step. Figure 5 illustrates the attention model of GTR-LSTM. Inspired by Luong et al. (2015), we adapt the following equation to compute the weight of each vertex.

$$\alpha_n = \frac{\exp({h^d_t}^T W x_n)}{\sum_{j=1}^{|X|} \exp({h^d_t}^T W x_j)} \quad (8)$$

Here, ${h^d}_t$ is the previous hidden state of the decoder, $|X|$ is the total number of entities in the triples, W is a learned parameter matrix, x_n and x_j are hidden states of vertices, and $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ is the weight vector of all vertices. Then the input of the decoder for each time-step can be computed as follows.

$$h_T = \sum_{n=1}^{|X|} \alpha_n x_n \quad (9)$$

3.5 Decoder

The decoder of the proposed framework is a standard LSTM. It is trained to generate the output sequence by predicting the next output word w_t conditioned on the hidden state ${h^d}_t$. The current hidden state ${h^d}_t$ is conditioned on the hidden state of the previous time-step ${h^d}_{t-1}$, the output of the previous time-step w_{t-1}, and input vector representation h_T. The hidden state and the output of the decoder at time-step t are computed as:

$${h^d}_t = f({h^d}_{t-1}, w_{t-1}, h_T) \quad (10)$$

$$w_t = softmax(V h_t) \quad (11)$$

Here, f is a single LSTM unit, and V is the hidden-to-output weight matrix. The encoder and the decoder are trained to maximize the conditional log-likelihood:

$$p(S_n \mid T_n) = \sum_{t=1}^{|S_n|} \log w_t \quad (12)$$

Hence, the training objective is to minimize the negative conditional log-likelihood:

$$J = \sum_{n=1}^{N} -\log p(S_n \mid T_n) \quad (13)$$

where (S_n, T_n) is a pair of output word sequence and input RDF triple set given for the training.

3.6 Entity Masking

Entity masking makes our framework generalizes better to unseen entities. This technique addresses the problem of a limited training set which is faced by many NLG problems.

Entity masking replaces entity mentions with eids and entity types in both the input triples and the target sentences. However, we do not want our model to be overly generalized either. Thus, we need to have general and specific entity types. For example, the entity "`John Doe`" is replaced by "`ENT-1 PERSON GOVERNOR`". To add the entity types, we use the DBpedia lookup API. The

API returns several entity types. The general and specific entity types are defined by the level of the word in the WordNet (Fellbaum, 1998) hierarchy.

In the encoder side, each element of the triple $t_n = \langle s_n, p_n, o_n \rangle$ is transformed into $s_n = \langle l_{s_n}, g_{s_n}, d_{s_n} \rangle$, $p_n = \langle l_{p_n} \rangle$, and $o_n = \langle l_{o_n}, g_{o_n}, d_{o_n} \rangle$, where l is the label of an element, g is the general entity type, and d is the specific entity type. The labels of the subject and the object are latter replaced by eids, while the label of the predicate is preserved, since it indicates the relationship between the subject and the object.

On the decoder side, the entities in the target text are also replaced by their corresponding eids. Entity matching is beyond the scope of our study. We simply use a combination of three string matching methods to find entity mentions in the sentence: exact matching, *n-gram* matching, and parse tree matching. The exact matching is used to find the exact mention; the *n-gram* matching is used to handle partial matching with the same token length; and parse tree matching is used to find a partial matching with different token length.

4 Experiments

We evaluate our framework on two datasets. The first is the dataset from Gardent et al. (2017a). We call it the *WebNLG* dataset. This dataset contains 25,298 RDF triple set-text pairs, with 9,674 unique sets of RDF triples. The dataset consists of a Train+Dev dataset and a Test Unseen dataset. We split Train+Dev into a training set (80%), a development set (10%), and a Seen testing set (10%). The Train+Dev dataset contains RDF triples in ten categories (topics, e.g., astronaut, monument, food, etc.), while the Test Unseen dataset has five other unseen categories. The maximum number of triples in each RDF triple set is seven. For the second dataset, we collected data from Wikipedia pages regarding landmarks. We call it the *GKB* dataset. We first extract RDF triples from Wikipedia infoboxes and sentences from the Wikipedia text that contain entities mentioned in the RDF triples. Human annotators then filter out false matches to obtain 1,000 RDF triple set-text pairs. This dataset is split into the training and development set (80%) and the testing set (20%). Table 1 illustrates an example of the data pairs of WebNLG and GKB dataset.

We implement the existing models, the adapted model, and the proposed model using Keras[3]. We use three common evaluation metrics including BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2011), and TER (Snover et al., 2006). For the metric computation and significance testing, we use MultEval (Clark et al., 2011).

4.1 Tested Models

We compare our proposed graph-based triple encoder (GTR-LSTM, Section 3.4) with three existing model including the adapted standard BLSTM encoder (BLSTM, Section 3.2), Neural Wikipedian (Vougiouklis et al., 2017) (TFF), and statistical machine translation (Hoang and Koehn, 2008) (SMT) trained on a *6-gram* language model. We also compare with the adapted standard triple encoder (TLSTM, Section 3.3).

4.2 Hyperparameters

We use grid search to find the best hyperparameters for the neural networks. We use GloVe (Pennington et al., 2014) trained on the GKB and WebNLG training data and full English Wikipedia data dump to get 300-dimension word embeddings. We use 512 hidden units for both encoder and decoder. We use a 0.5 dropout rate for regularization on both encoder and decoder to avoid overfitting. We train our model on NVIDIA Tesla K40c. We find that using adaptive learning rates for the optimization is efficient and leads the model to converge faster. Thus, we use Adam (Kingma and Ba, 2015) with a learning rate of 0.0002 instead of stochastic gradient descent. The update of parameters in training is computed using a mini batch of 64 instances. We further apply early stopping to detect the convergence.

4.3 Effect of Entity Masking

Table 2 shows the overall comparison of model performance. It shows that entity masking gives a consistent performance improvement for all models. Generalizing the input triples and target sentences helps the models to learn the relationships between entities from their types. This is particularly helpful when there is limited training data. We use a combination of exact matching, *n-gram* matching and parse tree matching to find the entity mentions in the sentence. The entity masking accuracy for WebNLG dataset is 87.15%, while for

[3]https://nmt-keras.readthedocs.io/en/latest/

Model			BLEU↑			METEOR↑			TER↓		
		Metric/Dataset	Seen	Unseen	GKB	Seen	Unseen	GKB	Seen	Unseen	GKB
Entity Unmasking	Existing models	BLSTM	42.7	23.0	28.0	34.4	28.7	27.5	55.7	69.9	67.7
		SMT	41.1	23.9	27.7	33.2	28.3	27.6	57.0	70.1	63.8
		TFF	44.6	26.4	26.4	33.9	29.4	27.2	52.4	62.6	60.1
	Adapted model	TLSTM	45.9	28.1	29.4	34.9	30.1	28.5	50.5	62.7	59.0
	Our proposed	GTR-LSTM	54.0	29.2	37.1	37.3	27.8	30.6	45.3	59.8	55.1
Entity Masking	Existing models	BLSTM	49.8	28.0	34.8	38.3	29.4	28.6	49.9	64.9	65.8
		SMT	46.5	24.8	32.0	37.1	29.1	28.5	52.3	62.2	67.8
		TFF	47.8	28.4	33.7	35.9	30.5	28.9	49.9	61.2	58.4
	Adapted Model	TLSTM	50.5	31.6	36.7	36.5	30.7	30.1	47.7	60.4	57.2
	Our proposed	GTR-LSTM	**58.6**	**34.1**	**40.1**	**40.6**	**32.0**	**34.6**	**41.7**	**57.9**	**50.6**

Table 2: Comparison of model performance.

RDF inputs	⟨Elizabeth Tower, location, London⟩, ⟨Wembley Stadium, location, London⟩, ⟨London, capital of, England⟩, ⟨Theresa May, prime minister, England⟩
Reference	london , england is home to wembley stadium and the elizabeth tower. the name of the leader in england is theresa may.
BLSTM	england is lead by theresa may and **is located in the city of london .** the elizabeth tower is located **in the city of england** and is **located in the wembley stadium.**
SMT	wembley stadium is located in **london , elizabeth tower .** theresa may is the leader of **england , england.**
TFF	the elizabeth tower is located in london , england , where **wembley stadium is the leader** and theresa may is the leader.
TLSTM	the wembley stadium is located in london , england . the country is the location of elizabeth tower . **theresa may is the leader of london.**
GTR-LSTM	the wembley stadium and elizabeth tower are both located in london , england . theresa may is the leader of england.

Table 3: Sample output of the system. The error is highlighted in bold.

the GKB dataset is 82.45%.

Entity masking improves the BLEU score of the proposed GTR-LSTM model by 8.5% (from 54.0 on the Entity Unmasking model to 58.6 on the Entity Masking model), 16.7%, and 8.0% on the WebNLG seen testing data (denoted by "**Seen**"), WebNLG unseen testing data (denoted by "**Unseen**"), and the GKB testing data (denoted by "**GKB**"). Using the entity masking not only improves the performance by recognizing the unknown vocabulary via eid masking but also improves the running time performance by requiring a smaller training vocabulary.

4.4 Effect of Models

Table 2 also shows that the proposed GTR-LSTM triple encoder achieves a consistent improvement over the baseline models, and the improvement is statistically significant, with $p < 0.01$ based on the t-test of all metrics. We use MultEval to compute the p value based on an approximate randomization (Clark et al., 2011). The improvement on the BLEU score indicates that the model reduces the errors in the generated sentence. Our manual inspection confirms this result. The better (lower) TER score suggests that the model generates a more compact output (i.e., better aggregation).

Table 3 shows a sample output of all models. From this table, we can see that all baseline models produce sentences that contain wrong relationships between entities (e.g., the BLSTM output contains a wrong relationship "`the elizabeth tower is located in the city of england`"). Moreover, the baseline models generate sentences with a weak aggregation (e.g., "`Elizabeth Tower`" and "`Wembley Stadium`" are in separate sentences for TLSTM). The proposed GTR-LSTM model successfully avoids these problems.

Model training time. GTR-LSM is slower in training than the baseline models, which is expected as it needs to encode more information. However, its training time is no more than twice as that of any baseline models tested, and the training can complete within one day which seems reasonable. Meanwhile, the number of parameters trained for GTR-LSTM is up to 59% smaller than those of the baseline models, which saves the space cost for model storage.

4.5 Human Evaluation

To complement the automatic evaluation, we conduct human evaluations for all of the masked models. We ask five human annotators. Each of them

Model		Seen			Unseen			GKB		
	Dataset/Metric	Correctness	Grammar	Fluency	Correctness	Grammar	Fluency	Correctness	Grammar	Fluency
Existing Models	BLSTM	2.25	2.33	2.29	1.53	1.71	1.68	1.54	1.84	1.84
	SMT	2.03	2.11	2.07	1.36	1.48	1.44	1.81	1.99	1.89
	TFF	1.77	1.91	1.88	1.44	1.69	1.66	1.71	1.99	1.96
Adapted Model	TLSTM	2.53	2.61	2.55	1.75	1.93	1.86	2.21	2.38	2.35
Our Proposed	GTR-LSTM	**2.64**	**2.66**	**2.57**	**1.96**	**2.04**	**1.99**	**2.29**	**2.42**	**2.41**

Table 4: Human evaluation results.

has studied English for at least ten years and completed education in a full English environment for at least two years. We provide a website[4] that shows them the RDF triples and the generated text. The annotators are given training on the scoring criteria. We also provide scoring examples. We randomly selected 100 sets of triples along with the output of each model. We only select sets of triples that contain more than two triples. Following (Gardent et al., 2017b), we use three evaluation metrics including *correctness*, *grammaticality*, and *fluency*. For each pair of triple set and generated sentences, the annotators are asked to give a score between one to three for each metric.

Correctness is used to measure the semantics of the output sentence. A score of 3 is given to generated sentences that contain no errors in the relationships between entities; a score of 2 is given to generated sentences that contain one error in the relationship; and a score of 1 is given to generated sentences that contain more than one errors in the relationships. *Grammaticality* is used to rate the grammatical and spelling errors of the generated sentences. Similar to the *correctness* metric, a score of 3 is given to generated sentences with no grammatical and spelling errors; a score of 2 is given to generated sentences with one error; and a score of 1 for the others. The last metric, *fluency*, is used to measure the fluency of the sentence output. We ask the annotators to give a score based on the aggregation of the sentences and the existence of sentence repetition. Table 4 shows the results of the human evaluations. The results confirm the automatic evaluation in which our proposed model achieves the best scores.

Error analysis. We further perform a manual inspection of 100 randomly selected output sentences of GTR-LSTM and BLSTM on the Seen and Unseen test data. We find that 32% of BLSTM output contains wrong relationships between entities. In comparison, only 8% of GTR-LSTM output contains such errors. Besides, we find duplicate sub-sentences in the output of GTR-LSTM (15%). The following output is an example: "`beef kway teow is a dish from singapore, where english language is spoken and the leader is tony tan. the leader of singapore is tony tan.`" While the duplicate sentence is not wrong, it affects the reading experience. We conjecture that the LSTM in the decoder caused such an issue. We aim to solve this problem in future work.

[4]http://bit.ly/gkb-mappings

5 Conclusions

We proposed a novel graph-based triple encoder GTR-LSTM for sentence generation from RDF data. The proposed model maintains the structure of input RDF triples as a small knowledge graph to optimize the amount of information preserved in the input of the model. The proposed model can handle cycles to capture the global information of a knowledge graph and also handle non-predefined relationships between entities of a knowledge graph.

Our experiments show that GTR-LSTM offers a better performance than all the competitors. On the WebNLG dataset, our model outperforms the best existing model, the standard BLSTM model, by up to 17.6%, 6.0%, and 16.4% in terms of BLEU, METEOR, and TER scores, respectively. On the GKB dataset, our model outperforms the standard BLSTM model by up to 15.2%, 20.9%, and 23.1% in these three metrics, respectively.

Acknowledgments

Bayu Distiawan Trisedya is supported by the Indonesian Endowment Fund for Education (LPDP). This work is supported by Australian Research Council (ARC) Discovery Project DP180102050 and Future Fellowships Project FT120100832, and Google Faculty Research Award. This work is partly done while Jianzhong Qi is visiting the University of New South Wales. Wei Wang was partially supported by D2DCRC DC25002, DC25003, ARC DP 170103710 and 180103411.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*. http://arxiv.org/abs/1409.0473.

Kalina Bontcheva and Yorick Wilks. 2004. *Automatic Report Generation from Ontologies: The MIAKT Approach*, Springer, Berlin, Heidelberg, pages 324–335. https://doi.org/10.1007/978-3-540-27779-8_28.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 615–620. https://www.aclweb.org/anthology/D/D14/D14-1067.pdf.

Andrew Chisholm, Will Radford, and Ben Hachey. 2017. Learning to generate one-sentence biographies from wikidata. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. pages 633–642. http://aclweb.org/anthology/E17-1060.

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. http://www.aclweb.org/anthology/D14-1179.

Philipp Cimiano, Janna Lüker, David Nagel, and Christina Unger. 2013. Exploiting ontology lexica for generating natural language texts from rdf data. In *Proceedings of the 14th European Workshop on Natural Language Generation (ENLG)*. pages 10–19. http://www.aclweb.org/anthology/W13-2102.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*. pages 176–181. http://www.aclweb.org/anthology/P11-2031.

Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. 2010. Natural language interfaces to ontologies: combining syntactic analysis and ontology-based lookup through the user interaction. In *Proceedings of the 7th International Conference on The Semantic Web (ISWC)*. pages 106–120. https://doi.org/10.1007/978-3-642-13486-9_8.

Michael J. Denkowski and Alon Lavie. 2011. Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems. In *Proceedings of the Sixth Workshop on Statistical Machine Translation (WMT)*. pages 85–91. http://aclweb.org/anthology/W11-2107.

Daniel Duma and Ewan Klein. 2013. Generating natural language from linked data: Unsupervised template extraction. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS)*. pages 83–94. http://www.aclweb.org/anthology/W13-0108.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. pages 1156–1165. https://doi.org/10.1145/2623330.2623677.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. MIT Press. http://aclweb.org/anthology/J99-2008.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017a. Creating training corpora for nlg micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*. pages 179–188. http://aclweb.org/anthology/P17-1017.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017b. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th International Conference on Natural Language Generation (INLG)*. pages 124–133. http://www.aclweb.org/anthology/W17-3518.

Hieu Hoang and Philipp Koehn. 2008. Design of the moses decoder for statistical machine translation. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP)*. pages 58–65. http://www.aclweb.org/anthology/W08-0510.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1412.6980.

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 (EMNLP)*. pages 1203–1213. https://aclweb.org/anthology/D16-1128.

Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. 2016. Semantic object parsing with graph lstm. In *Proceedings of the 14th European Conference on Computer Vision (ECCV)*. pages 125–143. https://doi.org/10.1007/978-3-319-46448-0_8.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 1412–1421. http://aclweb.org/anthology/D15-1166.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. pages 720–730. http://www.aclweb.org/anthology/N16-1086.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. pages 311–318. http://aclweb.org/anthology/P02-1040.pdf.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. https://aclweb.org/anthology/D14-1162.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge University Press.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas (AMTA)*. pages 223–231. http://mt-archive.info/AMTA-2006-Snover.pdf.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL) and the 7th International Joint Conference on Natural Language Processing (IJCNLP)*. pages 1556–1566. http://www.aclweb.org/anthology/P15-1150.

Christina Unger, Lorenz Bhmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web (WWW)*. pages 639–648. https://doi.org/10.1145/2187836.2187923.

Pavlos Vougiouklis, Hady Elsahar, Lucie-Aime Kaffee, Christoph Gravier, Frederique Laforest, Jonathon Hare, and Elena Simperl. 2017. Neural wikipedian: Generating textual summaries from knowledge base triples. *arXiv preprint arXiv:1711.00155* https://arxiv.org/pdf/1711.00155.pdf.