Sentence Generation for Entity Description with Content-plan Attention

Bayu Distiawan Trisedya and **Jianzhong Qi** and **Rui Zhang**[*]
School of Computing and Information Systems, The University of Melbourne
{btrisedya@student., jianzhong.qi@, rui.zhang@}unimelb.edu.au

Abstract

We study neural data-to-text generation. Specifically, we consider a target entity that is associated with a set of attributes. We aim to generate a sentence to describe the target entity. Previous studies use encoder-decoder frameworks where the encoder treats the input as a linear sequence and uses LSTM to encode the sequence. However, linearizing a set of attributes may not yield the proper order of the attributes, and hence leads the encoder to produce an improper context to generate a description. To handle disordered input, recent studies propose two-stage neural models that use pointer networks to generate a content-plan (i.e., content-planner) and use the content-plan as input for an encoder-decoder model (i.e., text generator). However, in two-stage models, the content-planner may yield an incomplete content-plan, due to missing one or more salient attributes in the generated content-plan. This will in turn cause the text generator to generate an incomplete description. To address these problems, we propose a novel attention model that exploits content-plan to highlight salient attributes in a proper order. The challenge of integrating a content-plan in the attention model of an encoder-decoder framework is to align the content-plan and the generated description. We handle this problem by devising a coverage mechanism to track the extent to which the content-plan is exposed in the previous decoding time-step, and hence it helps our proposed attention model select the attributes to be mentioned in the description in a proper order. Experimental results show that our model outperforms state-of-the-art baselines by up to 3% and 5% in terms of BLEU score on two real-world datasets, respectively.

1 Introduction

Generating natural language description of an entity from structured data is important for various applications such as question answering (Bordes, Chopra, and Weston 2014) and profile summarizing (Lebret, Grangier, and Auli 2016). In this paper, we study how to generate an entity description from its attributes. Specifically, we aim to generate a description from a set of attributes of a target entity A; the attributes are in the form of pairs of key and value, i.e., $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, ...\langle k_n; v_n\rangle\}$, where k_n is the key of the attribute and v_n is the value of the attribute. Table 1 illustrates the input and output of the task; in this example, the attributes are `name`, `birth_place`, etc and their values are `"Keanu Reeves"`, `"Beirut, Lebanon"`, etc. Here, the attributes may have been extracted from a table, which makes the task *table-to-text generation*, or a knowledge graph (KG), which makes the task *rdf-to-text generation*. In table-to-text generation, the attributes are extracted from a two-column table (e.g., Wikipedia infobox) where the first column indicates the key and the second column indicates the value of the attributes. In rdf-to-text generation, the attributes are extracted by querying a KG for RDF triples (i.e., ⟨`subject`, `predicate`, `object`⟩) that contain the target entity as the subject. In both cases, the input will form a star-shaped graph with the target entity as the center of the star-shaped graph, the attribute values as the points of the star, and the attribute keys as the edges (cf. Fig. 1).

Input	⟨`name; Keanu Reeves`⟩ ⟨`birth_place; Beirut, Lebanon`⟩ ⟨`occupation; actor`⟩ ⟨`occupation; musician`⟩ ⟨`birth_date; September 2, 1964`⟩ ⟨`residence; California, U.S.`⟩ ⟨`birth_name; Keanu Charles Reeves`⟩ ⟨`citizenship; Canada`⟩ ⟨`citizenship; United States`⟩
Output	`Keanu Charles Reeves (born September 2, 1964 in Beirut, Lebanon) is a American actor who lives in California, U.S.`

Table 1: Data-to-text generation.

Recent studies proposed end-to-end models by adapting the encoder-decoder framework. The encoder-decoder framework is a sequence-to-sequence model that has been successfully used in many tasks including machine translation (Cho et al. 2014), data-to-text generation (Liu et al. 2018), and relation extraction (Trisedya et al. 2019). The adaption of the sequence-to-sequence model for data-to-text generation includes representing the input as a sequence, and hence the order of attributes is important to guide the decoder to generate a good description (Vinyals, Bengio,

[*]Rui Zhang is the corresponding author

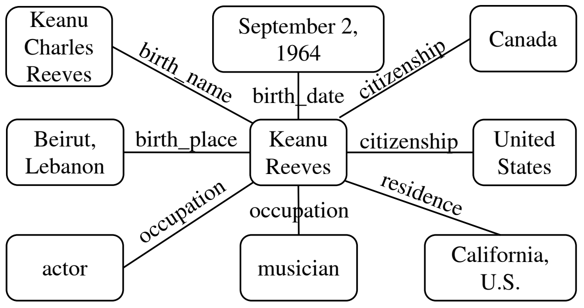


Figure 1: Star-shaped graph

and Kudlur 2016). Here, our definition of a proper order of attributes is the reasonable order of attributes in a well-organized sentence (i.e., a *content-plan*). For example, ⟨`birth_name`, `birth_date`, `birth_place`, `occupation`, `residence`⟩ is a content-plan for the output sentence in Table 1.

Previous studies (Bao et al. 2018; Liu et al. 2018; 2019; Sha et al. 2018) do not explicitly handle the order of input (i.e., the attributes). In fact, most data sources do not provide sets of attributes with a proper order. For example, the extracted attributes as a result of a query from a KG are typically disordered. Meanwhile, the extracted attributes from a web table are practically ordered, i.e., the salient attributes are relatively ordered but may have noises (non-salient attributes) between them, which disrupt the encoding. Moreover, Liu et al. (2018) reported a decrease in the performance of their model when experimenting on disordered input. Trisedya et al. (2018) proposed a graph-based encoder to exploit the input structure for generating sentences from a knowledge base (Geographic Knowledge Base). However, they require a graph that has a reasonable order of nodes (i.e., attributes) when traversed using topological sort and breadth-first traversal, which usually are expensive to obtain and unavailable in practice (Gardent et al. 2017).

Puduppully, Dong, and Lapata (2019) proposed Neural Content Planning (NCP) which is a two-stage model that includes content-planning to handle disordered input. First, NCP uses pointer networks (i.e., content-planner) (Vinyals, Fortunato, and Jaitly 2015) to generate a content-plan. Then, the generated content-plan is used as the input of the encoder-decoder model (i.e., text generator) (Bahdanau, Cho, and Bengio 2015) to generate a description. However, this two-stage model suffers from error propagation between the content-planner and the text generator. The generated content-plan may contain errors (e.g., missing one or more attributes that should be mentioned in the description) that lead the text generator to produce an incomplete description.

In this paper, we address the issues above by proposing an end-to-end model that jointly learns the content-planner and the text generator by integrating the content-plan in the attention model of an encoder-decoder model. The challenge of the integration is to align the learned content-plan and the generated description. To address this problem, we propose the *content-plan-based bag of tokens* attention model by adapting the coverage mechanism (Tu et al. 2016) to track the order of attributes in a content-plan for computing the attention of the attributes. This mechanism helps the attention module of the encoder-decoder model captures the most salient attribute at each time-step of the description generation phase in a proper order. Unlike the existing data-to-text generation models which treat the input as a sequence of attributes, our model treats the input as a bag of tokens and uses pointer networks to learn a content plan to handle disordered attributes. Our model maintains the original data (i.e., the original set of attributes) as the input of the text generator while exploiting the learned content-plan to highlight the attributes to be mentioned and hence reducing the error propagation between the content-planner and the text generator. To collect training data for the content-planner, we use string matching for extracting the order of attributes that are mentioned in the description as the target content-plan.

Our contributions are summarized as follows:

- We propose an end-to-end model that employs joint learning of content-planning and text generation to handle disordered input for generating a description of an entity from its attributes. The model reduces error propagation between the content-planner and the text generator, which two-stage models are prone to.
- We propose a *content-plan-based bag of tokens* attention model to effectively capture salient attributes in a proper order based on a content-plan.
- We evaluate the proposed model over two real-world datasets. The experimental results show that our model consistently outperforms state-of-the-art baselines for data-to-text generation (Liu et al. 2018; Puduppully, Dong, and Lapata 2019; Trisedya et al. 2018).

2 Related Work

Traditional approaches for text generation (McKeown 1992) consist of three components: (1) a *content-planner* that selects the data to be expressed, (2) a *sentence planner* that decides the structure of sentences or paragraphs based on the content-plan, and (3) a *surface realizer* that generates the final output based on the sentence plan. Earlier studies on content-planning employed handcrafted rules (Duboue and McKeown 2003) or a machine learning model as a content classifier (Barzilay and Lapata 2005; Duboue and Mckeown 2002). For sentence planning and surface realization, earlier studies proposed template-based models (McKeown et al. 1997; van Deemter, Krahmer, and Theune 2005), machine learning models using various linguistic features (Reiter and Dale 2000; Lu and Ng 2011; Belz 2008; Angeli, Liang, and Klein 2010), ordering constrained models (Mellish et al. 1998; Duboue and Mckeown 2001), and tree-based models (Kim and Mooney 2010; Lu, Ng, and Lee 2009). Typically, these approaches use handcrafted rules or shallow statistical models which mostly cannot deal with unseen and complex cases.

Recent studies proposed end-to-end models based on the encoder-decoder framework (Bahdanau, Cho, and Bengio 2015) for data-to-text generation. Serban et al. (2016) applied the encoder-decoder to generate questions from facts in a KG. Wiseman, Shieber, and Rush (2017) employed the encoder-decoder to generate NBA game summaries. Mei, Bansal, and Walter (2016) proposed an aligner model that integrates the content selection mechanism into the encoder-decoder for generating weather forecast from a set of database records. Lebret, Grangier, and Auli (2016) proposed a conditional language model for biography summarization. The follow-up studies on biography summarization employed the encoder-decoder framework. Sha et al. (2018) proposed link-based attention model to capture the relationships between attributes. Liu et al. (2018) proposed field-gating LSTM and dual attention mechanism to encode the attributes and inter-attribute relevance, respectively. These end-to-end models produce fluent text on an ordered input but had decreased performance on disordered input.

To address the above problem, previous studies proposed graph-based encoder to exploit the input structure. Marcheggiani and Perez-Beltrachini (2018) applied the graph convolutional networks (Kipf and Welling 2017) as the encoder to capture the input structure. Trisedya et al. (2018) proposed a graph LSTM that captures the order of the input by using topological sort and breadth-first traversal over the input graph. However, these models fail to capture the relationship between entities of a star-shaped graph since there is no edges between nodes that shows the reasonable order of nodes (i.e., attributes). Recently, Puduppully, Dong, and Lapata (2019) proposed a two-stage neural text generation architecture. First, they employ pointer networks (Vinyals, Fortunato, and Jaitly 2015) to select salient attributes and learn its order as a content-plan, then use the content-plan to generate a summary using the encoder-decoder framework. However, two-stage models are prone to propagate errors.

3 Preliminary

We start with the problem definition. Let A be a set of attributes of an entity in the form of pairs of key and value in any order, i.e., $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, ...\langle k_n; v_n\rangle\}$, where k_n is the key of the attribute and v_n is the value of the attribute. We consider A as the input and aim to generate a sentence $S = \langle t_1, t_2, ..., t_l\rangle$ as the description of an entity, where t_l is a token at position l in the sentence. Table 1 illustrates the input and output of the task.

Most data-to-text generation models are built on top of an encoder-decoder framework (Wiseman, Shieber, and Rush 2017; Mei, Bansal, and Walter 2016; Sha et al. 2018; Liu et al. 2018). We first discuss the encoder-decoder framework (Bahdanau, Cho, and Bengio 2015) and its limitation when generating text from disordered input.

3.1 Encoder-Decoder Framework

The encoder-decoder framework is a sequence-to-sequence learning model that takes a variable-length input T and generates a variable-length output T' where the length of T and T' may differ. The encoder reads each token of the input sequentially and computes a hidden state of each token. The hidden state of the last token represents a summary of the input sequence in the form of a fixed-length vector representation (i.e., context vector $\boldsymbol{c}$). The decoder is trained to generate a sequence by predicting the next token given the previous hidden state of the decoder and the context vector $\boldsymbol{c}$. This framework has been successfully applied in machine translation (Cho et al. 2014) to translate a sequence of words from one language to another.

In data-to-text generation, encoder-decoder is used to generate text (e.g., entity description) from structured data (e.g., a set of attributes of an entity). Here, the encoder learns to encode the attributes into a fixed-length vector representation, which is used as a context vector by the decoder to generate a description. Different from machine translation, in data-to-text generation, the input (i.e., attributes) may be disordered, and linearizing the input may not yield the proper order of the attributes. Hence, reading the disordered input sequentially may produce an improper context vector.

Bahdanau, Cho, and Bengio (2015) propose an attention model that improves the performance of sequence-to-sequence models. The attention model allows the encoder to dynamically compute the context vector for each decoding time-step by computing the weighted sum of each hidden state of the encoder. The weight represents the importance score of each token of the attributes and helps the encoder computes a specific context vector for each decoding time-step. However, the computations of the context vector are based on the hidden states of the encoder, which may not be appropriate for disordered input because a hidden state represents a summary of the previous tokens that do not hold the proper order. Besides, if we use the embeddings of the input (i.e., embeddings of the attribute token) instead of the hidden states to compute the attention, the model may not capture the relationships between attributes.

Next, we detail our model to address these limitations.

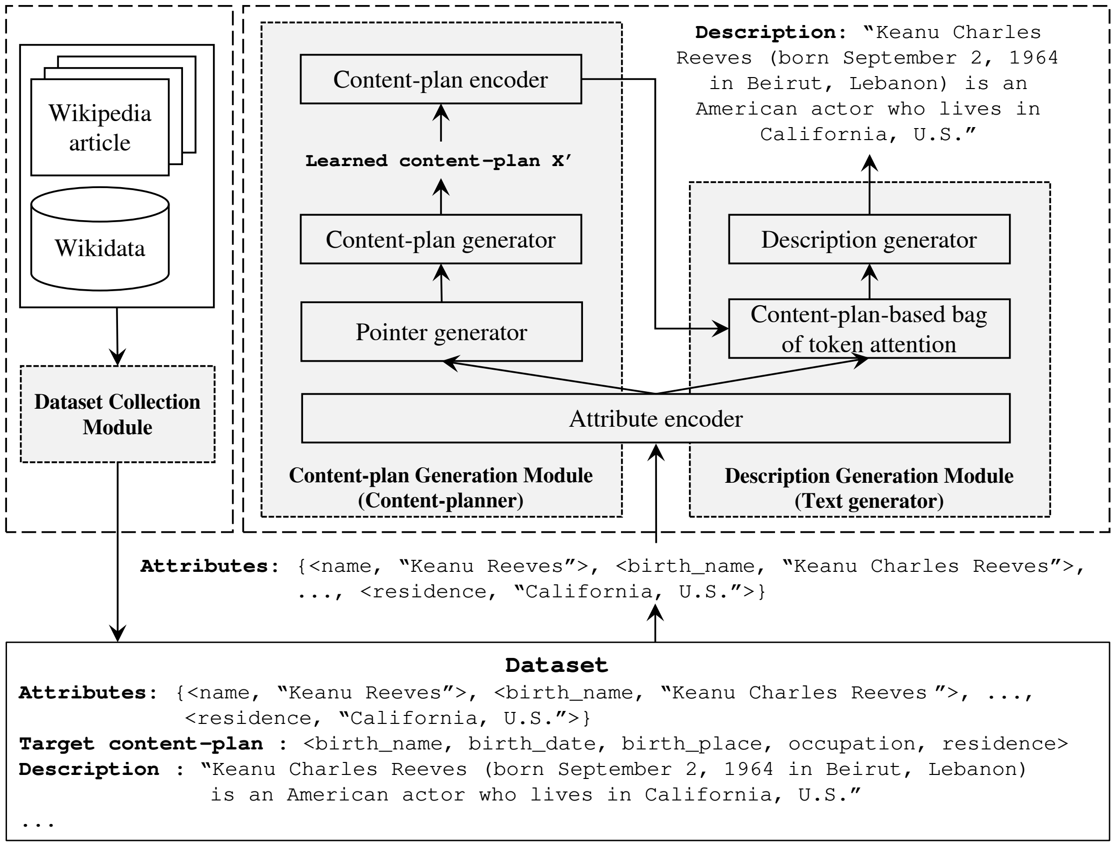
4 Proposed Model

4.1 Solution Framework

Figure 2 illustrates the overall solution framework. Our framework consists of three components: a *data collection module*, a *content-plan generation module*, and a *description generation module*.

In the data collection module (cf. Section 4.2), we collect a dataset in the form of triples of attributes, content-plan, and entity description. The attributes are extracted by querying Wikidata for RDF triples that contain the target entity as the subject. The description is obtained from Wikipedia by extracting the first sentence of the Wikipedia page of the target entity. The content-plan is extracted by finding the order of attributes in the description using string matching.

In the content-plan generation module (content-planner, cf. Section 4.3), we train pointer networks (Vinyals, Fortunato, and Jaitly 2015) to learn a content-plan that helps the attention model of the description generation module highlights the attributes in a proper order. This module consists of four components: (1) an *attribute encoder* that encodes a set of attributes into a vector by computing the average of the linear transformation of each token embeddings of the attributes; (2) a *pointer generator* that generates a sequence of indexes (pointers) that represents the order of attributes in the description; (3) a *content-plan generator* that gener-



Wikipedia article
Wikidata
Dataset Collection Module
Content-plan encoder
Learned content-plan X′
Content-plan generator
Pointer generator
Attribute encoder
Content-plan Generation Module (Content-planner)
Description: "Keanu Charles Reeves (born September 2, 1964 in Beirut, Lebanon) is an American actor who lives in California, U.S."
Description generator
Content-plan-based bag of token attention
Description Generation Module (Text generator)
Attributes: {<name, "Keanu Reeves">, <birth_name, "Keanu Charles Reeves">, ..., <residence, "California, U.S.">}

Dataset
Attributes: {<name, "Keanu Reeves">, <birth_name, "Keanu Charles Reeves ">, ..., <residence, "California, U.S.">}
Target content-plan : <birth_name, birth_date, birth_place, occupation, residence>
Description : "Keanu Charles Reeves (born September 2, 1964 in Beirut, Lebanon) is an American actor who lives in California, U.S."
...

Figure 2: Overview of our proposed solution

ates the content-plan based on the learned pointers; and (4) a *content-plan encoder* that encodes the learned content-plan to be used in the description generation module.

In the description generation module (text generator, cf. Section 4.4), we integrate the content-plan into the attention mechanism of the encoder-decoder model (Bahdanau, Cho, and Bengio 2015). We use the same encoder as in the content-plan generation module that treats the input (i.e., attributes) as a bag of tokens to ensure that the same set of attributes with different orders have the same representation. We do not use the recurrent model (e.g., LSTM (Hochreiter and Schmidhuber 1997), GRU (Cho et al. 2014)) to encode the attributes because they may compute improper context from disordered input (cf. Section 3.1). However, we use LSTM (i.e., content-plan encoder) to encode the learned content-plan that holds the proper order of attributes to capture the relationships between attributes. To integrate the learned content-plan into the attention mechanism of the encoder-decoder model, we propose the *content-plan-based bag of tokens* attention model by adapting the coverage mechanism (Tu et al. 2016) to track the order of attributes in a content-plan for computing the attention of the attributes. This way, our proposed attention model selects the salient attributes conditioned by the content-plan and hence provides a better context (i.e., attention of the attributes in an ordered fashion) for each decoding time-step.

4.2 Dataset Collection

We aim to generate a description of an entity from its attributes where the attributes may be disordered. To handle disordered input, we propose a model that performs joint learning of content-planning and text generation. To train such a model, we need labeled training data in the form of triples of attributes, content-plan, and entity description.

Following Lebret, Grangier, and Auli (2016), we extract the first sentence of a Wikipedia page of a target entity as the description. Different from Lebret, Grangier, and Auli (2016) who collected a specific type of entities (e.g., `Person`), we do not restrict the type of entities to be collected. We extract the attributes of a target entity by querying Wikidata for RDF triples that contain the target entity as the subject. In other words, we extract the direct relationships (i.e., attributes) of an entity which form a star-shaped graph. We querying the attributes from Wikidata instead of extracting from Wikipedia infobox to avoid additional processing (e.g., HTML tag removal, normalization, etc.).

We use string matching to find the order of attributes that are mentioned in the description as the content-plan. First, for each matched attribute, we store their position (index of the first character of the mentioned attribute value) in the description. Then, we sort the matched attributes based on their position ascendingly. For the example in Table 1, the extracted content plan is ⟨`birth_name, birth_date,` `birth_place, occupation, residence`⟩.

Our proposed model is trained to generate a description from a set of attributes of a target entity, which includes selecting salient attributes to be described. Since we automatically extract the description from Wikipedia, the extracted description may contain information (i.e., entities) that are not listed in the related extracted attributes, which creates noises in the dataset. This problem may be caused by the delayed synchronization of a KG (i.e., the Wikipedia page has been updated, but the Wikidata records have not been updated yet), which often occurs on frequently updated information such as the current club of a football player, the latest movie of an actor, etc. Hence, to obtain high-quality data, we filter descriptions that contain noises. First, we use a Named Entity Recognizer[1] to detect all entities in a description. Then, we remove descriptions that contain any entity that is not listed in the related extracted attributes.

The collected dataset contains $152,231$ triples of attributes, content-plan, and description (we call it the **WIKIALL** dataset). The dataset contains 53 entity types with an average of 15 attributes per entity, and an average of 20 tokens per description. For benchmarking, we also use the **WIKIBIO** dataset (Lebret, Grangier, and Auli 2016) which contains 728,321 biographies from Wikipedia. The average number of attributes per entity of WIKIBIO dataset is 19, and the average number of tokens per description is 26. We split each dataset into train set (80%), dev set (10%) and test set (10%).

4.3 Content-plan Generation

We adapt pointer networks (Vinyals, Fortunato, and Jaitly 2015) to learn a content-plan given a set of attributes (i.e., we use the pairs of attributes and content-plan from the dataset to train the networks). The pointer networks model uses an attention mechanism to generate a sequence of pointers that refer to the input so that it is suitable to rearrange the attributes as a content-plan. This module consists of four components, including the attribute encoder, the pointer generator, the content-plan generator, and the content-plan encoder.

Attribute Encoder. The attribute encoder takes a set of attributes $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, ...\langle k_n; v_n\rangle\}$ as the input. Here, the value of an attribute may consist of multiple tokens (i.e., $v_n = \langle v_n^1, v_n^2, ...v_n^j\rangle$). We transform the multiple tokens into a single token representation and add positional encoding to maintain its internal order. Thus, the attributes can be represented as $A = [\langle k_1^1, v_1^1, f_1^1, r_1^1\rangle, \langle k_1^2, v_1^2, f_1^2, r_1^2\rangle, ..., \langle k_n^j, v_n^j, f_n^j, r_n^j\rangle]$ where f_n^j and r_n^j are the forward and reverse positions respectively (cf. Table 2). We call the quadruple of key, value, forward position, and reverse position as *attribute-token*. The representation of each attribute-token $\boldsymbol{x_a}$ is computed as follows.

$$\boldsymbol{z_k}_n^j = \tanh(\boldsymbol{W_k}[\boldsymbol{k}_n^j; \boldsymbol{f}_n^j; \boldsymbol{r}_n^j] + \boldsymbol{b_k}) \tag{1}$$

$$\boldsymbol{z_v}_n^j = \tanh(\boldsymbol{W_v}[\boldsymbol{v}_n^j; \boldsymbol{f}_n^j; \boldsymbol{r}_n^j] + \boldsymbol{b_v}) \tag{2}$$

$$\boldsymbol{x_a}_n^j = \tanh(\boldsymbol{z_k}_n^j + \boldsymbol{z_v}_n^j) \tag{3}$$

where $[;]$ indicates vector concatenation, $\boldsymbol{b}$ indicates bias vector, and $\boldsymbol{W_k}$ and $\boldsymbol{W_v}$ are learned parameters. $\boldsymbol{z_k}$ and $\boldsymbol{z_v}$ are the vector representations of the attributes' key and value, respectively. To ensure that the same set of attributes with different orders have the same representation, we use the element-wise average of attribute-token vectors as the vector representation of a set of attributes. This vector is used as the initial hidden state of the decoder of the pointer networks (i.e., the pointer generator) and the encoder-decoder model since the attribute encoder is shared with the description generation module (cf. Section 4.4).

[1]https://spacy.io

Key	Value	Forward position	Reverse position
name (k_1^1)	Keanu (v_1^1)	1 (f_1^1)	2 (r_1^1)
name (k_1^2)	Reeves (v_1^2)	2 (f_1^2)	1 (r_1^2)
birth_name (k_2^1)	Keanu (v_2^1)	1 (f_2^1)	3 (r_2^1)
birth_name (k_2^2)	Charles (v_2^2)	2 (f_2^2)	2 (r_2^2)
birth_name (k_2^3)	Reeves (v_2^3)	3 (f_2^3)	1 (r_2^3)
...	...	...	...
residence (k_n^1)	California (v_n^1)	1 (f_n^1)	2 (r_n^1)
residence (k_n^2)	U.S. (v_v^2)	2 (f_n^2)	1 (r_n^2)

Table 2: Input representation.

Pointer Generator. Given a sequence of attribute-token vectors that are computed by the attribute encoder $X = \langle \boldsymbol{x_{a1}}, ..., \boldsymbol{x_{am}}\rangle$ (m is the number of attribute-tokens in the input), the pointer generator aims to generate a sequence of pointer-indexes $I = \langle i_1, ..., i_g\rangle$ (g is the number of attribute-tokens in the target content-plan). Here, i_g indicates an index that points to an attribute-token. The pointer generator uses LSTM to encode the attribute-token that are selected as part of the content-plan in the previous time-step $\boldsymbol{x}_{i_{g-1}}$ as a context vector (cf. Eq. 4) to compute the attention of the attributes. The pointer generator predicts the next index by selecting attribute-token with the highest attention (cf. Eq. 6) that are computed as follows.

$$\boldsymbol{c_{ptr}}_g = f_{lstm}(\boldsymbol{x}_{i_{g-1}}) \tag{4}$$

$$\boldsymbol{u_{ptr}}_g = \tanh(\boldsymbol{W_{p}}_1 \boldsymbol{x_a} + \boldsymbol{W_{p}}_2 \boldsymbol{c_{ptr}}_g) \tag{5}$$

$$\hat{i}_g = \text{softmax}(\boldsymbol{u_{ptr}}_g) \tag{6}$$

Here, $\hat{i}_g$ is the pointer-index output probability distribution over the vocabulary (in this case, the vocabulary is the attribute-token input), f_{lstm} is a single LSTM unit, and $\boldsymbol{W_p}_1$ and $\boldsymbol{W_p}_2$ are learned parameters. The pointer generator is trained to maximize the conditional log-likelihood:

$$p(I_d \mid A_d) = \sum_g \sum_{j=1}^{j=m} {i'}_{g,j} \times \log \hat{i}_{g,j} \tag{7}$$

$$J_{ptr} = \frac{1}{D} \sum_{d=1}^{D} -\log p(I_d \mid A_d) \tag{8}$$

where (A_d, I_d) is a pair of attributes and target pointer-index (generated by finding the position of the attribute-token of the target content-plan in the original input) given for training, i' is the matrix of the target pointer-index over the vocabulary, D is the number of records in the dataset and J_{ptr} is the objective function of the pointer generator.

Content-plan Generator and Encoder. The pointer-index I is a sequence of indexes that refers to the attribute-token X in a proper order that represents a content-plan. Hence, the content-plan generator uses the pointer-index to rearrange the sequence of attribute-token into a content-plan X'. In the content-plan encoder, we use LSTM to encode the learned content-plan X' to capture the relationships between attributes. The hidden states of the content-plan encoder $\langle \boldsymbol{x_{cp}}_1, ..., \boldsymbol{x_{cp}}_g \rangle$ are forwarded to the text generator to help its attention model select attributes in a proper order.

4.4 Description Generation

We adapt the encoder-decoder model (Bahdanau, Cho, and Bengio 2015) to generate entity description by integrating a content-plan to help the attention mechanism of the encoder-decoder computes the attributes to be mentioned and their order. For this adaptation, we propose the *content-plan-based bag of tokens* attention model.

Content-plan-based Bag of Tokens Attention. We use the same encoder as in the content-plan generation module (cf. Section 4.3). This encoder treats the attributes as a bag of tokens to allow our model handles disordered input. However, this representation does not capture the relationships between attributes. To capture the relationships between attributes, we use LSTM to encode the content-plan (cf. Section 4.3). We integrate the learned (encoded) content-plan into the encoder-decoder model to help the attention mechanism of the model selects the attributes in a proper order. This way, our proposed model has two advantages. First, our model yields the same vector representation for the same set of attributes regardless of their order while capturing the relationships between attributes via the content-plan. Second, our model computes a context vector based on the original input (i.e., attributes) and the content-plan, and hence reduces the error propagation (e.g., missing attribute errors).

The integration of the learned content-plan in the encoder-decoder model is done by adapting the coverage mechanism (Tu et al. 2016; See, Liu, and Manning 2017) as follows. First, we use a coverage vector $\boldsymbol{d_{cov}}$ to keep track of the content-plan history. We use the sum of a content-plan attention distribution $\boldsymbol{a_{cp}}_{l'}$ from the previous decoding (i.e., description generator) time-step to maintain the information about which attributes in the content-plan have been exposed (cf. Eq. 9). Second, we use the coverage vector $\boldsymbol{d_{cov}}$ to compute the attention (weight) of the content-plan $\boldsymbol{a_{cp}}$ (cf. Eq. 10 and 11). Then, the content-plan attention is used as an additional input to compute the attention of the attribute-token $\boldsymbol{a_d}$ (cf. Eq. 12). Finally, the attribute-token attention is used to compute a context vector $\boldsymbol{c_d}$ for the decoder (cf. Eq. 13).

$$\boldsymbol{d_{cov}}_l = \sum_{l'=0}^{l-1} \boldsymbol{a_{cp}}_{l'} \tag{9}$$

$$\boldsymbol{u_{cp}}_l = \tanh(\boldsymbol{W}_{c1}\boldsymbol{x_{cp}} + \boldsymbol{W}_{c2}\boldsymbol{h}_{dl-1} + \boldsymbol{w}_{c3}\boldsymbol{d_{cov}}_l) \tag{10}$$

$$\boldsymbol{a_{cp}}_l = \sum_g \text{softmax}(\boldsymbol{u_{cp}}_l)\boldsymbol{x_{cp}}_g \tag{11}$$

$$\boldsymbol{a_d}_l = \tanh(\boldsymbol{W}_{d1}\boldsymbol{x_a} + \boldsymbol{W}_{d2}\boldsymbol{h}_{dl-1} + \boldsymbol{W}_{d3}\boldsymbol{a_{cp}}_l) \tag{12}$$

$$\boldsymbol{c_d}_l = \sum_m \text{softmax}(\boldsymbol{a_d}_l)\boldsymbol{x_a}_m \tag{13}$$

Here, $\boldsymbol{h_d}_l$ is the decoder hidden state at time-step l, and $\boldsymbol{W}$ and $\boldsymbol{w}$ are learned parameters.

Description Generator. We use LSTM for the description generator (i.e., the decoder). The decoder predicts the next token of the description conditioned by the previous hidden state of the decoder $\boldsymbol{h_d}_{l-1}$, the previous generated token t_{l-1}, and the context vector $\boldsymbol{c_d}_l$.

$$\boldsymbol{h_d}_l = f_{lstm}([\boldsymbol{h_d}_{l-1}; \boldsymbol{t}_{l-1}; \boldsymbol{c_d}_l]) \tag{14}$$

$$\hat{t}_l = \text{softmax}(\boldsymbol{V}\boldsymbol{h_d}_l) \tag{15}$$

Here, $\hat{t}_l$ is the output probability distribution over the vocabulary, and $\boldsymbol{V}$ is the hidden-to-output weight matrix. The decoder is trained to maximize the conditional log-likelihood:

$$p(S_d \mid A_d) = \sum_l \sum_{j=1}^{j=|V|} t'_{l,j} \times \log \hat{t}_{l,j} \tag{16}$$

$$J_{dec} = \frac{1}{D}\sum_{d=1}^{D} -\log p(S_d \mid A_d) \tag{17}$$

$$J = J_{ptr} + J_{dec} \tag{18}$$

where (A_d, S_d) is a pair of attributes and entity description given for training, t' is the matrix of the target token description over the vocabulary V, J_{dec} is the objective function of the description generator, and J is the overall objective function of our proposed model.

5 Experiments

We evaluate our model on two real-world datasets, including WIKIALL and WIKIBIO datasets. The attributes in WIKIALL are disordered since they are the result of a query to Wikidata. Meanwhile, the attributes in WIKIBIO are practically ordered, i.e., the salient attributes are relatively ordered but may have noises (non-salient attributes) between them. To test on disordered attributes of WIKIBIO dataset, we randomly shuffle the attributes.

5.1 Models

We compare our proposed model[2] with four existing models including: (1) the Field Gating with Dual Attention model (**FGDA**), which is the state-of-the-art table-to-text generation model (Liu et al. 2018); (2) the Graph-based Triple LSTM encoder model (**GTRLSTM**), which is the state-of-the-art rdf-to-text generation model (Trisedya et al. 2018); (3) the Neural Content-Planning model (**NCP**), which is a data-to-text generation model that uses content-plan as one of its features (Puduppully, Dong, and Lapata 2019); and (4) the modified encoder-decoder model (**MED**), which is a modification of the standard encoder-decoder model that uses the embeddings of its input instead of the hidden state of the encoder to compute the attention.

5.2 Results

Table 3 shows that our proposed model achieves a consistent improvement over the baselines, and the improvement is statistically significant, with $p < 0.01$ based on the t-test of

[2]Code and dataset: http://www.ruizhang.info/GKB/gkb.htm

Model	WIKIALL				WIKIBIO (disordered)				WIKIBIO (ordered)			
	BLEU↑	ROUGE↑	METEOR↑	TER↓	BLEU↑	ROUGE↑	METEOR↑	TER↓	BLEU↑	ROUGE↑	METEOR↑	TER↓
MED	58.54	54.01	42.80	34.40	38.21	33.32	30.20	55.20	40.25	35.63	30.90	56.40
GTRLSTM	62.71	57.02	44.30	29.50	42.64	38.35	32.60	54.40	42.06	37.90	32.20	54.80
NCP	63.21	57.28	44.30	28.50	43.07	38.76	33.90	53.20	43.12	38.82	33.90	53.30
FGDA	62.61	57.11	44.10	30.20	42.31	38.93	32.20	55.00	44.59	40.20	34.10	52.10
Proposed	**65.12**	**58.07**	**45.90**	**27.50**	**45.32**	**40.80**	**34.40**	**51.50**	**45.46**	**40.31**	**34.70**	**51.30**

Table 3: Experimental results.

Model	Correctness	Grammaticality	Fluency
MED	2.25	2.32	2.26
GTRLSTM	2.31	2.40	2.36
NCP	2.54	2.68	2.51
FGDA	2.51	2.58	2.54
Proposed	**2.68**	**2.76**	**2.57**

Table 4: Human evaluation results.

the BLEU scores. We use MultEval to compute the p value based on an approximate randomization (Clark et al. 2011). Our model achieves higher BLEU and ROUGE scores than the baselines, which indicate that our model generates descriptions with a better order of attribute mention. Moreover, the better (lower) TER scores indicate that our model generates a concise description (i.e., following the content-plan).

On disordered input experiments, the content-plan based models (i.e., our proposed model and NCP) achieve stable performance with our model getting the highest score on all metrics. These results show that content-planning helps neural data-to-text generation models select and arrange the data (i.e., attributes) to be mentioned in the text.

The content-planner (the pointer networks) achieves 83.41 and 87.12 BLEU scores on the WIKIBIO and WIKIALL datasets, respectively. We further conduct experiments to show that our model reduces error propagation between the content-planner and the text generator. We use the content-plan gold standard (i.e., the target content-plan extracted from the description, cf. Section 4.2) as the input of the text generator. On this setup, our model achieves comparable performance with NCP. Our model achieves 46.5 and 66.97 BLUE scores on the WIKIBIO and WIKIALL datasets, respectively. Meanwhile, NCP achieves 46.3 and 66.69 BLUE scores on the WIKIBIO and WIKIALL datasets, respectively. These results are expected because both models take the same content-plan.

Human Evaluations. Following Trisedya et al. (2018), we conduct manual evaluations on the generated descriptions using three metrics including *correctness*, *grammaticality*, and *fluency*. *Correctness* is used to measure the semantics of the generated description (i.e., contains wrong order of attribute mention or not, e.g., `"born in USA, New York"`); *grammaticality* is used to rate the grammatical and spelling errors; and *fluency* is used to measure the fluency of the output (e.g., contain repetition or not). For each metric, a score of 3 is given to output that contains no errors; a score of 2 is given to output that contains one error; and a score of 1 is given to output that contains more than one error. We randomly choose 300 records of WIKIALL dataset along with the output of each model. We manage to get six annotators who have studied English for at least ten years and completed education in an English environment for at least two years. The total time spent for these evaluations is around 250 hours. Table 4 shows the results of the human evaluations. The results confirm the automatic evaluations in which our proposed model achieves the best scores.

6 Conclusions and Future Work

We proposed an end-to-end data-to-text generation model on top of an encoder-decoder framework that includes content-planning to address the problem of disordered input. Our model employs joint learning of content-planning and text generation to reduce error propagation between them for generating a description of an entity from its attributes. To integrate a content-plan into the encoder-decoder framework, we propose the *content-plan-based bag of tokens* attention model. Our attention model effectively captures salient attributes in a proper order. Experimental results show that our proposed model outperforms the baselines and achieves the highest score in all metrics on the WIKIALL and WIKIBIO test datasets. Moreover, our model obtains stable performance on disordered input and achieves a consistent improvement over the baselines by up to 5%.

The proposed model requires training data in the form of triples of attributes, content-plan, and description. However, extracting a content-plan from a description is a non-trivial task. We use string matching to find the order of attributes in the description as a content-plan. However, the string matching does not capture the semantic similarity between attributes and text. From the example in Table 1, the extracted content-plan does not include attribute `citizenship` since the string matching cannot capture the similarity between `United States` and `American`. We consider using semantic similarity search as future work.

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 Google Faculty Research Award.

References

Angeli, G.; Liang, P.; and Klein, D. 2010. A simple domain-independent probabilistic approach to generation. In *EMNLP*, 502–512.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Bao, J.; Tang, D.; Duan, N.; Yan, Z.; Lv, Y.; Zhou, M.; and Zhao, T. 2018. Table-to-text: Describing table region with natural language. In *AAAI*, 5020–5027.

Barzilay, R., and Lapata, M. 2005. Collective content selection for concept-to-text generation. In *EMNLP*, 331–338.

Belz, A. 2008. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering* 14(4):431–455.

Bordes, A.; Chopra, S.; and Weston, J. 2014. Question answering with subgraph embeddings. In *EMNLP*, 615–620.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 1724–1734.

Clark, J. H.; Dyer, C.; Lavie, A.; and Smith, N. A. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *ACL*, 176–181.

Duboue, P. A., and Mckeown, K. R. 2001. Empirically estimating order constraints for content planning in generation. In *ACL*, 172–179.

Duboue, P. A., and Mckeown, K. R. 2002. Content planner construction via evolutionary algorithms and a corpus-based fitness function. In *INLG*, 89–96.

Duboue, P. A., and McKeown, K. R. 2003. Statistical acquisition of content selection rules for natural language generation. In *EMNLP*, 121–128.

Gardent, C.; Shimorina, A.; Narayan, S.; and Perez-Beltrachini, L. 2017. Creating training corpora for nlg micro-planners. In *ACL*, 179–188.

Geographic Knowledge Base. http://www.ruizhang.info/GKB/gkb.htm.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Kim, J., and Mooney, R. J. 2010. Generative alignment and semantic parsing for learning from ambiguous supervision. In *COLING*, 543–551.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Lebret, R.; Grangier, D.; and Auli, M. 2016. Neural text generation from structured data with application to the biography domain. In *EMNLP*, 1203–1213.

Liu, T.; Wang, K.; Sha, L.; Sui, Z.; and Chang, B. 2018. Table-to-text generation by structure-aware seq2seq learning. In *AAAI*, 4881–4888.

Liu, T.; Ma, S.; Xia, Q.; Luo, F.; Chang, B.; and Sui, Z. 2019. Hierarchical encoder with auxiliary supervision for table-to-text generation: Learning better representation for tables. In *AAAI*, 6786–6793.

Lu, W., and Ng, H. T. 2011. A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *EMNLP*, 1611–1622.

Lu, W.; Ng, H. T.; and Lee, W. S. 2009. Natural language generation with tree conditional random fields. In *EMNLP*, 400–409.

Marcheggiani, D., and Perez-Beltrachini, L. 2018. Deep graph convolutional encoders for structured data to text generation. In *INLG*, 1–9.

McKeown, K. R.; Jordan, D. A.; Pan, S.; Shaw, J.; and Allen, B. A. 1997. Language generation for multimedia healthcare briefings. In *ANLP*, 277–282.

McKeown, K. 1992. *Text Generation*. Cambridge University Press.

Mei, H.; Bansal, M.; and Walter, M. R. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *NAACL-HLT*, 720–730.

Mellish, C.; Knott, A.; Oberlander, J.; and O'Donnell, M. 1998. Experiments using stochastic search for text planning. In *Natural Language Generation*, 98–107.

Puduppully, R.; Dong, L.; and Lapata, M. 2019. Data-to-text generation with content selection and planning. In *AAAI*, 6908–6915.

Reiter, E., and Dale, R. 2000. *Building natural language generation systems*. Cambridge University Press.

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*, 1073–1083.

Serban, I. V.; García-Durán, A.; Gulcehre, C.; Ahn, S.; Chandar, S.; Courville, A.; and Bengio, Y. 2016. Generating factoid questions with recurrent neural networks: The 30M factoid question-answer corpus. In *ACL*, 588–598.

Sha, L.; Mou, L.; Liu, T.; Poupart, P.; Li, S.; Chang, B.; and Sui, Z. 2018. Order-planning neural text generation from structured data. In *AAAI*, 5414–5421.

Trisedya, B. D.; Qi, J.; Zhang, R.; and Wang, W. 2018. Gtr-lstm: A triple encoder for sentence generation from rdf data. In *ACL*, 1627–1637.

Trisedya, B. D.; Weikum, G.; Qi, J.; and Zhang, R. 2019. Neural relation extraction for knowledge base enrichment. In *ACL*, 229–240.

Tu, Z.; Lu, Z.; Liu, Y.; Liu, X.; and Li, H. 2016. Modeling coverage for neural machine translation. In *ACL*, 76–85.

van Deemter, K.; Krahmer, E.; and Theune, M. 2005. Real versus template-based natural language generation: A false opposition? *Computational Linguistics* 31(1):15–24.

Vinyals, O.; Bengio, S.; and Kudlur, M. 2016. Order matters: Sequence to sequence for sets. In *ICLR*.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In *NIPS*, 2692–2700.

Wiseman, S. J.; Shieber, S. M.; and Rush, A. S. M. 2017. Challenges in data-to-document generation. In *EMNLP*, 2253–2263.