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

Introduction

- Data→Text
- Attribute→Description
- Attribute:

  $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, \cdots, \langle k_n, v_n\rangle\}$
- Description:

  $S = \langle t_1, t_2, \ldots, t_l\rangle$

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.

Summary

- 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.
- Propose a content-plan-based bag of tokens attention model to effectively capture salient attributes in a proper order based on a content-plan

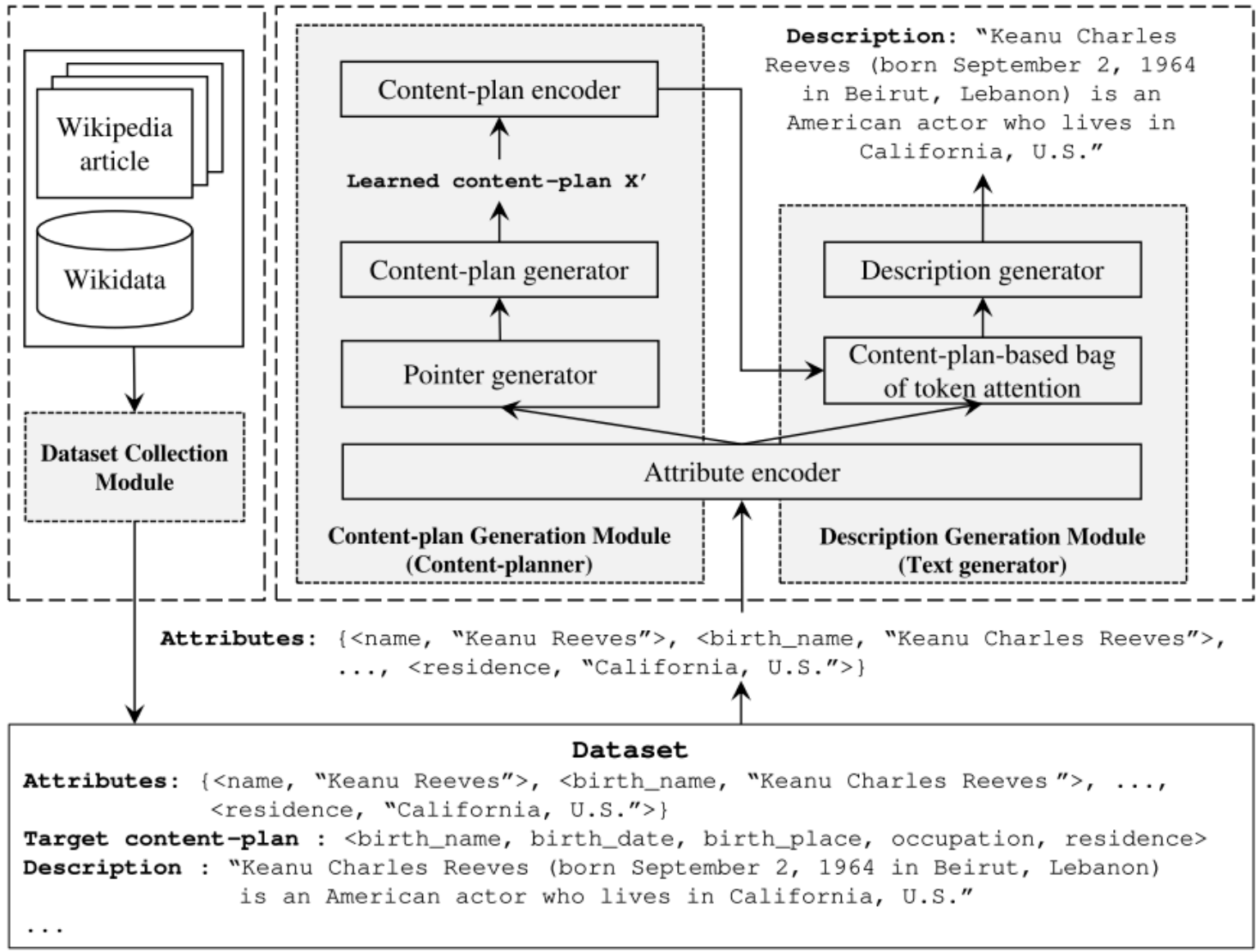
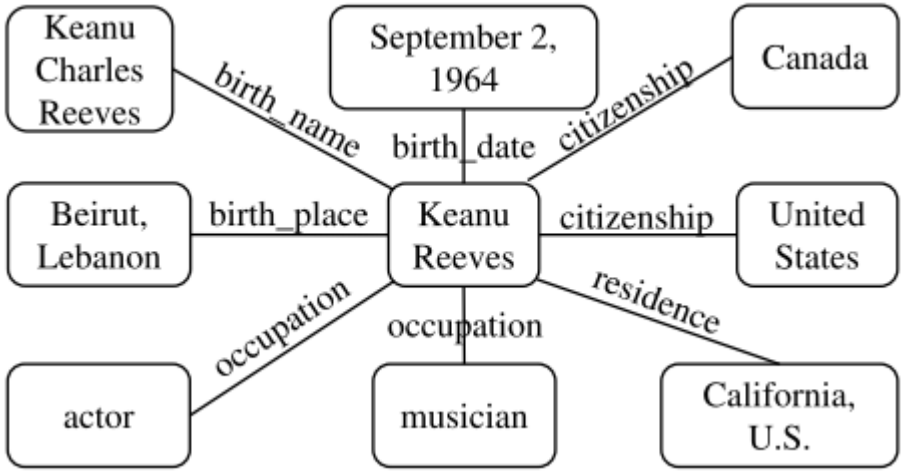


Figure 2: Overview of our proposed solution

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.



Firstly, extract the first sentence of a Wikipedia page of a ctarget entity as the description. **Then**, extract the attributes of a target entity by querying Wikidata for RDF triples that contain the target entity as the subject

- **Attributes**
- **Content-plan**
- **Entity description**

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

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.

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)

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

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

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

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

Pointer Generator. Given a sequence of attribute-token vectors that are computed by the attribute encoder $X = \langle \boldsymbol{x}_{\boldsymbol{a}1}, ..., \boldsymbol{x}_{\boldsymbol{a}m} \rangle$ (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.

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

$$\boldsymbol{u}_{\boldsymbol{ptr}_g} = \tanh(\boldsymbol{W}_{\boldsymbol{p}_1}\boldsymbol{x}_{\boldsymbol{a}} + \boldsymbol{W}_{\boldsymbol{p}_2}\boldsymbol{c}_{\boldsymbol{ptr}_g}) \tag{5}$$

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

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 $\boldsymbol{W}_{\boldsymbol{p}_1}$ and $\boldsymbol{W}_{\boldsymbol{p}_2}$ are learned parameters. The pointer generator is trained to maximize the conditional log-likelihood:

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

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

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 $\langle \boldsymbol{x}_{\boldsymbol{cp}_1}, ..., \boldsymbol{x}_{\boldsymbol{cp}_g} \rangle$ are forwarded to the text generator to help its attention model select attributes in a proper order.

Description Generation

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.

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

$$\boldsymbol{u_{cp}}_l = \tanh(\boldsymbol{W_{c1}} \boldsymbol{x_{cp}} + \boldsymbol{W_{c2}} \boldsymbol{h_{d}}_{l-1} + \boldsymbol{w_{c3}} \boldsymbol{d_{cov}}_l) \tag{10}$$

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

$$\boldsymbol{a_{d}}_l = \tanh(\boldsymbol{W_{d1}} \boldsymbol{x_a} + \boldsymbol{W_{d2}} \boldsymbol{h_{d}}_{l-1} + \boldsymbol{W_{d3}} \boldsymbol{a_{cp}}_l) \tag{12}$$

$$\boldsymbol{c_{d}}_l = \sum_{m} \text{softmax}(\boldsymbol{a_{d}}_l) \boldsymbol{x_a}_m \tag{13}$$

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:

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

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

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

Experiments

- FGDA : the Field Gating with Dual Attention model
- GTRLSTM : the Graph-based Triple LSTM encoder model
- NCP : the Neural Content-Planning model
- MED : the modified encoder-decoder model

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.

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.