Ensuring Readability and Data-fidelity using Head-modifier Templates in Deep Type Description Generation

Jiangjie Chen[†], **Ao Wang**[†], **Haiyun Jiang**[†], **Suo Feng**[†], **Chenguang Li**[†], **Yanghua Xiao**[†‡*]
[†]Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, China
[‡]Shanghai Institute of Intelligent Electronics & Systems, Shanghai, China
{jiangjiechen14, awang15, jianghy16, fengs17, cgli17, shawyh}@fudan.edu.cn

Abstract

A type description is a succinct noun compound which helps human and machines to quickly grasp the informative and distinctive information of an entity. Entities in most knowledge graphs (KGs) still lack such descriptions, thus calling for automatic methods to supplement such information. However, existing generative methods either overlook the grammatical structure or make factual mistakes in generated texts. To solve these problems, we propose a head-modifier template-based method to ensure the readability and data fidelity of generated type descriptions. We also propose a new dataset and two automatic metrics for this task. Experiments show that our method improves substantially compared with baselines and achieves state-of-the-art performance on both datasets.

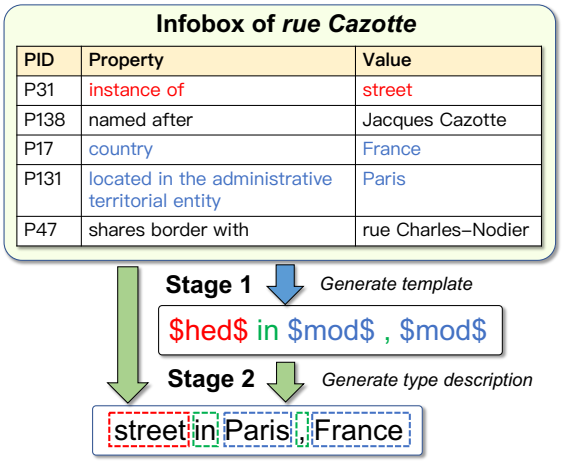

Figure 1: An example of the two-stage generation of our head-modifier template-based method. `$hed$` and `$mod$` are the placeholder for head and modifier components in the template.

1 Introduction

Large-scale open domain KGs such as DBpedia (Auer et al., 2007), Wikidata (Vrandečić and Krötzsch, 2014) and CN-DBpedia (Xu et al., 2017) are increasingly drawing the attention from both academia and industries, and have been successfully used in many applications that require background knowledge to understand texts.

In KGs, a **type description** (Bhowmik and de Melo, 2018) is a kind of description which reflects the rich information of an entity with little cognitive efforts. A type description must be *informative*, *distinctive* and *succinct* to help human quickly grasp the essence of an unfamiliar entity. Compared to other kinds of data in a KG, *types* in entity-typing task (Shimaoka et al., 2016; Ren et al., 2016) are too general and not informative enough (e.g., when asked about "what is `rue Cazotte`?", `street in Paris, France` is obviously more informative and distinctive than a type *location*.), and the fixed type set is too inflexible to expand; while *infobox* and *abstract* are too long with too much information, which increases cognitive burden.

Type descriptions are useful for a wide range of applications, including question answering (e.g. what is *rue Cazotte*?), named entity disambiguation (e.g. Apple (*fruit of the apple tree*) vs Apple (*American technology company*)), taxonomy enrichment, etc. However, many entities in current open-domain KGs still lack such descriptions. For example, in DBpedia and CN-DBpedia respectively, there are only about 21% and 1.8% entities that are provided with such descriptions[1].

Essentially, a type description is a noun compound, which follows a grammatical rule called *head-modifier rule* (Hippisley et al., 2005; Wang et al., 2014). It always contains a *head* component (also head words or heads), and usually contains a *modifier* component (also modifier words or modifiers). The head component representing the type

*Corresponding author: Yanghua Xiao.

[1]According to DBpedia 2016-10 dump and CN-DBpedia 2015-07 dump.

information of the entity makes it *distinctive* from entities of other types; the modifier component limits the scope of that type, making it more fine-grained and *informative*. For example, in `street in Paris, France`, the head word `street` indicates that it is a *street*, and the modifier words `Paris` and `France` indicate the street is located in *Paris, France*.

Due to the low recall and limited patterns of extractive methods (Hearst, 1992), generative methods are more suitable to acquire more type descriptions. Generally, there are several challenges in generating a type description from an infobox: 1) it must be grammatically correct to be readable, given that a trivial mistake could lead to a syntax error (e.g. `street` *`with`* `Paris, France`); 2) it must guarantee the data fidelity towards input infobox, e.g., the system shouldn't generate `street in` *`Germany`* for a French street; 3) its heads must be the correct types for the entity, and a mistake in heads is more severe than in modifiers, e.g., in this case, *`river`* `in France` is much worse than `street in` *`Germany`*.

We argue that the head-modifier rule is crucial to ensure readability and data-fidelity in type description generation. However, existing methods pay little attention to it. Bhowmik and de Melo (2018) first propose a dynamic memory-based generative network to generate type descriptions from infobox in a neural manner. They utilize a memory component to help the model better remember the training data. However, it tends to lose the grammatical structure of the output, as it cannot distinguish heads from modifiers in the generation process. Also, it cannot handle the out-of-vocabulary (OOV) problem, and many modifier words may be rare and OOV. Other data-to-text (Wiseman et al., 2017; Sha et al., 2018) and text-to-text (Gu et al., 2016; Gulcehre et al., 2016; See et al., 2017) models equipped with copy mechanism alleviate OOV problem, without considering the difference between heads and modifiers, resulting in grammatical or factual mistakes.

To solve the problems above, we propose a head-modifier template-based method. To the best of our knowledge, we are the first to integrate head-modifier rule into neural generative models. Our method is based on the observation that a head-modifier template exists in many type descriptions. For example, by replacing heads and modifiers with placeholders `$hed$` and `$mod$`, the template for `street in Paris, France` is `$hed$ in $mod$, $mod$`, which is also the template for a series of similar type descriptions such as `library in California, America`, `lake in Siberia, Russia`, etc. Note that, the `$hed$` and `$mod$` can appear multiple times, and punctuation like a comma is also an important component of a template.

Identifying the head and modifier components is helpful for providing structural and contextual cues in content selection and surface realization in generation, which correspond to data fidelity and readability respectively. As shown in Fig.1, the model can easily select the corresponding properties and values and organize them by the guidance of the template. The head-modifier template is universal as the head-modifier rule exists in any noun compound in English, even in Chinese (Hippisley et al., 2005). Therefore, the templates are applicable for open domain KGs, with no need to design new templates for entities from other KGs.

There are no existing head-modifier templates to train from, therefore we use the dependency parsing technique (Manning et al., 2014) to acquire templates in training data. Then, as presented in Fig.1, our method consists of two stages: in Stage 1, we use an encoder-decoder framework with an attention mechanism to generate a template; in Stage 2, we use a new encoder-decoder framework to generate a type description, and reuse previously encoded infobox and apply a copy mechanism to preserve information from source to target. Meanwhile, we apply another attention mechanism upon generated templates to control the output's structure. We then apply a context gate mechanism to dynamically select contexts during decoding.

In brief, our contributions[2] in this paper include, 1) we propose a new head-modifier template-based method to improve the readability and data fidelity of generating type descriptions, which is also the first attempt of integrating head-modifier rule into neural generative models; 2) we apply copy and context gate mechanism to enhance the model's ability of choosing contents with the guidance of templates; 3) we propose a new dataset with two new automatic metrics for this task, and experiments show that our method achieves state-of-the-art performance on both datasets.

[2]https://github.com/Michael0134/HedModTmplGen

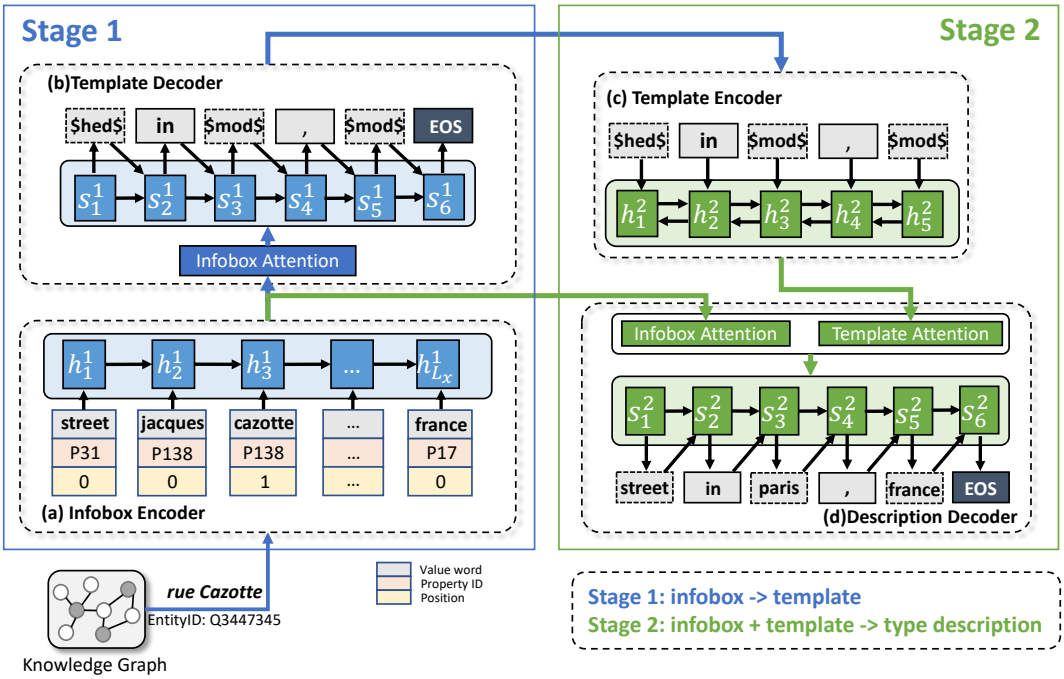

Figure 2: Overall architecture of our method. In Stage 1, the model generates a template from infobox of entity `rue Cazotte` (the entity can be found at Wikidata by EntityID), then in Stage 2 the model completes this template by reusing the infobox and generates a type description for this entity.

2 Method

In this section, we demonstrate our method in detail. As shown in Fig.2, given an entity from Wikidata[3] and its corresponding infobox, we split the generation process into two stages. In Stage 1, the model takes as input an infobox and generates a head-modifier template. In Stage 2, the model takes as input the previously encoded infobox and the output template, and produces a type description. Note that our model is trained in an end-to-end manner.

2.1 Stage 1: Template Generation

In this stage, we use an encoder-decoder framework to generate a head-modifier template of the type description.

2.1.1 Infobox Encoder

Our model takes as input an infobox of an entity, which is a series of (`property`, `value`) pairs denoted as $\mathcal{I}$. We then reconstruct them into a sequence of words to apply Seq2Seq learning. In order to embed structural information from the infobox into word embedding $\mathbf{x}_i$, following Lebret et al. (2016), we represent $\mathbf{x}_i = [\mathbf{v}_{x_i}; \mathbf{f}_{x_i}; \mathbf{p}_{x_i}]$ for the `i`-th word x_i in the values, with the word embedding $\mathbf{v}_{x_i}$ for x_i, a corresponding property embedding $\mathbf{f}_{x_i}$ and the positional information embedding $\mathbf{p}_{x_i}$, and $[\cdot;\cdot]$ stands for vector concatenation.

Property	Value
P31: instance of	street
P138:named after	Jacques Cazotte
P17: country	France
P131: located in the administrative territorial entity	Paris
P47: shares border with	rue Charles–Nodier

word	(property, position)
street	(instance_of, 0)
Jacques	(named_after, 0)
Cazotte	(named_after, 1)
France	(country, 0)
Paris	(located_..._entity, 0)
rue	(shares_border_with, 0)
Charles–Nodier	(shares_border_with, 1)

Figure 3: An example of reconstructing a Wikidata infobox (left) into a sequence of words with property and position information (right). `PN` denotes a property ID in Wikidata.

For example, as shown in Fig.3, we reconstruct (`named after`, `Jacques Cazotte`) into `Jacques` with (`named_after`, `0`) and `Cazotte` with (`named_after`, `1`), as `Jacques` is the first token in the value and `Cazotte` is the second. Next, we concatenate the embedding of `Jacques`, `named_after` and `0` as the reconstructed embedding for `Jacques`. Notice that, we have three separate embedding matrices for properties, value words and position, that is, even though the property `country` is the same string as the value `country`, they are not

[3] www.wikidata.org

the same token.

Then, we employ a standard GRU (Chung et al., 2014) to read the input $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{L_x}$, then produce a sequence of hidden states $\mathbf{H}_x = \{\mathbf{h}_i^1\}_{i=1}^{L_x}$, which are shared in both stages, where L_x is the length of the input sequence.

2.1.2 Template Annotation

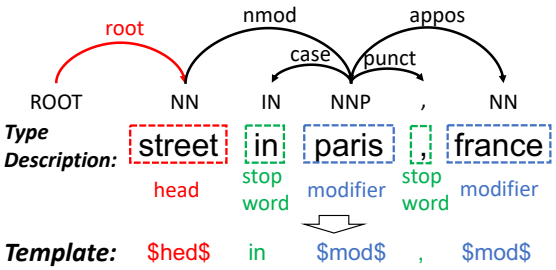

Figure 4: An example of extracting head-modifier template from type description by dependency parsing using Stanford CoreNLP toolkit.

In this task, the type descriptions are diversified yet following the head-modifier rule. The Stage 1 in our model learns the templates from training data, but there are no existing templates for the template generation training. Therefore, we acquire head-modifier templates by using a dependency parser provided by Stanford CoreNLP (Manning et al., 2014).

Specifically, a type description is formed by head words (or heads), modifier words (or modifiers) and conjunctions. In our work, we refer to words that are types as heads in a type description, so there could be multiple heads. For example, `singer` and `producer` in `American singer, producer` are both head words.

During dependency parsing, the `root` of a noun compound is always a head word of the type description. Therefore, we acquire heads by finding the root and its parallel terms. The remaining words except conjunctions and stopwords are considered to be modifiers. We then obtain the template by substituting heads with `$hed$` and modifiers with `$mod$`, as shown in Fig.4.

2.1.3 Template Decoder

In template generation, the template decoder $\mathcal{D}_1$ takes as input the previous encoded hidden states $\mathbf{H}_x$ and produces a series of hidden states $\{\mathbf{s}_1^1, \mathbf{s}_2^1, ..., \mathbf{s}_{L_x}^1\}$ and a template sequence $\mathcal{T} = \{t_1, t_2, ..., t_{L_t}\}$, where L_t is the length of the generated template. As template generation is a relatively lighter and easier task, we apply a canonical attention decoder as $\mathcal{D}_1$, with GRU as the RNN unit.

Formally, at each time step j, the decoder produces a context vector $\mathbf{c}_j^1$,

$$\mathbf{c}_j^1 = \sum_{i=1}^{L_x} \alpha_{ij}\mathbf{h}_i^1; \alpha_{ij} = \frac{\eta(\mathbf{s}_{j-1}^1, \mathbf{h}_i^1)}{\sum_{k=1}^{L_x} \eta(\mathbf{s}_{j-1}^1, \mathbf{h}_i^1))} \quad (1)$$

where $\eta(\mathbf{s}_j^1, \mathbf{h}_i^1)$ is a relevant score between encoder hidden state $\mathbf{h}_i^1$ and a decoder hidden state $\mathbf{s}_j^1$. Among many ways to compute the score, in this work, we apply general product (Luong et al., 2015) to measure the similarity between both:

$$\eta(\mathbf{h}_i^1, \mathbf{s}_{j-1}^1) = \mathbf{h}_i^{1\top}\mathbf{W}_1\mathbf{s}_{j-1}^1 \quad (2)$$

where $\mathbf{W}_1$ is a learnable parameter.

Then the decoder state is updated by $\mathbf{s}_j^1 = GRU([t_{j-1}; \mathbf{c}_j^1], \mathbf{s}_{j-1}^1)$. Finally, the results are fed into a softmax layer, from which the system produces t_j.

2.2 Stage 2: Description Generation

After Stage 1 is finished, the generated template sequence $\mathcal{T}$ and the infobox encoder hidden states $\mathbf{H}_x$ are fed into Stage 2 to produce the final type description.

2.2.1 Template Encoder

As the template is an ordered sequence, we use a bidirectional (Schuster and Paliwal, 1997) GRU to encode template sequence into another series of hidden states $\mathbf{H}_t = \{\mathbf{h}_i^2\}_{i=1}^{L_t}$. Then we fed both $\mathbf{H}_t$ and $\mathbf{H}_x$ to the description decoder for further refinement.

2.2.2 Description Decoder

The description decoder $\mathcal{D}_2$ is a GRU-based decoder, which utilizes a dual attention mechanism: a canonical attention mechanism and a copy mechanism to attend over template representation $\mathbf{H}_t$ and infobox representation $\mathbf{H}_x$ respectively. This is because we need the model to preserve information from the source while maintaining the head-modifier structure learned from the templates.

In detail, let $\mathbf{s}_j^2$ be $\mathcal{D}_2$'s hidden state at time step j. The first canonical attention mechanism is similar to the one described in Section 2.1.3, except that the decoder hidden states are replaced and related learnable parameters are changed. By applying this, we obtain a context vector $\mathbf{c}_j^t$ of $\mathbf{H}_t$ and a context vector $\mathbf{c}_j^x$ of $\mathbf{H}_x$.

Then, we use context gates proposed by Tu et al. (2017) to dynamically balance the contexts from infobox, template, and target, and decide the ratio at which three contexts contribute to the generation of target words.

Formally, we calculate the context gates g_j^* by

$$\begin{aligned} g_j^x &= \sigma(\mathbf{W}_g^x e(y_{j-1}) + \mathbf{U}_g^x \mathbf{s}_{j-1} + \mathbf{C}_g^x \mathbf{c}_j^x) \\ g_j^t &= \sigma(\mathbf{W}_g^t e(y_{j-1}) + \mathbf{U}_g^t \mathbf{s}_{j-1} + \mathbf{C}_g^t \mathbf{c}_j^t) \end{aligned} \tag{3}$$

where $\mathbf{W}_g^*, \mathbf{U}_g^*, \mathbf{C}_g^*$ are all learnable parameters, σ is a sigmoid layer, and $e(y)$ embeds the word y. After that, we apply a linear interpolation to integrate these contexts and update the decoder state:

$$\begin{aligned} \mathbf{c}_j^2 =& (1 - g_j^x - g_j^t)(\mathbf{W} e(y_{j-1}) + \mathbf{U}\mathbf{s}_{j-1}^2) + \\ & g_j^x \mathbf{C}_1 \mathbf{c}_j^x + g_j^t \mathbf{C}_2 \mathbf{c}_j^t \\ \mathbf{s}_j^2 =& GRU([e(y_{j-1}); \mathbf{c}_j^2], \mathbf{s}_{j-1}^2) \end{aligned} \tag{4}$$

where $\mathbf{W}, \mathbf{U}, \mathbf{C}_1, \mathbf{C}_2$ are all learnable parameters.

To conduct a sort of slot filling procedure and enhance the model's ability of directly copying words from infobox, we further apply conditional copy mechanism (Gulcehre et al., 2016) upon $\mathbf{H}_x$. As the produced words may come from the vocabulary or directly from the infobox , we assume a new decoding vocabulary $\mathcal{V}' = \mathcal{V} \cup \{x_i\}_{i=1}^{L_x}$, where $\mathcal{V}$ is the original vocabulary with the vocabulary size of N, and `unk` is the replacement for out-of-vocabulary words.

Following Wiseman et al. (2017), the probabilistic function of y_j is as follows:

$$\begin{aligned} & p(y_j, z_j | y_{<j}, \mathcal{I}, \mathcal{T}) = \\ & \begin{cases} p_{copy}(y_j | y_{<j}, \mathcal{I}, \mathcal{T}) p(z_j | y_{<j}, \mathcal{I}), & z_j = 0 \\ p_{gen}(y_j | y_{<j}, \mathcal{I}, \mathcal{T}) p(z_j | y_{<j}, \mathcal{I}), & z_j = 1 \end{cases} \end{aligned} \tag{5}$$

where z_j is a binary variable deciding whether y_j is copied from $\mathcal{I}$ or generated, and $p(z_j|\cdot)$ is the switcher between copy and generate mode which is implemented as a multi-layer perceptron (MLP). $p_{copy}(y_j|\cdot)$ and $p_{gen}(y_j|\cdot)$ are the probabilities of copy mode and generate mode respectively, which are calculated by applying softmax on copy scores ϕ_{copy} and generation scores ϕ_{gen}. These scores are defined as follows:

$$\begin{aligned} \phi_{gen}(y_j = v) &= \mathbf{W}_g [\mathbf{s}_j^2; \mathbf{c}_j^2], v \in \mathcal{V} \cup \{\texttt{unk}\} \\ \phi_{copy}(y_j = x_i) &= \tanh(\mathbf{h}_i^x \mathbf{W}_c) \mathbf{s}_j^2, x_i \in \mathcal{V}' - \mathcal{V} \end{aligned} \tag{6}$$

where W_c, W_g are both learnable parameters. Therefore, a word is considered as a copied word if it appears in the value portion of the source infobox.

2.3 Learning

Our model is able to be optimized in an end-to-end manner and is trained to minimize the negative log-likelihood of the annotated templates $\mathcal{T}$ given infobox $\mathcal{I}$ and the ground truth type descriptions given $\mathcal{T}$ and $\mathcal{I}$. Formally,

$$\begin{aligned} \mathcal{L}_1 &= -\sum_{i=1}^{L_t} \log p(t_i | t_{<i}, \mathcal{I}) \\ \mathcal{L}_2 &= -\sum_{i=1}^{L_y} \log p(y_i | y_{<i}, \mathcal{I}, \mathcal{T}) \\ \mathcal{L} &= \mathcal{L}_1 + \mathcal{L}_2 \end{aligned} \tag{7}$$

where $\mathcal{L}_1$ is the loss in Stage 1, $\mathcal{L}_2$ is the loss in Stage 2, and L_y is the length of the target.

3 Experiments

In this section, we conduct several experiments to demonstrate the effectiveness of our method.

3.1 Datasets

We conduct experiments on two English datasets sampled from Wikidata, which are referred to as **Wiki10K** and **Wiki200K** respectively. **Wiki10K** is the original dataset proposed by Bhowmik and de Melo (2018), which is sampled from Wikidata and consists of 10K entities sampled from the official RDF exports of Wikidata dated 2016-08-01. However, this dataset is not only too small to reveal the subtlety of models, but it's also relatively imbalanced with too many `human` entities based on the property `instance of`. Therefore, we propose a new and larger dataset **Wiki200K**, which consists of 200K entities more evenly sampled from Wikidata dated 2018-10-01. Note that, in both **Wiki10K** and **Wiki200K**, we filter all the properties whose data type are not `wikibase-item`, `wikibase-property` or `time` according to Wikidata database reports[4].

KGs such as Wikidata are typically composed of semantic triples. A semantic triple is formed by a subject, a predicate, and an object, corresponding to entity, property and value in Wikidata.

[4] https://www.wikidata.org/wiki/Wikidata:Database_reports/List_of_properties/all

We make sure that every entity from both datasets has at least 5 property-value pairs (or *statement* in Wikidata parlance) and an English type description. The basic statistics of the two datasets are demonstrated in Table 1. Then, we randomly divide two datasets into train, validation and test sets by the ratio of 8:1:1.

Datasets	Wiki10K	Wiki200K
# entities	10,000	200,000
# properties	480	900
vocabulary size	28,785	130,686
# *avg* statement	8.90	7.96
Copy(%)	88.24	71.30

Table 1: Statistics for both datasets, where "#" denotes the number counted, and *avg* is short for *average*. "Copy(%)" denotes the copy ratio in the golden type descriptions excluding stopwords, which is similar to the metric **ModCopy** defined in Section 3.2.

3.2 Evaluation Metrics

Following the common practice, we evaluate different aspects of the generation quality with automatic metrics broadly applied in many natural language generation tasks, including BLEU (B-1, B-2) (Papineni et al., 2002), ROUGE (RG-L) (Lin, 2004), METEOR (Banerjee and Lavie, 2005) and CIDEr (Vedantam et al., 2015). BLEU measures the n-gram overlap between results and ground truth, giving a broad point of view regarding fluency, while ROUGE emphasizes on the precision and recall between both. METEOR matches human perception better and CIDEr captures human consensus.

Nonetheless, these metrics depend highly on the comparison with ground truth, instead of the system's input. In this task, the output may still be correct judging by input infobox even if it's different from the ground truth. Therefore, we introduce two simple automatic metrics designed for this task to give a better perspective of the data fidelity of generated texts from the following aspects:

- **Modifier Copy Ratio (ModCopy)**. We evaluate the data fidelity regarding preserving source facts by computing the ratio of modifier words (that is, excluding stopwords and head words) in the type descriptions that are copied from the source. In detail, we roughly consider a word in a type description as a copied word if it shares a *L*-character (4 in our experiments) prefix with any word but stopwords in the values of source infobox. For example, modifier `Japanese` could be a copied modifier word from the fact (`country`, `Japan`). To clarify, the copy ratio of a type description can be calculated by $\frac{\#copied\_words}{\#all\_words - \#stopwords}$. The *Modifier Copy Ratio* measures to what extent the informative words are preserved in the modifiers of the model's output.

- **Head Accuracy (HedAcc)**. For a type description, it is crucial to make sure that the head word is the right type of entity. Therefore, in order to give an approximate estimate of the data fidelity regarding head words, we also evaluate the head word's accuracy in the output. Note that aside from ground truth, infobox is also a reliable source to provide candidate types. Specifically, in Wikidata, the values in `instance of` (P31) and `subclass of` (P279) are usually suitable types for an entity, though not every entity has these properties and these types could be too coarse-grained like `human`. Therefore, after dependency parsing, we count the head words in the output with heads from corresponding ground truth and values of corresponding infobox properties, then gives an accuracy of the heads of output. The *Head Accuracy* measures model's ability of predicting the right type of the entity.

3.3 Baselines and Experimental Setup

We compared our method with several competitive generative models. All models except DGN are implemented with the help of OpenNMT-py (Klein et al., 2017). Note that we use the same infobox reconstructing method described in Section 2.1.1 to apply Seq2Seq learning for all models except DGN since it has its own encoding method. The baselines include:

- **AttnSeq2Seq** (Luong et al., 2015). AttnS2S is a standard RNN-based Seq2Seq model with an attention mechanism.

- **Pointer-Generator** (See et al., 2017). PtrGen is originally designed for text summarization, providing a strong baseline with a copy mechanism. Note that, in order to make

Wiki10K							
Model	**B-1**	**B-2**	**RG-L**	**METEOR**	**CIDEr**	**ModCopy**	**HedAcc**
AttnS2S	53.96	47.56	55.25	29.95	2.753	69.45	52.82
Ptr-Gen	64.24	57.11	65.37	36.42	3.536	83.88	67.92
Transformer	61.63	54.93	63.14	35.01	3.400	75.37	61.13
DGN	63.24	57.52	64.50	35.92	3.372	77.53	64.65
Our work	**65.09**	**58.72**	**66.92**	**37.55**	**3.717**	**86.04**	**70.68**
Wiki200K							
Model	**B-1**	**B-2**	**RG-L**	**METEOR**	**CIDEr**	**ModCopy**	**HedAcc**
AttnS2S	66.15	61.61	70.55	37.65	4.105	49.59	79.76
Ptr-Gen	70.13	66.21	75.21	41.38	4.664	**58.27**	85.38
Transformer	69.78	66.07	75.60	41.52	4.654	53.85	85.55
DGN	62.60	57.86	69.30	34.84	3.815	48.30	81.31
Our work	**73.69**	**69.59**	**76.77**	**43.54**	**4.847**	58.14	**85.81**

Table 2: Evaluation results of different models on both datasets.

a fairer comparison with our model, we additionally equip Ptr-Gen with context gate mechanism so that it becomes a no-template version of our method.

- **Transformer** (Vaswani et al., 2017). Transformer recently outperforms traditional RNN architecture in many NLP tasks, which makes it also a competitive baseline, even if it's not specifically designed for this task.

- **DGN** (Bhowmik and de Melo, 2018). DGN uses a dynamic memory based network with a positional encoder and an RNN decoder. It achieved state-of-the-art performance in this task.

In experiments, we decapitalize all words and keep vocabularies at the size of 10,000 and 50,000 for **Wiki10K** and **Wiki200K** respectively, and use `unk` to represent other out-of-vocabulary words.

For the sake of fairness, the hidden size of RNN (GRU in our experiments) and Transformer in all models are set to 256. The word embedding size is set to 256, and the property and position embedding sizes are both set to 128. During training, we use Adam (Kingma and Ba, 2014) as the optimization algorithm.

3.4 Results and Analysis

The experimental results of metrics described in Section 3.2 are listed in Table 2. In general, our method achieves state-of-the-art performance over proposed baselines.

As shown in the table, our method improves substantially compared with standard encoder-decoder models (AttnS2S and Transformer) and the previous state-of-the-art method (DGN). Interestingly, DGN is out-performed by Ptr-Gen in **Wiki10K** and by most of the models in the larger dataset **Wiki200K**. We also notice that Transformer performs much better on **Wiki200K**, which is most likely because of its learning ability through massive training data. These results further prove the necessity of proposing our new dataset. Among baselines, Ptr-Gen achieves relatively better results due to copy mechanism and context gate mechanism. These mechanisms give the model the ability to cope with the OOV problem and to directly preserve information from the source, which is important in this task. Note that, as described in Section 3.3, we enhance the Pointer-Generator to become a no-template version of our model, therefore the effect of the head-modifier template can be measured by comparing the results of these two methods. And the results demonstrate that our head-modifier template plays an important role in generating type descriptions.

In terms of the two proposed metrics, we find these metrics roughly positively correlated with traditional metrics, which in a way justifies our metrics. These metrics provide interesting points of view on measuring generation quality. The performance on **ModCopy** indicates that methods (Ptr-Gen, ours) with copy mechanism improves data fidelity by copying facts from the source, and the template helps the model know where and how

to copy. The performance on **HedAcc** demonstrates that our method is relatively better at predicting types for an entity, which in a way suggests the templates help the generated text maintain the head-modifier structure so that the head word is successfully parsed by the dependency parsing technique. Although, we notice that in **Wiki200K**, models perform relatively worse on **ModCopy** and better on **HedAcc** than in **Wiki10K**. This is most likely because the types of entities are finite, and more training data leads to more accuracy in predicting types. Due to the size of the dataset and the limit of vocabulary size, the factual information is harder to preserve in the output. This again proves the necessity of the new dataset.

3.4.1 Manual Evaluation

In this task, the readability of the generated type description is mostly related to its grammatical correctness, which benefits from the head-modifier templates. Therefore, in order to measure the influence the templates make in terms of readability as well as how **ModCopy** (**M.C.**) and **HedAcc** (**H.A.**) correlate with manual judgment, we manually evaluate the generation from two aspects: *Grammar Accuracy* (G.A.) and *Overall Accuracy* (O.A.). In detail, *Grammar Accuracy* is the grammatical correctness judging by the grammar of the generated text alone; *Overall Accuracy* is the grammatical and de facto correctness of the generated type description given an infobox and the ground truth. Note that *Overall Accuracy* is always lower than or equal to *Grammar Accuracy*.

In our experiment, we randomly select 200 pieces of data from the test set of **Wiki200K**, and provide the results of each method to the volunteers (who are all undergraduates) for manual evaluation. We make sure each result is evaluated by two volunteers so as to eliminate the influence of subjective factors to some extent.

Model	*G.A.*	*O.A.*	**M.C.**	**H.A.**
AttnS2S	92.25	50.50	51.53	80.27
Ptr-Gen	90.00	65.00	**62.50**	88.01
Transformer	95.25	58.00	55.70	89.67
DGN	89.50	56.00	47.29	81.37
Our work	**96.50**	**66.25**	61.32	**90.29**

Table 3: Results of manual evaluation as well as two proposed metrics.

The results, as shown in Table 3, prove again the effectiveness of our method. Our method outperforms other baselines in term of *Grammar Accuracy*, which demonstrates that the model benefits from the head-modifier templates in term of readability by knowing "how to say it". In particular, the templates improves the *Grammar Accuracy* substantially compared with Ptr-Gen. Results on the *Overall Accuracy* indicate that our method ensures readability as well as data-fidelity, which indicates that the model benefits from the templates by knowing "what to say". As for the proposed metrics **ModCopy** and **HedAcc**, they are, in line with intuition, relatively positively correlated with human judgment in general. Also, notice that the statistics on both metrics are consistent with Table 2.

3.4.2 Effect of Templates

We aim to investigate whether the model is able to correct itself if the template generated in Stage 1 deviates from the correct one. We select cases from **Wiki10K** test set to conduct experiments. During inference, we deliberately replace the template in Stage 2 to see if the generated text still complies with the given template or if the model will be able to generate the right type description.

```
Entity ID: Q859415
Gold: commune in paris, france
Template 1: $hed$ in $mod$, $mod$
Output 1: commune in paris, france
Template 2: $mod$ $hed$
Output 2: commune in france
Template 3: $hed$ $mod$
Output 3: commune
```

```
Entity ID: Q18758590
Gold: italian architect and teacher
Template 1: $mod$ $hed$ and $hed$
Output 1: italian architect and architect
Template 2: $mod$ $hed$
Output 2: italian architect
Template 3: $hed$ $mod$ and $mod$
Output 3: italy and teacher
```

Figure 5: Examples of replacing templates. `Template 1`'s are the inital generated templates, while the remaining ones are produced by the authors. We use **bold** to denote the heads and use *italic red* to denote mistaken words.

The experimental results, as presented in Fig. 5, show our method's resilience against mistaken templates. In the first case: 1) the replaced template `Template 2` is obviously inconsistent with the golden template `Template 1` (though

it's also a possible template for other type descriptions), yet the model still manages to generate a type description though `paris` is lost; 2) `Template 3` doesn't have the conjunction `in`, which causes confusion but the model still successfully predicts the right head.

In the second case, the model originally generates repetitive heads: 1) in `Template 2`, we delete the second `$hed$` in `Template 1`, and as a result, the model successfully generates a correct though incomplete output; 2) while `Template 3` is completely wrong judging by the head-modifier rule, and as a result `Output 3` is lost in readability. Nevertheless, due to the fact that the number of type descriptions is infinite yet the number of head-modifier templates is rather finite, the model can hardly generate a template that's completely wrong, therefore this scenario rarely happens in real life. Still, the model tries to maintain a similar structure and successfully keeps data fidelity by predicting `teacher`, and preserving `italy`.

4 Related Work

There has been extensive work on mining entity-type pairs (i.e. isA relations) automatically. Hearst (1992) uses a pattern-based method to extract isA pairs directly from free text with *Hearst Patterns* (e.g., NP_1 *is a* NP_2; NP_0 *such as* $\{NP_1, NP_2, ..., (and|or)\}NP_n$) from which taxonomies can be induced (Poon and Domingos, 2010; Velardi et al., 2013; Bansal et al., 2014). But these methods are limited in patterns, which often results in low recall and precision.

The most related line of work regarding predicting types for entities is entity-typing (Collins and Singer, 1999; Jiang and Zhai, 2006; Ratinov and Roth, 2009), which aims to assign types such as `people`, `location` from a fixed set to entity mentions in a document, and most of them model it a classification task. However, the types, even for those aiming at fine-grained entity-typing (Shimaoka et al., 2016; Ren et al., 2016; Anand et al., 2017) are too coarse-grained to be informative about the entity. Also, the type set is too small and inflexible to meet the need for an ever-expanding KG.

In this task, the structured infobox is a source more suitable than textural data compared with text summarization task (Gu et al., 2016; See et al., 2017; Cao et al., 2018), because not every entity in a KG possesses a paragraph of description. For example, in CN-DBpedia (Xu et al., 2017), which is one of the biggest Chinese KG, only a quarter of the entities have textual descriptions, yet almost every entity has an infobox.

Natural language generation (NLG) from structured data is a classic problem, in which many efforts have been made. A common approach is to use hand-crafted templates (Kukich, 1983; McKeown, 1992), but the acquisition of these templates in a specific domain is too costly. Some also focus on automatically creating templates by clustering sentences and then use hand-crafted rules to induce templates (Angeli et al., 2010; Konstas and Lapata, 2013). Recently with the rise of neural networks, many methods generate text in an end-to-end manner (Liu et al., 2017; Wiseman et al., 2017; Bhowmik and de Melo, 2018). However, they pay little attention to the grammatical structure of the output which may be ignored in generating long sentences, but it is crucial in generating short noun compounds like type descriptions.

5 Conclusion and Future Work

In this paper, we propose a head-modifier template-based type description generation method, powered by a copy mechanism and context gating mechanism. We also propose a larger dataset and two metrics designed for this task. Experimental results demonstrate that our method achieves state-of-the-art performance over baselines on both datasets while ensuring data fidelity and readability in generated type descriptions. Further experiments regarding the effect of templates show that our model is not only controllable through templates, but resilient against wrong templates and able to correct itself. Aside from such syntax templates, in the future, we aim to explore how semantic templates contribute to type description generation.

6 Acknowledgements

We thank the anonymous reviewers for valuable comments. This work was supported by National Key R&D Program of China (No.2017YFC0803700), Shanghai Municipal Science and Technology Major Project (Grant No 16JC1420400).

References

Ashish Anand, Amit Awekar, et al. 2017. Fine-grained entity type classification by jointly learning representations and label embeddings. *arXiv preprint arXiv:1702.06709*.

Gabor Angeli, Percy Liang, and Dan Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512. Association for Computational Linguistics.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Mohit Bansal, David Burkett, Gerard De Melo, and Dan Klein. 2014. Structured learning for taxonomy induction with belief propagation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1041–1051.

Rajarshi Bhowmik and Gerard de Melo. 2018. Generating fine-grained open vocabulary entity type descriptions. In *Proceedings of ACL 2018*.

Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2018. Retrieve, rerank and rewrite: Soft template based neural summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 152–161.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Michael Collins and Yoram Singer. 1999. Unsupervised models for named entity classification. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*.

Marti A Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics.

Andrew Hippisley, David Cheng, and Khurshid Ahmad. 2005. The head-modifier principle and multilingual term extraction. *Natural Language Engineering*, 11(2):129–157.

Jing Jiang and ChengXiang Zhai. 2006. Exploiting domain structure for named entity recognition. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 74–81. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*.

Ioannis Konstas and Mirella Lapata. 2013. A global model for concept-to-text generation. *Journal of Artificial Intelligence Research*, 48:305–346.

Karen Kukich. 1983. Design of a knowledge-based report generator. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 145–150. Association for Computational Linguistics.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. *arXiv preprint arXiv:1603.07771*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2017. Table-to-text generation by structure-aware seq2seq learning. *arXiv preprint arXiv:1711.09724*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Kathleen McKeown. 1992. *Text generation*. Cambridge University Press.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Hoifung Poon and Pedro Domingos. 2010. Unsupervised ontology induction from text. In *Proceedings of the 48th annual meeting of the Association for Computational Linguistics*, pages 296–305. Association for Computational Linguistics.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Xiang Ren, Wenqi He, Meng Qu, Lifu Huang, Heng Ji, and Jiawei Han. 2016. Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1369–1378.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Abigail See, Peter J Liu, and Cristopher D Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks . pages 1–20.

Lei Sha, Lili Mou, Tianyu Liu, Pascal Poupart, Sujian Li, Baobao Chang, and Zhifang Sui. 2018. Order-planning neural text generation from structured data. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Sonse Shimaoka, Pontus Stenetorp, Kentaro Inui, and Sebastian Riedel. 2016. Neural architectures for fine-grained entity type classification. *arXiv preprint arXiv:1606.01341*.

Zhaopeng Tu, Yang Liu, Zhengdong Lu, Xiaohua Liu, and Hang Li. 2017. Context gates for neural machine translation. *Transactions of the Association for Computational Linguistics*, 5:87–99.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575.

Paola Velardi, Stefano Faralli, and Roberto Navigli. 2013. Ontolearn reloaded: A graph-based algorithm for taxonomy induction. *Computational Linguistics*, 39(3):665–707.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Zhongyuan Wang, Haixun Wang, and Zhirui Hu. 2014. Head, modifier, and constraint detection in short texts. In *2014 IEEE 30th International Conference on Data Engineering*, pages 280–291. IEEE.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in Data-to-Document Generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bo Xu, Yong Xu, Jiaqing Liang, Chenhao Xie, Bin Liang, Wanyun Cui, and Yanghua Xiao. 2017. Cndbpedia: A never-ending chinese knowledge extraction system. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 428–438. Springer.