Make Templates Smarter: A Template Based Data2Text System Powered by Text Stitch Model

Bingfeng Luo*, **Zuo Bai**, **Kunfeng Lai***, **Jianping Shen**
PingAn Life Insurance, Shenzhen, China
{luobingfeng981,baizuo822}@pingan.com.cn
{laikunfeng597,shenjianping324}@pingan.com.cn

Abstract

Neural network (NN) based data2text models achieve state-of-the-art (SOTA) performance in most metrics, but they sometimes drop or modify the information in the input, and it is hard to control the generation contents. Moreover, it requires paired training data that are usually expensive to collect. Template-based methods have good fidelity and controllability but require heavy human involvement. We propose a novel template-based data2text system powered by a text stitch model. It ensures fidelity and controllability by using templates to produce the main contents. In addition, it reduces human involvement in template design by using a text stitch model to automatically stitch adjacent template units, which is a step that usually requires careful template design and limits template reusability. The text stitch model can be trained in self-supervised fashion, which only requires free texts. The experiments on a benchmark dataset show that our system outperforms SOTA NN-based systems in fidelity and surpasses template-based systems in diversity and human involvement.

1 Introduction

Data2Text takes structured data like key-value pairs as inputs and generates corresponding texts. It has been used in various applications like automatically generating weather reports (Angeli et al., 2010), restaurant descriptions (Dušek et al., 2019), etc.

Recent works on Data2Text mainly focus on neural network (NN) based models (Liu et al., 2018; Puduppully et al., 2019). Despite their great success,their fidelity (express all data correctly) and controllability (control the generated contents) are always their main issues. For example, for most NN-based generation models, we cannot guarantee that the generated texts do not drop or modify the information in the input. Besides, it is hard to fix errors made by NN-based generation models.

Considering these issues, template-based data2text models (Reiter and Dale, 1997; Bouayad-Agha et al., 2011; Dou et al., 2018) are still widely used in real-world applications. As shown in Fig. 1, one of the most commonly used templates is the slot-filling style template (Dou et al., 2018). It does not require any linguistic knowledge. Users just write a normal sentence but leave the changeable parts as slots. While this kind of system can produce faithful and controllable texts in specific domains, writing these templates typically requires a lot of human efforts, especially when we want to have both variety and fluency in the output texts.

A main reason for intensive human involvement is the template reusability. It is difficult to reuse sentence-level templates in new tasks that usually have different requirements. Therefore, people usually break long templates into smaller template units (TUs). Each TU contains candidate expressions expressing the same topic, and we randomly choose one to use during generation (see Fig. 1). It allows us to recombine existing TUs to cover new tasks. However, different expressions may require different contexts. As we include more candidates in a TU, the difficulty would rise rapidly to find suitable phrases to connect adjacent TUs for all expressions. For example, in Fig. 1, expressions *[Price] price range* and *[Price] price* in *TU-Price* require different prepositions ahead of them. However, if we instead put expressions requiring different contexts into different TUs, the increased number of TUs would raise the cost to use them.

In this paper, we propose a **T**ext **S**titch Powered **T**emplate **S**ystem (TS2), which reduces human involvement in template writing by removing the need to manually write phrases to connect adjacent

*The work was done when Bingfeng Luo and Kunfeng Lai were with PingAn Life. Bingfeng Luo is now at ByteDance (Email: luobingfeng@bytedance.com), Kunfeng Lai is now at Huawei (Email: laikunfeng@huawei.com)

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1057–1062

TUs. As shown in Fig. 1, once the step of manually writing connection phrases is removed, the whole template becomes much simpler, and more expressions can be added in the same TU. Besides, the increased number of TU expressions and the automatically generated connection phrases enable us to generate more diversed and natural sentences. Moreover, unlike NN-based models, since we only generate connection phrases, we can control most of the contents, especially the essential part containing input information. It guarantees output fidelity and delivers better generation controllability.

Since the vocabulary of the connection phrases is limited, we automatically generate text stitch training data by dropping certain words in free texts with simple rules. Therefore, we can train a high-quality text stitch model in a self-supervised paradigm. By experimenting on a benchmark data2text dataset, we find that the text stitch model trained on an open-domain corpus can produce fluent text in most cases. When task-related data are added, our self-supervised TS2 system clearly outperforms the state-of-the-art (SOTA) template system. In addition, it outputs more faithful text compared with the SOTA NN-based system.

2 Approach

2.1 Text Stitch Powered Template System

The templates in our **T**ext **S**titch Powered **T**emplate **S**ystem (**TS2**) can be represented as: $T = TU_1|TU_2|...|TU_n$, where TU_i is a *template unit* (TU). Each TU_i contains several candidates C_{ij} that express the same topic. C_{ij} can be 1) a text snippet, 2) a slot, 3) a combination of them (see Fig. 1). Each TU candidate can be associated with some conditions, that only when the condition is fulfilled would it be used for generation. We can also add constraints that, if some required slots are not present, the TU candidate outputs empty string (see Fig. 1). During the generation phase, TUs are instantiated into text snippets by filling the slots with corresponding input data and randomly selecting a TU candidate with satisfied conditions for each TU. After that, the instantiated TUs are stitched together by our text stitch model.

2.2 Text Stitch Model

Our text stitch model takes in a sequence of text snippets $t_1|t_2|...|t_n$. It inserts connection texts between each pair of $t_i|t_{i+1}$, so that the generated text is fluent and retains the original meaning. Note that it only inserts connection phrases and preserves all the contents in the input. Therefore, compared with traditional encoder-decoder frameworks that generate texts from scratch, edition-based methods are better fits for this setting.

Our text stitch model utilizes a similar architecture to Levenshtein Transformer (Gu et al., 2019) (LevT), but the deletion operation is not used. As shown in Fig. 2, the model uses the encoder-decoder framework but produces the final output in a refinement fashion. On the encoder side, a special token [SEP] is inserted between adjacent input text snippets $t_i|t_{i+1}$ to denote the position to insert the connection texts. Then, the new input, together with randomly initialized position embeddings, are fed to several transformer layers (Vaswani et al., 2017) to produce the input embeddings.

On the decoder side, the refinement process is initiated with the original input text without [SEP] tokens. The token embeddings and position embeddings are also fed into transformer layers to produce an embedding for each token. The encoder attention mechanism (Vaswani et al., 2017) is employed to provide information about the original input and the insertion positions.

The text stitch process is conducted by performing placeholder prediction and token prediction iteratively. In placeholder prediction, we predict how many tokens to insert for each position i:

$$\pi_{plh}(p|i,\mathbf{x},\mathbf{y}) = softmax(\mathbf{W}_{plh}concat(\mathbf{h}'_i,\mathbf{h}'_{i+1})) \quad (1)$$

where $\mathbf{h}'_i$ is the decoder embedding for position i, and $\mathbf{W}_{plh}$ is the weight matrix. Based on the number $0-k_{max}$ of tokens it predicts, several [PLH] tokens are inserted to position i. Then we replace each [PLH] with a token via token prediction:

$$\pi_{tok}(t|i,\mathbf{x},\mathbf{y}) = softmax(\mathbf{W}_{tok}\mathbf{h}'_i) \quad (2)$$

The objective function for each sample is:

$$\sum_{p_i^*\in\mathbf{p}^*} log\pi_{plh}(p_i^*|i,\mathbf{x},\mathbf{y}) + \sum_{t_i^*\in\mathbf{t}^*} log\pi_{tok}(t_i^*|i,\mathbf{x},\mathbf{y}') \quad (3)$$

where $\mathbf{p}^*$ and $\mathbf{t}^*$ are the target placeholders and tokens derived from the ground-truth text. $\mathbf{y}'$ is the output after inserting placeholders $\mathbf{p}^*$ upon $\mathbf{y}$.

2.3 Self-Supervised Training

Compared with traditional text generation, the generation vocabulary of text stitch is limited. This enables automatically generating high-quality training data from free texts.

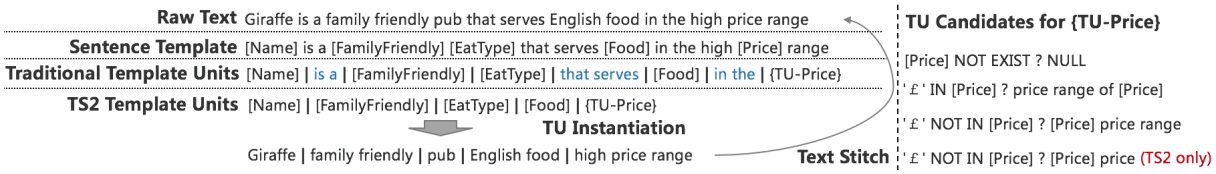

Figure 1: Example templates. 1) Left side shows a sentence with corresponding templates. [*] refers to slots. TUs are seperated by |. {*} refers to a TU ID. Connection phrases are shown in blue. 2) Right side demonstrates the TU candidates in the template unit TU-Price. Condition and result are seperated by ? in each TU candidate.

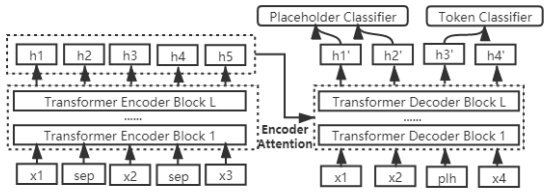

Figure 2: Our text stitch model.

We find tokens with POS tags *adp, aux, cconj, part, punct, sconj, verb* are often parts of a connection phrase. Therefore, for each pair of adjacent sentences, we consider each token tok_i with these POS tags as an indicator of potential segmentation of two TU instantiations. Then we randomly remove 1-5 consecutive tokens to the left and right of tok_i. We require the removed tokens to have the aforementioned POS tags, plus *adj, adv, det, intj, pron* which are also found in connection phrases sometimes but with lower probability. To increase dataset variety, we also randomly skip a segmentation with a probability 5%. The remaining text can be seen as the input to the text stitch model, while the original text is the output. Note that two adjacent sentences are taken as the input instead of one to make sure that our model can also stitch the boundary of two sentences.

3 Experiments

We aim to answer 4 research questions. **Q1)** Does TS2 produce more faithful texts than NN-based systems? **Q2)** Does TS2 produce more natural texts than template-based systems? **Q3)** Can the pre-trained template stitch model be directly used in specific tasks? **Q4)** Does TS2 reduce human efforts compared with pure template-based systems?

Experimental Setup We use the E2E dataset (Dušek et al., 2019) with 42063, 4672, 630 data for training, development and test. Both template- and NN-based systems are extensively studied in E2E, which enables us to have a comprehensive comparison with existing data2text systems. We adopt BERT tokenization, 70K max training steps, and batch-size of 8k tokens. More training details can be found in the appendix.

System	Dropped Slots	Modified Slots	Not Fluent
Prag (SOTA NN)	0.393	0	0.047
TGen (Baseline NN)	0.183	0.040	0.033
TUDA (SOTA Template)	0	0	0
TS2	0	0	0.030
TS2_pt	0	0	0.097
TS2_1k_FT	0	0	0.033
TS_all_FT	0	0	0.027
TS_random	0	0	0.087

Table 1: Human evaluation on 300 random test data.

Method	METEOR	BLEU	NIST	R-L	CIDEr
Prag (SOTA NN)	**45.25**	**68.60**	**8.73**	**70.82**	**2.37**
TGen (Baseline NN)	44.83	65.93	8.61	68.5	2.23
TUDA (SOTA Template)	45.29	56.57	7.45	66.14	1.82
TS2	45.37	56.47	7.48	**66.92**	**1.89**
TS2_1k	38.01	34.26	5.15	55.28	0.58
TS2_10k	44.37	55.00	7.35	65.74	1.84
TS2_pt	43.55	49.85	6.80	64.00	1.19
TS2_1k_ft	44.65	53.08	7.22	66.81	1.61
TS2_10k_ft	44.88	55.62	7.41	66.17	1.75
TS2_all_ft	**45.38**	**56.87**	**7.51**	66.37	1.80
TS2_random	43.72	50.85	6.93	58.72	1.27

Table 2: Automatic evaluation. R-L is ROUGE-L. We hightlight the best NN- and Template-based results. Underline refers to the best score among all systems.

3.1 Train with Unpaired Task-Related Data

We first train the text stitch model using unpaired free texts in the E2E training set in the self-supervised fashion. We compare TS2 with TUDA (Puzikov and Gurevych, 2018), Prag (Shen et al., 2019), TGen (Dušek et al., 2018), which are the SOTA template-based system, SOTA NN-based system, and a baseline NN-based system in the E2E dataset, respectively. To make a fair comparison with TUDA, we follow the template design of TUDA, but remove the connection phrases.

Human Evaluation In human evaluation, we answer **Q1** by evaluating the fidelity of the generated text. We ask two annotators to annotate the generated texts. Conflicts are resolved by discussion between these two annotators. Annotation standards and other details can be found in the appendix.

As shown in Table 1, both SOTA and baseline NN-based systems drop input slots with a high frequency, and not all the outputs are fluent. Even worse, TGen sometimes modifies the original input slot. This leads to unfaithful texts that are unacceptable in many applications. The outputs of the

SOTA template-based system are fluent and perfectly preserve the information in the inputs, yet at the cost of intensive human involvement.

Compared with NN-based systems, TS2 achieves perfect fidelity by using templates to preserve the input information. Although TS2 produces some influent sentences, the fraction of these sentences is relatively small. Besides, we can write simple rules for specific TU pairs to fix the fluency problem, or even use rule-based logic to replace the stitch model, which demonstrates the superiority of TS2 over NN-based systems on controllability.

Compared with template-based systems, although TS2 introduces a small fraction of influent sentences, it significantly reduces human efforts in designing templates. The TUDA code contains 24 *if*s to organize the orders and connection phrases of TUs. In contrast, TS2 only needs 3 templates to constrain the TU orders and does not need to consider the connection phrases. Therefore, significant human efforts are reduced in template design and **Q4** is partially answered.

Automatic Evaluation Automatic metrics compare the generated texts with human-written references. Human tends to alter the words for different slot values. Therefore, we answer **Q2** here to see if TS2 picks more suitable words for different slot values than template-based systems.

As shown in Table 2, TS2 outperforms TUDA in most automatic metrics, which demonstrates that automatically stitching TUs generates more suitable connection texts for the given slot values. However, while TS2 achieves the best METEOR score, it is inferior to NN-based ones in other metrics. The reason lies in the logic of each metric. METEOR calculates the precision and recall of the matched words between the generated and reference texts after alignment by taking paraphrases into account. Therefore, it is less sensitive to expression variations and content orders than other automatic metrics. Template-based systems are usually inferior to NN-based ones in the variety of the output, which leads to lower scores in metrics other than METEOR. While TS2 improves the variety of pure templates, the templates used in TS still make the output less diverse than NN systems.

3.2 Pre-train with Open-Domain Data

To answer **Q3**, we pre-train our text stitch model on an open domain corpus with 1m Wikipedia sentences, 1m newscrawl sentences, 1m web sentences from Leipzig Corpora (Goldhahn et al., 2012).

Pre-trained Model As shown in Table 1 and 2, `TS2_pt` is the model trained only on the open domain corpus. Majority of the sentences produced by `TS2_pt` are fluent and faithful. Most influent sentences have the same pattern: *xxx located in the [Area], located near [Near]*. This is due to the imperfection of the open domain corpus that it has few sentences expressing a place is located in an area and near another site. Since most errors are introduced when stitching slots *[Area]* and *[Near]*, we can easily fix it using simple rules.

Finetuned Model As shown in Table 1 and 2, we also evaluate the text stitch model finetuned on 1k, 10k, and all unpaired texts in E2E training set (`1k_ft`, `10k_ft`, `all_ft`). Finetuning the pre-trained model clearly outperforms the model trained only on the same amount of E2E data (`1k`, `10k`). However, when the size of task-related data is large enough, using a pre-trained model does not deliver much benefits (`TS2` and `TS_all_ft`). On the other hand, finetuning clearly improves the performance of `TS2_pt` in both human and automatic metrics, where using only 1k domain data already produces satisfying scores in human metrics.

3.3 Randomly Arranged Template Units

We also apply the fully finetuned text stitch model to randomly arranged TUs (`TS2_random`), which further answers **Q4** that TS2 is able to fluently stitch most TU pairs and thus reduces human efforts involved when writing templates. We fix the first two TUs to express slots [Name] and [EatType] to avoid TU sequences that can not form fluent sentences. As shown in Table 1, even in this difficult setting, only 0.087 outputs are influent, which are typically caused by rare TU combinations. This implies that TUs can be freely arranged when writing templates. As long as adjacent TU pairs are not rare, there is a high probability that TS2 can output faithful and fluent texts.

4 Conclusion

We propose a novel text generation framework that combines the advantages of both template- and NN-based data2text systems. Compared with NN models, it guarantees the fidelity of the output and improves system controllability. Compared with pure template systems, it produces more varied texts and reduces human involvement in designing templates by removing the need to design connection texts for template units.

References

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.

Nadjet Bouayad-Agha, Gerard Casamayor, and Leo Wanner. 2011. Content selection from an ontology-based knowledge base for the generation of football summaries. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 72–81. Association for Computational Linguistics.

Longxu Dou, Guanghui Qin, Jinpeng Wang, Jin-Ge Yao, and Chin-Yew Lin. 2018. Data2text studio: Automated text generation from structured data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 13–18.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2018. Findings of the e2e nlg challenge. *arXiv preprint arXiv:1810.01170*.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2019. Evaluating the state-of-the-art of end-to-end natural language generation: The E2E NLG Challenge. *arXiv preprint arXiv:1901.11528*.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 552–562.

Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. 2012. Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In *LREC*, volume 29, pages 31–43.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, pages 11179–11189.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2018. Table-to-text generation by structure-aware seq2seq learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6908–6915.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2e nlg challenge: Neural models vs. templates. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 463–471.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Sheng Shen, Daniel Fried, Jacob Andreas, and Dan Klein. 2019. Pragmatically informative text generation. *arXiv preprint arXiv:1904.01301*.

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.

A Training Details

We use the original E2E dataset along with the default splits [1]. As for automatic evaluation, we use the tools provided in the E2E NLG Challenge [2].

We adopt Transformer base (Vaswani et al., 2017), with $d_{model} = 512$, $d_{hidden} = 2048$, $n_{heads} = 8$, $n_{layers} = 6$, $lr_{max} = 0.0005$, label-smooth=0.1, warmup=10000, dropout=0.3, weight-decay=0.01. Source and target side share embeddings. All the models are trained using 1 Nvidia Tesla V100 GPU with the batch size of 8000 tokens, maximum 70K steps for training or finetune on task-related data, and 1.5M steps for pre-training on open-domain. All the hyperparameters follow the default setting of the LevT paper (Gu et al., 2019) except for the training steps. We manually examine the performance of 10K, 30K, 50K, 70K, 90K, 110K training steps of the text stitch model trained on the free text of E2E training data, and find 70K performs best in METEOR. Then, we use 70K for all settings. As for pre-training, we choose the training steps by examined performances 1M, 1.5M, 2M steps. The whole model has about 60M parameters. We use the fairseq (Ott et al., 2019) framework, and it takes about 7 seconds to finish 100 steps.

B Annotation Rules

We randomly select 300 data from the E2E test set for human evaluation, and collect the generated texts of each system on these 300 data. Two paid annotators are asked to annotate the generated texts of each system. They discuss with each other to resolve conflict annotations.

We follow the *fluency* definition of Ferreira et al. (2019) that the sentence is fluent when it is grammatical and flow in a natural, easy to read manner. As for *dropped slots* and *modified slots*, we follow the definition of Puzikov and Gurevych (2018). They refer to the situation that the generated text dropped certain slots or modified the value of certain slots in the input data, respectively.

Since there can exist up to 8 slot-value pairs in the input data, it is very time-consuming to manually examine the dropped-slot errors and modified-slot errors for all data. Therefore, we first examine if the input slot values are present in the output texts. Then, only those sentences with missing slot values are sent to annotators for further examination. Pilot experiments show that this process introduces no extra errors.

C Detailed Templates

See supplementary materials.

[1] https://github.com/tuetschek/e2e-dataset
[2] https://github.com/tuetschek/e2e-metrics