User-oriented Paraphrase Generation with Multiple Types of Constraints

Nazanin Dehghani
Univ Rennes, CNRS, IRISA
Lannion, France
nazanin.dehghani@irisa.fr

Hassan Hajipoor*
Univ Rennes, CNRS, IRISA
Lannion, France
hassan.hajipoor@irisa.fr

Jonathan Chevelu
Univ Rennes, CNRS, IRISA
Lannion, France
jonathan.chevelu@irisa.fr

Gwénolé Lecorvé
Orange / Univ Rennes, CNRS, IRISA
Lannion, France
gwenole.lecorve@orange.com

Abstract

We propose an interactive paraphrase generation system that accepts a variety of user constraints imposed separately or in combination with one another. They include different linguistic factors (surface words, syntax and semantics) and different data shapes (scalar value, sequence and tree). These constraints are integrated in a paraphrase generation process using an attention-based encoder-decoder model trained and experimented on the ParaNMT-50M corpus. The results show that the constraints are well respected by the system and that they allow to improve the quality of the produced paraphrases [2].

1 Introduction

Paraphrase generation, *i.e.* the transformation of a sentence into a well-formed but lexically different one while preserving its original meaning, is a fundamental task of Natural Language Processing (NLP). Paraphrase generation systems are useful tools when users attempt to rephrase some parts of a text [1]. It can also be applied to improve downstream NLP tasks, for instance by increasing the training data with new examples of sentences [2] or by offering variation to the outputs of a given model. Examples of such tasks are text simplification [3] or speech synthesis [4].

In most works, the objective of paraphrase generation is to fit a target domain/task/style (e.g., 'Twitter', 'Shakespeare', etc. [3, 5]). As opposed to these approaches, we focus on applications where the user wants precise control on the generation process by constraining the desired linguistic parameters. To do so, related work have proposed to enforce constraints on the paraphrase's length [6] or its syntactic structure [7, 8].

In this context, we propose a system that shows to what extent the constraint-sensitive paraphrase generation paradigm can be extended. This system has the potentials for real-world use as writing assistant. It can also be used in text simplification and summarization applications [6]. Namely, our contribution is to consider, compare, and combine several types of user constraints, depicting (1) various linguistic aspects (syntax, lexicon and semantics), (2) with different data shapes (scalar value, sequence, tree), (3) obtaining either from the output or input sentence or both. For this

*Corresponding author
[2] http://150.239.171.51:2021/

1st CtrlGen: Controllable Generative Modeling in Language and Vision Workshop at NeurIPS 2021.

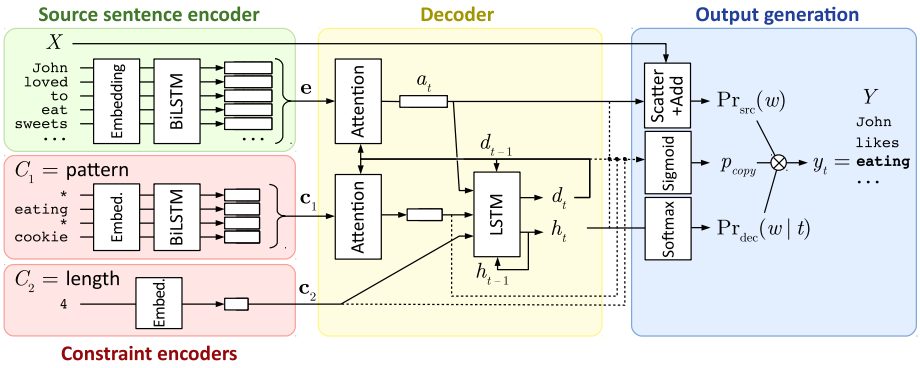


Figure 1: Overview of our constraint-sensitive model.

purpose, multiple attention-based encoder-decoder models[3] are trained following a unique consistent architecture. The performance of the models is investigated across the different types of constraints and their combinations through objective evaluations of the paraphrases, as well as by measuring how often the user constraints are satisfied.

In the remainder, Section 2 details our generic model architecture while Section 3 introduces the different types of constraints tested. Then, Section 4 describes the system in action. The evaluation setup and results are presented in Section 5 and Section 6, respectively.

2 Constraint-sensitive Model

Given an input sentence $X = [x_1, ..., x_{T_X}]$ and a set of constraints $\mathcal{C} = \{C_i\}_{1 \leq i \leq n}$ of target characteristics, our goal is to generate a paraphrase sentence $Y = [y_1, ..., y_{T_Y}]$ that satisfies the constraints. Each constraint can be either a single scalar value, i.e., the desired length, or can be complex/structured data, i.e., a pattern to insert, a syntax tree, etc. In the paper, we refer to this architecture as Constraint-sensitive Paraphrase Generation Network (CPGN).

Inspired by [2], we extend the usual encoder-decoder architecture to include several encoders, one for the source sequence and one for each constraint to apply, as shown in Figure 1. For input sentences and sequential constraints, each encoder relies on an embedding layer, then a bidirectional LSTM layer, while it is limited to an embedding layer for scalar constraints. The decoder relies on a unidirectional LSTM backed up by an attention mechanism for each sequential encoder. Finally, a copy mechanism mixes direct information from the source sentence and from the decoder to decide whether some source tokens should be simply duplicated.

Training Strategy CPGNs need constraints as one of the inputs. However, existing paraphrase datasets only contain the source and the target sentence pairs. To train CPGNs based on a general-purpose paraphrase corpus (`ParaNMT-50M`), first constraints are derived from each source-target pair and then used in inputs.

3 Constraints

This section describes different types of constraints and how they are modeled in this work.

3.1 Length

The output sentence can be constrained based on a desired length, i.e., number of tokens, which is directly represented as a single integer.

[3] As Egonmwan et al. have shown in [9] that the difference between transformer and recurrent neural networks (RNNs) is not so big in paraphrase generation, we use RNNs to be able to compare with related works.

3.2 Syntactic Structure

The syntactic structure of a sentence can be expressed as a parse tree where the internal nodes are phrases and leaves are tokens. To constrain a paraphrase to follow a given syntactic structure, leaves are removed and the remaining tree is linearized using a prefix-bracketed representation. To be user-friendly, the syntactic constraint is relaxed by only keeping the few first top levels of the tree as shown in Example 1. All items of the constraint sequence are mapped to an embedding space, and then contextually embedded with a recurrent layer [4].

Input:	*you ca n't just take her life and forget about ours .*	(Example 1)
Constraint:	*(ROOT (SBARQ (WHADVP) (SQ) (.)))*	
Output:	*why do n't you just take her life and forget about ours ?*	

3.3 Semantic Similarity

Given an input sentence, the goal is to generate a paraphrase with a desired semantic similarity. For this purpose, we use BERTScore [11] which successfully incorporates semantic information behind sentences [12]. For instance, Example 2 shows a pair of paraphrases where their semantic similarity is only 80%. In practice, this similarity cannot be too low as this would break the definition of paraphrases. Hence, in our training dataset, the BERTScore values only range from 0.65 to 1.[5]

Input:	*after all i saw , the idea of going back to mom and dad depresses me .*	(Example 2)
Constraint:	BERTScore Similarity = *0.8*	
Output:	*and my dad 's gon na hurt me .*	

3.4 Pattern of Desired Words

It is common that a user wants to include specific words in the output paraphrase *e.g.*, replacing difficult words with simple ones. For this purpose, we introduce *inclusive pattern* (pattern$_{\in}$), where a pattern of desirable words is provided to the network in the simple form of a regular expression. In Example 3, we ask the network to paraphrase the input in a way that the words *violated* and *ceasefire* appear in the output in this order. Here, asterisks (*) stem for "*zero or more words*", and dollar sign (*$*) for "*end of sentence"*. Likewise, patterns can be asked to start at the beginning of the sentence (not shown in the example).

Input:	*they 've broken the armistice*	(Example 3)
Constraint:	pattern$_{\in}$ = * *violated* * *ceasefire* $	
Output:	*they violated the ceasefire*	

3.5 Pattern of Undesired Words

Alternatively, users may want to discard some words that are in the source sentence but should *not* appear in the output sentence. These constraints are named *exclusive patterns*, denoted as pattern$_{\notin}$. They are modeled and fed into a network in the same way as pattern$_{\in}$.

3.6 Combination of Constraints

Finally, we study the possibility for the user to jointly take into account several types of constraints. As stated in Section 2, all constraints are independently encoded and fed to the decoder as additional input information.

[4] For the experiments, parse trees are provided by the Stanford parser [10].

[5] Let one note that a wider range could be considered by extending the corpus with poor quality paraphrases. However, we decided here to stick to the original training data to propose a consistent experimental pipeline across all the constraints.

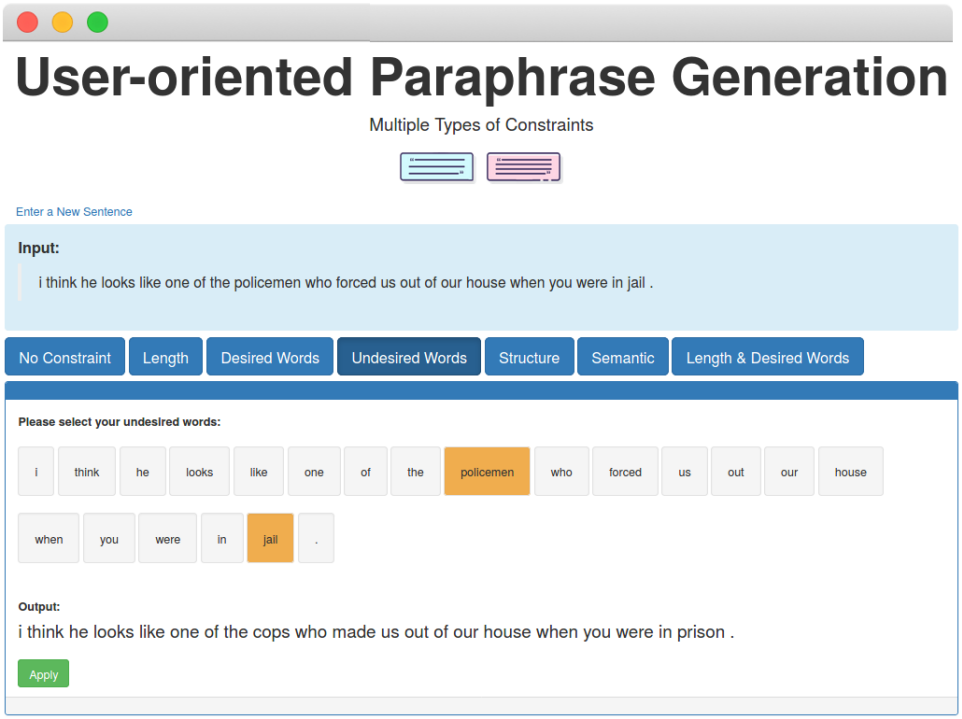


Figure 2: A screenshot of our system. For a given input sentence, the user can switch between different tabs to explore constraints. In this example, the user chooses the 'Undesired Words' tab and selects words *policemen* and *jail* to be excluded from the paraphrase. The output shows that these words are replaced by *cops* and *prison*.

4 The System in Action

Our system takes the input sentence and allows user to explore the effect of each constraint on the paraphrase generation. To begin with, the user is asked to enter an input sentence or randomly choose from a pool of sample sentences. Then, the input sentence is tokenized, padded and passed to the 'No-Constraint' model to simply paraphrase without any constraint.

The user is then directed to the panel where, in addition to viewing the output of the 'No-Constraint' model, he/she can apply desired constraints. As Figure 2 illustrates, this panel includes a tab for each constraint where the user can play with different values of it. For example, in this figure, the user chooses the 'Undesired Words' tab and selects words *policemen* and *jail* to be excluded from the paraphrase. By clicking on the *Apply* button, the constructed pattern$_{\notin}$ constraint is passed to the model along with the input sentence and the result is shown in the output field.

In a similar manner, in the 'Length' tab, the user is given a slider to choose the desired length. Depending on the need to summarize or expand the input, the user can request a shorter or longer paraphrase, respectively. In the 'Desired Words' tab, the user is supposed to choose one or two words from the dictionary and determine whether they should appear in the beginning, in the end or somewhere in the middle of the output from a combo box. Then, based on the desired words and their position, the pattern$_{\in}$ is constructed and passed to the model along with the input sentence. The appearance of the desired words is indicated by a distinct color in the output. In the 'Structure' tab and to generate paraphrases with different syntactical structures, the user can select the desired structure from the 6 most frequent top-level parse trees[6] and modify the syntactical structure of the output. In the 'Semantic' tab, the user can constraint the semantic similarity of the output with the input in terms of BERTScore and in the range of 65% to 100% (See Section 3.3 for more details). User is also able to impose length and pattern$_{\in}$ together in the 'Length & Desired Words' as an example of combined constraints.

[6]Hovering mouse over each tree shows an example as a tooltip.

5 Evaluation Setup

We trained different models separately for each single type of constraints from Section 3 and some combinations of them. To evaluate the performance of our system, we set up the following experiments:

Reference-based Experiments To automatically assess the quality of constrained paraphrase generators, we set up the experiments where the constraints are set such that the system targets the reference. For example, if the length of the reference is 10, the network is asked to generate a paraphrase with length 10 and the output is compared against the reference in terms of automatic measures like BLEU.

Free Constraints Experiments To investigate how the system generalizes in a less controlled environment, we explore a plausible range of constraint values, regardless of the reference. Thereby, for length constraint, we test values from 5 to 40 for each input in the test set. For *BERTScore*, we consider the values in the range of 0.6 to 1.0 with step 0.05. For *pattern*$_{\in}$, we construct maximum 5 patterns by randomly choosing the synonym of input tokens[7]. For *pattern*$_{\notin}$, we construct maximum 5 patterns by randomly choosing one or two input tokens. We extracted the most 10 frequent top-level *parse trees* from the training set and generated paraphrases of each sample with these 10 parse trees. Note that for these experiments, natural language generation (NLG) metrics cannot be computed since no reference exists for each constraint. Only constraint satisfaction can be reported.

5.1 Dataset

All experiments are conducted on the `ParaNMT-50M` corpus [15] such that $49M$, $12K$ and $100K$ paraphrases were used for training, validation and test, respectively.

5.2 Baselines

Naive Systems: Naive constraint-aware systems are built by designing constraint-specific post-processing steps. These steps are directly applied on the output of the *No Constraint* model. In the case of length constraints, post-processing consists in inserting or deleting random tokens until the target length is met. Regarding patterns, the presence (*pattern*$_{\in}$) or absence (*pattern*$_{\notin}$) of the regex constraint is forced by replacing random tokens with pattern ones, or inversely.

Length and Syntactic Baselines We compare with *LenEmb* [6] that proposed to control the length of output via the embedding of the remaining length and also with *LRPE* [16] that proposed to use positional encoding to control the length of output[8]. In addition, we compare with *SCPN* [2] that generates a paraphrase from full parse tree.

5.3 Evaluation Metrics

Constraint Satisfaction Metrics: The following mean absolute error (MAE) metrics are defined to study how well the constraints are effectively satisfied in the outputs:

$$\text{MAE}_{length} = \frac{1}{|N|}\sum_{N}\left|\frac{len(\text{output}) - length_{constraint}}{length_{constraint}}\right|$$

in which $len(x)$ is the number of tokens in sentence x and N is the normalization factor.

$$\text{MAE}_{\text{BERTScore}} = \frac{1}{|N|}\sum_{N}|\,\text{BERTScore(input, output)} - \text{BERTScore}_{constraint}\,|$$

$$\text{MAE}_{\text{pattern}_{\in/\notin}} = 1 - \frac{\text{\# of matched patterns}}{\text{\# of constraint patterns}}$$

MAE_{tree} [17] is computed based on evaluating the tree edit distance between the parse tree of output and the desired parse tree.

[7] We used Word2Vec [13] and WordNet [14] to extract synonyms.

[8] Both of these works proposed to control the length in *character* level, however, ours works in *word* level.

Table 1: Paraphrase quality (BLEU and BS(o,r), the higher the better) and MAEs (the lower the better) for various types of constraints. MAEs associated with the system constraints are in bold. BS(o, r) stands for the BERTScore between the output, *i.e.* the generated paraphrase, and the reference. Models prefixed with $\star$ are based on the proposed CPGN architecture. "All types" includes length, top-2 Parse Tree, BERTScore, pattern$_{\in}$ and pattern$_{\notin}$.

Constraint	**Model**	**NLG metrics** ↑		**M**ean **A**bsolute **E**rror (**MAE**) (%) ↓				
		BLEU	BS(o,r)	Length	Tree	BERTScore	Pattern$_{\in}$	Pattern$_{\notin}$
None	$\star$No-Constraint	30.0	92.2	22.2	12.1	4.8	72.9	52.03
Length	Naive Length	23.2	90.5	**0.0**	14.3	3.2	100.0	84.5
	LenEmb [6]	26.2	91.0	**12.2**	13.7	3.9	94.2	61.7
	LRPE [16]	29.4	91.6	**10.7**	11.6	3.7	59.1	46.3
	$\star$Length	31.6	92.7	$\sim$**0.0**	11.4	3.3	66.3	49.9
Syntax	SCPN [2]	25.9	92.1	23.2	**10.9**	3.4	63.1	59.3
	$\star$Top-2 Parse Tree	30.7	92.7	18.7	**10.8**	4.7	60.0	48.6
	$\star$Full Parse Tree	52.9	95.5	1.7	**2.5**	3.8	38.8	33.4
Semantics	$\star$BERTScore	31.0	93.2	20.4	11.5	**3.2**	64.2	48.9
Pattern$_{\in}$	Naive Pattern$_{\in}$	25.8	91.1	22.9	16.9	4.1	**0.0**	0.5
	$\star$Pattern$_{\in}$	44.7	94.8	15.5	9.3	3.2	**7.4**	35.2
Pattern$_{\notin}$	Naive Pattern$_{\notin}$	21.7	90.8	23.9	7.9	5.2	79.5	**0.0**
	$\star$Pattern$_{\notin}$	34.7	93.1	18.0	10.8	4.4	61.5	**41.2**
Multiple	$\star$Length + BERTScore	31.9	93.9	**<0.1**	11.1	**2.6**	62.5	46.8
	$\star$Length + Top-2 ParseTree	33.0	93.2	$\sim$**0.0**	**8.3**	2.6	57.1	42.9
	$\star$Length + Pattern$_{\in}$	46.6	95.2	$\sim$**0.0**	8.7	3.8	**14.0**	34.8
	$\star$BERTScore + Pattern$_{\in}$	45.6	95.1	13.0	8.8	**4.0**	**21.8**	33.5
	$\star$All types	47.2	94.8	**2.7**	**8.8**	**3.8**	**11.3**	**26.6**

Paraphrase Quality Metrics: We rely on the standard automatic NLG evaluation metric BLEU [18]. BERTScore values between the output and reference are also reported to evaluate their semantic similarity[9].

6 Evaluation Results

As detailed in Table 1, this section reports paraphrase quality and constraint satisfaction when targeting a given reference.

6.1 Constraint Satisfaction

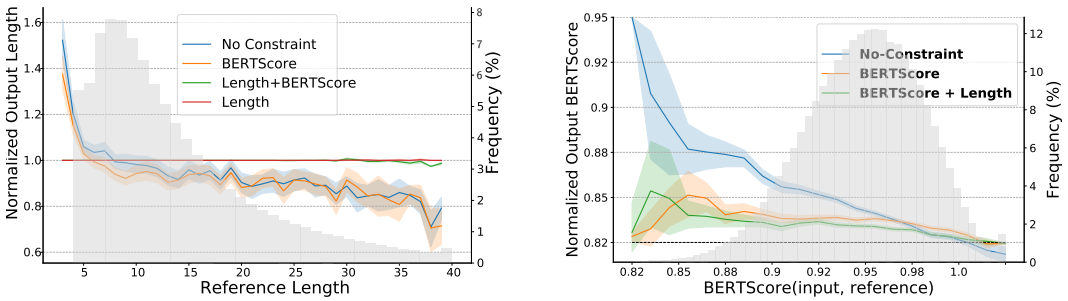
The following comments on the main results w.r.t. constraints satisfaction (MAE columns).

Length is precisely followed. The minimum MAE$_{\text{length}}$ is obtained when *length* is applied as a constraint. Figure 3a provides additional details regarding the requested (reference) length. The *y-axis* shows the length of the produced paraphrase normalized by the requested length. It appears that the model properly learns to generate the desired length even for long sentences.

The more structural information, the better the network follows it. To investigate syntactic constraint satisfaction, full and top-2 level parse trees from the reference are compared. A significant improvement in MAE$_{\text{tree}}$ is reported when the full parse tree is provided. Considering the top-2 levels is much less efficient, but still outperforming the no-constraint baseline. This is because the top-2 levels are too generic to describe the desired changes.

Semantics can be controlled by BERTScore representations. Results in Table 1 show that the highest semantic similarities with the reference are for the models based on the BERTScore constraints. Likewise, it is interesting to highlight that BERTScore values come from a very limited range of values, as shown in Figure 3b.

[9] A similar trend was observed for the results in terms of METEOR, ROUGE-L and GSM which we do not report.



(a) Length constraint satisfaction.

(b) BERTScore constraint satisfaction.

Figure 3: Detailed constraint satisfaction for length (a) and BERTScore (b). Lines show the ratio between requested and generated constraints, 1 meaning that they exactly match. Gray histograms are the distribution of constraints in the training set. Details in Section 6.1.

CPGNs are able to include or remove patterns. The results show the success of generated sentences in following the patterns. It also reveals that inclusive patterns are more informative than exclusive patterns since they provide information about new tokens and their order.

Combination of constraints is challenging for the network. Finally, Table 1 reports that CPGNs are able to combine constraints to achieve better results which come from *cooperation* or *compromise*. For instance, in 'Length + BERTScore' model, $\text{MAE}_{\text{BERTScore}}$ improves from 3.19 to 2.58 while $\text{MAE}_{\text{length}}$ remains almost unchanged. In this case, the information about the target length guides the system to follow the semantic constraint. The compromise case occurs when each isolated constraint has already achieved very good results on their respective domains. For instance, in 'Length + $\text{Pattern}_{\in}$' model, $\text{MAE}_{\text{pattern}_{\in}}$ is worse than the $\text{pattern}_{\in}$ only model. In parallel, $\text{MAE}_{\text{pattern}_{\in}}$ is better than the length only model. Hence, the system selects a compromise between the two constraints.

Results of Free Constraints Experiments To complement the analysis of CPGNs, we study what is happening when free constraints—which are *not* derived from the target reference—are explored (See Section 5 for more details.). Results in Table 2 show that scores stay close to the MAE values reported in Table 1 but a little bit worse. Thereby, CPGNs are still able to follow the constraints even with a larger range of values, including when constraints are combined.

6.2 Paraphrase Quality

This section comments on Table 1 w.r.t. BLEU and BERTScore between output and reference to prove that CPGNs do not blindly apply the constraint, but indeed produce good paraphrases.

Structural constraints, *i.e.* length and parse tree, improve the quality of paraphrases. Models with length constraint outperform the no-constraint model and the length-aware naive baseline on all metrics. It proves that providing the length constraint is not just a simple template for the network: it learns how to properly use it to generate better paraphrases.

BERTScore constraints enhance semantic quality as well. About semantics, the quality of the generated paraphrase is improved when considering the BERTScore constraint as the $BS(o, i)$ increases in comparison to the No-constraint baseline. However, targeting a given BERTScore is difficult as words are discrete and changing one word inevitably changes the meaning.

Following patterns drastically improves the quality of paraphrases. As CPGNs are successful in following the patterns, significant improvements are also reported in the quality of paraphrases. The poor results of the naive approaches show that this improvement does not only come from the valuable information of patterns, but also from extra knowledge learned and properly used by the network. Applying $\text{pattern}_{\notin}$ results in less improvements because giving information about which words should be removed does not help to find the same substitution as in the reference.

Table 2: Constraint satisfaction as MAEs (%, the lower the better) for single or double constraints.

Constraints (c_1 and c_2)	**MAE**$_{c_1}$	**MAE**$_{c_2}$
Length	0.1	-
Top-2 ParseTree	1.2	-
BERTScore	8.3	-
Pattern$_{\in}$	20.9	-
Pattern$_{\notin}$	30.9	-
Length + BERTScore	0.4	8.6
Length + Pattern$_{\in}$	0.1	20.9
BERTScore + Pattern$_{\in}$	11.2	18.6
Length + Top-2 ParseTree	0.8	1.9

Table 3: Results of the human evaluation (✗ = no paraphrase, $\sim$ = ungrammatical paraphrase, ✓ = grammatical paraphrase).

Model	✗	$\sim$	✓
No Constraint	0.64	0.17	0.19
Free Constraint CPGNs			
Length	0.67	0.24	0.09
Top-2 Parse Tree	0.45	0.30	0.25
BERTScore	0.52	0.14	0.34
Pattern$_{\in}$	0.71	0.09	0.20
Pattern$_{\notin}$	0.69	0.10	0.21
Average	0.61	0.17	0.22

Combining constraints brings synergy. Among different combinations of constraints, we provided the results of some that we find interesting from an application viewpoint. As shown in the last rows of Table 1, combining two types of constraints is always beneficial in terms of paraphrase quality, w.r.t. using one of each two in isolation.

Paraphrase Quality in Free Constraints Experiments Due to the lack of reference for each constraint value in free constraint experiments, NLG metrics cannot be used. For this reason, to measure the quality of free constraint paraphrases, a human evaluation is performed in which workers are asked to rate a paraphrase pair $< x, y >$ on the three-point scale from [19], where x is the source sentence and y is the generated sentence. A value 2 in this scale indicates that y is a grammatical paraphrase of x (i.e., meaning preservation and grammar are OK), while 1 means y is an ungrammatical paraphrase of x (only meaning preservation is OK) and 0 means no paraphrase relationship (none of them is OK). Table 3 shows that applying free constraints leads to acceptable paraphrases in terms of quality, mostly in line with scores of the "no-constraint" model. Deeper analysis reveals that syntax-related constraints, i.e., length and parse tree, spoil the grammar more than the others. Second, the model mostly applies the patterns (inclusive or exclusive) with correct grammar. Finally, paraphrasing with the BERTScore constraint is the most challenging of all.

7 Conclusion and Future Work

We developed a constraint-sensitive paraphrase generation system to integrate various types of constraints which thus provide control on the desired output. The results show that this system is effectively able to take into account these constraints—although this is with variable performance according to the type of constraints—and that this can help to produce better paraphrases.

References

[1] Aurélien Max. Local rephrasing suggestions for supporing the work of writers. In Bengt Nordström and Aarne Ranta, editors, *Proceedings of the Advances in Natural Language Processing*, pages 324–335, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[2] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1875–1885, 2018.

[3] Tomoyuki Kajiwara. Negative lexically constrained decoding for paraphrase generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6047–6052, 2019.

[4] Cedric Boidin, Verena Rieser, Lonneke van der Plas, Oliver Lemon, and Jonathan Chevelu. Predicting how it sounds: Re-ranking dialogue prompts based on tts quality for adaptive spoken dialogue systems. In *Proceedings of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2009.

[5] Kalpesh Krishna, John Wieting, and Mohit Iyyer. Reformulating unsupervised style transfer as paraphrase generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 737–762, 2020.

[6] Yuta Kikuchi, Graham Neubig, Ryohei Sasano, Hiroya Takamura, and Manabu Okumura. Controlling output length in neural encoder-decoders. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1328–1338, 2016.

[7] Ashutosh Kumar, Kabir Ahuja, Raghuram Vadapalli, and Partha Talukdar. Syntax-guided controlled generation of paraphrases. *Transactions of the Association for Computational Linguistics*, 8:330–345, 2020.

[8] Tanya Goyal and Greg Durrett. Neural syntactic preordering for controlled paraphrase generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[9] Elozino Egonmwan and Yllias Chali. Transformer and seq2seq model for paraphrase generation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 249–255, 2019.

[10] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL): system demonstrations*, pages 55–60, 2014.

[11] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[14] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[15] John Wieting and Kevin Gimpel. Paranmt-50m: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 451–462, 2018.

[16] Sho Takase and Naoaki Okazaki. Positional encoding to control output sequence length. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3999–4004, 2019.

[17] Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. Controllable paraphrase generation with a syntactic exemplar. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5972–5984, 2019.

[18] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[19] Stanley Kok and Chris Brockett. Hitting the right paraphrases in good time. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 145–153, 2010.