Generating Syntactically Controlled Paraphrases without Using Annotated Parallel Pairs

Kuan-Hao Huang
University of California, Los Angeles
khhuang@cs.ucla.edu

Kai-Wei Chang
University of California, Los Angeles
kwchang@cs.ucla.edu

Abstract

Paraphrase generation plays an essential role in natural language process (NLP), and it has many downstream applications. However, training supervised paraphrase models requires many annotated paraphrase pairs, which are usually costly to obtain. On the other hand, the paraphrases generated by existing unsupervised approaches are usually syntactically similar to the source sentences and are limited in diversity. In this paper, we demonstrate that it is possible to generate syntactically various paraphrases without the need for annotated paraphrase pairs. We propose *Syntactically controlled Paraphrase Generator* (SynPG), an encoder-decoder based model that learns to disentangle the semantics and the syntax of a sentence from a collection of unannotated texts. The disentanglement enables SynPG to control the syntax of output paraphrases by manipulating the embedding in the syntactic space. Extensive experiments using automatic metrics and human evaluation show that SynPG performs better syntactic control than unsupervised baselines, while the quality of the generated paraphrases is competitive. We also demonstrate that the performance of SynPG is competitive or even better than supervised models when the unannotated data is large. Finally, we show that the syntactically controlled paraphrases generated by SynPG can be utilized for data augmentation to improve the robustness of NLP models.

1 Introduction

Paraphrase generation (McKeown, 1983) is a long-lasting task in natural language processing (NLP) and has been greatly improved by recently developed machine learning approaches and large data collections. Paraphrase generation demonstrates the potential of machines in semantic abstraction and sentence reorganization and has already been applied to many NLP downstream applications,

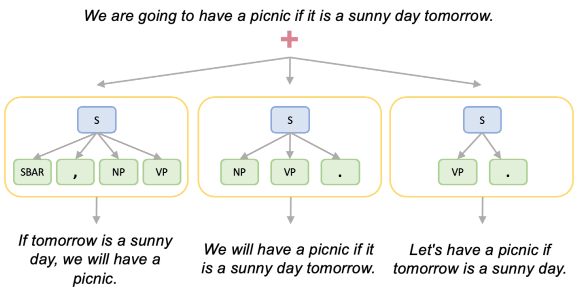

We are going to have a picnic if it is a sunny day tomorrow.
S
SBAR
,
NP
VP
S
NP
VP
.
S
VP
.
If tomorrow is a sunny day, we will have a picnic.
We will have a picnic if it is a sunny day tomorrow.
Let's have a picnic if tomorrow is a sunny day.

Figure 1: Paraphrase generation with syntactic control. Given a source sentence and a target syntactic specification (either a full parse tree or top levels of a parse tree), the model is expected to generate a paraphrase with the syntax following the given specification.

such as question answering (Yu et al., 2018), chatbot engines (Yan et al., 2016), and sentence simplification (Zhao et al., 2018).

In recent years, various approaches have been proposed to train sequence-to-sequence (seq2seq) models on a large number of annotated paraphrase pairs (Prakash et al., 2016; Mallinson et al., 2017; Cao et al., 2017; Egonmwan and Chali, 2019). Some of them control the syntax of output sentences to improve the diversity of paraphrase generation (Iyyer et al., 2018; Goyal and Durrett, 2020; Kumar et al., 2020). However, collecting annotated pairs is expensive and induces challenges for some languages and domains. On the contrary, unsupervised approaches build paraphrase models without using parallel corpora (Li et al., 2018; Roy and Grangier, 2019; Zhang et al., 2019). Most of them are based on the variational autoencoder (Bowman et al., 2016) or back-translation (Mallinson et al., 2017; Wieting and Gimpel, 2018; Hu et al., 2019). Nevertheless, without the consideration of controlling syntax, their generated paraphrases are often similar to the source sentences and are not diverse in syntax.

This paper presents a pioneering study on syntactically controlled paraphrase generation based on disentangling semantics and syntax. We aim to disentangle one sentence into two parts: 1) the semantic part and 2) the syntactic part. The semantic aspect focuses on the meaning of the sentence, while the syntactic part represents the grammatical structure. When two sentences are paraphrased, their semantic aspects are supposed to be similar, while their syntactic parts should be different. To generate a syntactically different paraphrase of one sentence, we can keep its semantic part unchanged and modify its syntactic part.

Based on this idea, we propose **Syn**tactically Controlled **P**araphrase **G**enerator (SynPG)[1], a Transformer-based model (Vaswani et al., 2017) that can generate syntactically different paraphrases of one source sentence based on some target syntactic parses. SynPG consists of a semantic encoder, a syntactic encoder, and a decoder. The semantic encoder considers the source sentence as a bag of words without ordering and learns a contextualized embedding containing only the semantic information. The syntactic encoder embeds the target parse into a contextualized embedding including only the syntactic information. Then, the decoder combines the two representations and generates a paraphrase sentence. The design of disentangling semantics and syntax enables SynPG to learn the association between words and parses and be trained by reconstructing the source sentence given its unordered words and its parse. Therefore, we do not require any annotated paraphrase pairs but only unannotated texts to train SynPG.

We verify SynPG on four paraphrase datasets: ParaNMT-50M (Wieting and Gimpel, 2018), Quora (Iyer et al., 2017), PAN (Madnani et al., 2012), and MRPC (Dolan et al., 2004). The experimental results reveal that when being provided with the syntactic structures of the target sentences, SynPG can generate paraphrases with the syntax more similar to the ground truth than the unsupervised baselines. The human evaluation results indicate that SynPG achieves competitive paraphrase quality to other baselines while its generated paraphrases are more accurate in following the syntactic specifications. In addition, we show that when the training data is large enough, the performance of SynPG is competitive or even better than supervised approaches. Finally, we demonstrate that the syntactically controlled paraphrases generated by SynPG can be used for data augmentation to defense syntactically adversarial attack (Iyyer et al., 2018) and improve the robustness of NLP models.

[1]Our code and the pretrained models are available at `https://github.com/uclanlp/synpg`

2 Unsupervised Paraphrase Generation

We aim to train a paraphrase model without using annotated paraphrase pairs. Given a source sentence $\mathbf{x} = (x_1, x_2, ..., x_n)$, our goal is to generate a paraphrase sentence $\mathbf{y} = (y_1, y_2, ..., y_m)$ that is expected to maintain the same meaning of $\mathbf{x}$ but has a different syntactic structure from $\mathbf{x}$.

Syntactic control. Motivated by previous work (Iyyer et al., 2018; Zhang et al., 2019; Kumar et al., 2020), we allow our model to access additional syntactic specifications as the control signals to guide the paraphrase generation. More specifically, in addition to the source sentence $\mathbf{x}$, we give the model a target constituency parse $\mathbf{p}$ as another input. Given the input $(\mathbf{x}, \mathbf{p})$, the model is expected to generate a paraphrase $\mathbf{y}$ that is semantically similar to the source sentence $\mathbf{x}$ and syntactically follows the target parse $\mathbf{p}$. In the following discussions, we assume the target parse $\mathbf{p}$ to be a full constituency parse tree. Later on, in Section 2.3, we will relax the syntax guidance to be a *template*, which is defined as the top two levels of a full parse tree. We expect that a successful model can control the syntax of output sentences and generate syntactically different paraphrases based on different target parses, as illustrated in Figure 1.

Similar to previous work (Iyyer et al., 2018; Zhang et al., 2019), we linearize the constituency parse tree to a sequence. For example, the linearized parse of the sentence "*He eats apples.*" is `(S(NP(PRP))(VP(VBZ)(NP(NNS)))(.))`. Accordingly, a parse tree can be considered as a sentence $\mathbf{p} = (p_1, p_2, ..., p_k)$, where the tokens in $\mathbf{p}$ are non-terminal symbols and parentheses.

2.1 Proposed Model

Our main idea is to disentangle a sentence into the semantic part and the syntactic part. Once the model learns the disentanglement, it can generate a syntactically different paraphrase of one given sentence by keeping its semantic part unchanged and modifying only the syntactic part.

Figure 2 illustrates the proposed paraphrase model called SynPG, a seq2seq model consisting of a semantic encoder, a syntactic encoder, and a decoder. The semantic encoder captures only the semantic information of the source sentence $\mathbf{x}$,

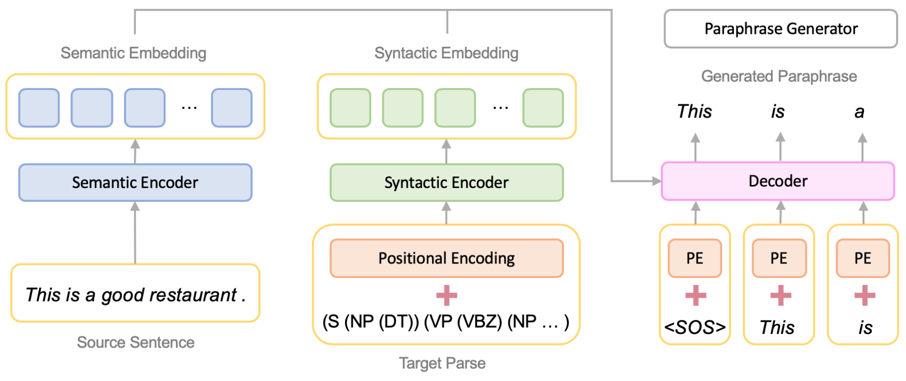


Figure 2: SynPG embeds the source sentence and the target parse into a semantic embedding and a syntactic embedding, respectively. Then, SynPG generates a paraphrase sentence based on the two embeddings.

while the syntactic encoder extracts only the syntactic information from the target parse $\mathbf{p}$. The decoder then combines the encoded semantic and syntactic information and generates a paraphrase $\mathbf{y}$. We discuss the details of SynPG in the following.

Semantic encoder. The semantic encoder embeds a source sentence $\mathbf{x}$ into a contextualized semantic embedding $\mathbf{z}_{sem}$. In other words,

$$\mathbf{z}_{sem} = (z_1, z_2, ..., z_n) = \text{Enc}_{sem}((x_1, x_2, .., x_n)).$$

The semantic embedding $\mathbf{z}_{sem}$ is supposed to contain only the semantic information of the source sentence $\mathbf{x}$. To separate the semantic information from the syntactic information, we use a Transformer (Vaswani et al., 2017) *without the positional encoding* as the semantic encoder. We posit that by removing position information from the source sentence $\mathbf{x}$, the semantic embedding $\mathbf{z}_{sem}$ would encode less syntactic information.

We assume that words without ordering capture most of the semantics of one sentence. Indeed, semantics is also related to the order. For example, exchanging the subject and the object of a sentence changes its meaning. However, the decoder trained on a large corpus also captures the *selectional preferences* (Katz and Fodor, 1963; Wilks, 1975) in generation, which enables the decoder to infer the proper order of words. In addition, we observe that when two sentences are paraphrased, they usually share similar words, especially those words related to the semantics. For example, "*What is the best way to improve writing skills?*" and "*How can I improve my writing skills?*" are paraphrased, and the shared words (*improve*, *writing*, and *skills*) are strongly related to the semantics. In Section 4, we show that our designed semantic embedding captures enough semantic information to generate paraphrases.

Syntactic encoder. The syntactic encoder embeds the target parse $\mathbf{p} = (p_1, p_2, ..., p_k)$ into a contextualized syntactic embedding $\mathbf{z}_{syn}$. That is,

$$\mathbf{z}_{syn} = (z_1, z_2, ..., z_k) = \text{Enc}_{syn}((p_1, p_2, .., p_k)).$$

Since the target parse $\mathbf{p}$ contains no semantic information but only syntactic information, we use a Transformer *with the positional encoding* as the syntactic encoder.

Decoder. Finally, we design a decoder that takes the semantic embedding $\mathbf{z}_{sem}$ and the syntactic embedding $\mathbf{z}_{syn}$ as the input and generates a paraphrase $\mathbf{y}$. In other words,

$$\mathbf{y} = (y_1, y_2, ..., y_m) = \text{Dec}(\mathbf{z}_{sem}, \mathbf{z}_{syn}).$$

We choose Transformer as the decoder to generate $\mathbf{y}$ autoregressively. Notice that the semantic embedding $\mathbf{z}_{sem}$ does not encode the position information and the syntactic embedding $\mathbf{z}_{syn}$ does not contain semantics. This forces the decoder to extract the semantics from $\mathbf{z}_{sem}$ and retrieve the syntactic structure from $\mathbf{z}_{syn}$. The attention weights attaching to $\mathbf{z}_{sem}$ and $\mathbf{z}_{syn}$ make the decoder learn the association between the semantics and the syntax as well as the relation between the word order and the parse structures. Therefore, SynPG is able to reorganize the source sentence and use the given syntactic structure to rephrase the source sentence.

2.2 Unsupervised Training

Our design of the disentanglement makes it possible to train SynPG without using annotated pairs. We train SynPG with the objective to reconstruct the source sentences. More specifically, when training on a sentence $\mathbf{x}$, we first separate $\mathbf{x}$ into two parts: 1) an unordered word list $\bar{\mathbf{x}}$ and 2) its linearized parse $\mathbf{p}_x$ (can be obtained by a pretrained parser). Then, SynPG is trained to reconstruct $\mathbf{x}$ from $(\bar{\mathbf{x}}, \mathbf{p}_x)$ with the reconstruction loss

$$\mathcal{L} = -\sum_{i=1}^{n} \log P(y_i = x_i | \bar{\mathbf{x}}, \mathbf{p}_x, \mathbf{y}_1, ..., \mathbf{y}_{i-1}).$$

Notice that if we do not disentangle the semantics and the syntax, and directly use a seq2seq model to reconstruct $\mathbf{x}$ from $(\mathbf{x}, \mathbf{p}_x)$, it is likely that the seq2seq model only learns to copy $\mathbf{x}$ and ignores $\mathbf{p}_x$ since $\mathbf{x}$ contains all the necessary information for the reconstruction. Consequently, at inference time, no matter what target parse $\mathbf{p}$ is given, the seq2seq model always copies the whole source sentence $\mathbf{x}$ as the output (more discussion in Section 4).

On the contrary, SynPG learns the disentangled embeddings $\mathbf{z}_{sem}$ and $\mathbf{z}_{syn}$. This makes SynPG capture the relation between the semantics and the syntax to reconstruct the source sentence $\mathbf{x}$. Therefore, at test time, given the source sentence $\mathbf{x}$ and a new target parse $\mathbf{p}$, SynPG is able to apply the learned relation to rephrase the source sentence $\mathbf{x}$ according to the target parse $\mathbf{p}$.

Word dropout. We observe that the ground truth paraphrase may contain some words not appearing in the source sentence; however, the paraphrases generated by the vanilla SynPG tend to include only words appearing in the source sentence due to the reconstruction training objective. To encourage SynPG to improve the diversity of the word choices in the generated paraphrases, we randomly discard some words from the source sentence during training. More precisely, each word has a probability to be dropped out in each training iteration. Accordingly, SynPG has to predict the missing words during the reconstruction, and this enables SynPG to select different words from the source sentence to generate paraphrases. More details are discussed in Section 4.5.

2.3 Templates and Parse Generator

In the previous discussion, we assume that a full target constituency parse tree is provided as the input to SynPG. However, the full parse tree of the target paraphrase sentence is unlikely available at inference time. Therefore, following the setting in Iyyer et al. (2018), we consider generating the paraphrase based on the *template*, which is defined as the top two levels of the full constituency parse tree. For example, the template of `(S(NP(PRP))(VP(VBZ)(NP(NNS)))(.))` is `(S(NP)(VP))(.))`.

Motivated by Iyyer et al. (2018), we train a parse generator to generate full parses from templates. The proposed parse generator has the same architecture as SynPG, but the input and the output are different. The parse generator takes two inputs: a tag sequence $\mathbf{tag}_x$ and a target template $\mathbf{t}$. The tag sequence $\mathbf{tag}_x$ contains all the POS tags of the source sentence $\mathbf{x}$. For example, the tag sequence of the sentence "*He eats apples.*" is "`<PRP> <VBZ> <NNS> <.>`". Similar to the source sentence in SynPG, we do not consider the word order of the tag sequence during encoding. The expected output of the parse generator is a full parse $\tilde{\mathbf{p}}$ whose a syntactic structure follows the target template $\mathbf{t}$.

We train the parse generator without any additional annotations as well. Let $\mathbf{t}_x$ be the the template of $\mathbf{p}_x$ (the parse of $\mathbf{x}$), we end-to-end train the parse generator with the input being $(\mathbf{tag}_x, \mathbf{t}_x)$ and the output being $\mathbf{p}_x$.

Generating paraphrases from templates. The parse generator makes us generate paraphrases by providing target templates instead of target parses. The steps to generate a paraphrase given a source sentence $\mathbf{x}$ and a target template $\mathbf{t}$ are as follows:

1. Get the tag sequence $\mathbf{tag}_x$ of the source sentence $\mathbf{x}$.
2. Use the parse generator to generate a full parse $\tilde{\mathbf{p}}$ with input $(\mathbf{tag}_x, \mathbf{t})$.
3. Use SynPG to generate a paraphrase $\mathbf{y}$ with input $(\mathbf{x}, \tilde{\mathbf{p}})$.

Post-processing. We notice that certain templates are not suitable for some source sentences and therefore the generated paraphrases are nonsensical. We follow Iyyer et al. (2018) and use n-gram overlap and paraphrastic similarity computed by the model[2] from Wieting and Gimpel (2018) to remove nonsensical paraphrases[3].

3 Experimental Settings

We conduct extensive experiments to demonstrate that SynPG performs better syntactic control than other unsupervised paraphrase models, while the quality of generated paraphrases by SynPG is comparable to others. In addition, we show that the performance of SynPG is competitive or even better than supervised models when the training data is large enough.

[2]https://github.com/jwieting/para-nmt-50m

[3]We set the minimum n-gram overlap to 0.3 and the minimum paraphrastic similarity to 0.7.

3.1 Datasets

For the training data, we consider ParaNMT-50M (Wieting and Gimpel, 2018), a paraphrase dataset containing over 50 million pairs of reference sentences and the corresponding paraphrases as well as the quality scores. We select about 21 million pairs with higher quality scores as our training examples. Notice that we use *only the reference sentences* to train SynPG and unsupervised paraphrase models since we do not require paraphrase pairs.

We sample 6,400 pairs from ParaNMT-50M as the testing data. To evaluate the transferability of SynPG, we also consider the other three datasets: 1) Quora (Iyer et al., 2017) contains over 400,000 paraphrase pairs and we sample 6,400 pairs from them. 2) PAN (Madnani et al., 2012) contains 5,000 paraphrase pairs. 3) MRPC (Dolan et al., 2004) contains 2,753 paraphrase pairs.

3.2 Evaluation

We consider paraphrase pairs to evaluate all the models. For each test paraphrase pair $(\mathbf{x}_1, \mathbf{x}_2)$, we consider $\mathbf{x}_1$ as the source sentence and treat $\mathbf{x}_2$ as the target sentence (ground truth). Let $\mathbf{p}_2$ be the parse of $\mathbf{x}_2$, given $(\mathbf{x}_1, \mathbf{p}_2)$, The model is expected to generate a paraphrase $\mathbf{y}$ that is similar to the target sentence $\mathbf{x}_2$.

We use BLEU score (Papineni et al., 2002) and human evaluation to measure the similarity between $\mathbf{x}_2$ and $\mathbf{y}$. Moreover, to evaluate how well the generated paraphrase $\mathbf{y}$ follows the target parse $\mathbf{p}_2$, we define the *template matching accuracy* (TMA) as follows. For each ground truth sentence $\mathbf{x}_2$ and the corresponding generated paraphrase $\mathbf{y}$, we get their parses ($\mathbf{p}_2$ and $\mathbf{p}_y$) and templates ($\mathbf{t}_2$ and $\mathbf{t}_y$). Then, we calculate the percentage of pairs whose $\mathbf{t}_y$ *exactly matches* $\mathbf{t}_2$ as the *template matching accuracy*.

3.3 Models for Comparison

We consider the following unsupervised paraphrase models: 1) **CopyInput**: a naïve baseline which directly copies the source sentence as the output without paraphrasing. 2) **BackTrans**: back-translation is proposed to generate paraphrases (Mallinson et al., 2017; Wieting and Gimpel, 2018; Hu et al., 2019). In our experiment, we use the pretrained EN-DE and DE-EN translation models[4] proposed by Ng et al. (2019) to conduct back-translation. Notice that training translation models requires additional translation pairs. Therefore, BackTrans needs more resources than ours and the translation data may not available for some low-resource languages. 3) **VAE**: we consider a vanilla variational autoencoder (Bowman et al., 2016) as a simple baseline. 4) **SIVAE**: syntax-infused variational autoencoder (Zhang et al., 2019) utilizes additional syntax information to improve the quality of sentence generation and paraphrase generation. Unlike SynPG, SIVAE does not disentangle the semantics and syntax. 5) **Seq2seq-Syn**: we train a seq2seq model with Transformer architecture to reconstruct $\mathbf{x}$ from $(\mathbf{x}, \mathbf{p}_x)$ without the disentanglement. We use this model to study the influence of the disentanglement. 6) **SynPG**: our proposed model which learns disentangled embeddings.

We also compare SynPG with supervised approaches. We consider the following: 1) **Seq2seq-Sup**: a seq2seq model with Transformer architecture trained on whole ParaNMT-50M pairs. 2) **SCPN**: syntactically controlled paraphrase network (Iyyer et al., 2018) is a supervised paraphrase model with syntactic control trained on ParaNMT-50M pairs. We use their pretrained model[5].

3.4 Implementation Details

We consider byte pair encoding (Sennrich et al., 2016) for tokenization and use Stanford CoreNLP parser (Manning et al., 2014) to get constituency parses. We set the max length of sentences to 40 and set the max length of linearized parses to 160 for all the models. For the encoders and the decoder of SynPG, we use the standard Transformer (Vaswani et al., 2017) with default parameters. The word embedding is initialized by GloVe (Pennington et al., 2014). We use Adam optimizer with the learning rate being 10^{-4} and the weight decay being 10^{-5}. We set the word dropout probability to 0.4 (more discussion in Section 4.5). The number of epoch for training is set to 5.

Seq2seq-Syn, Seq2seq-Sup are trained with the similar setting. We reimplemnt VAE and SIVAE, and all the parameters are set to the default value in the original papers.

4 Results and Discussion

4.1 Syntactic Control

We first discuss if the syntactic specification enables SynPG to control the output syntax better.

[4]https://github.com/pytorch/fairseq/tree/master/examples/wmt19

[5]https://github.com/miyyer/scpn

	Model	ParaNMT		Quora		PAN		MRPC	
		TMA	BLEU	TMA	BLEU	TMA	BLEU	TMA	BLEU
No Paraphrasing	CopyInput	33.6	16.4	55.0	20.0	37.3	26.8	47.9	30.7
Unsupervised Models	BackTrans	29.0	16.3	53.0	16.4	27.9	16.2	47.2	21.6
	VAE	26.3	9.6	44.0	8.1	19.4	5.2	20.8	1.2
With Syntactic Specifications	SIVAE	30.0	12.8	48.3	13.1	26.6	11.8	21.5	5.1
	Seq2seq-Syn	33.5	16.3	54.9	19.8	37.1	**26.5**	47.7	**30.4**
	SynPG	**71.0**	**32.2**	**82.6**	**33.2**	**66.3**	26.4	**74.0**	26.2

Table 1: Paraphrase results on four datasets. TMA denotes the template matching accuracy, which evaluates how often the generated paraphrases follow the target parses. With the syntactic control, SynPG obtains higher BLEU score and the template matching accuracy. This implies the paraphrases generated by SynPG are more similar to the ground truths and follow the target parses more accurately.

Model	Example 1 (ParaNMT)	Example 2 (Quora)
Source Sent.	these children are gonna die if we don't act now.	what are the best ways to improve writing skills?
Ground Truth	if we don't act quickly, the children will die.	how could i improve my writing skill?
BackTrans	these children will die if we do not act now.	what are the best ways to improve your writing skills?
VAE	these children are gonna die if we don't act now.	what are the best ways to improve writing skills?
SIVAE	these children are gonna die if we don't act now .	what are the best ways to improve writing skills?
Seq2seq-Syn	these children are gonna die if we don't act now.	what are the best ways to improve writing skills?
SynPG	if we don't act now, these children will die.	how can i improve my writing skills?

Table 2: Paraphrases generated by each model. SynPG can generate paraphrases with the syntax more similar to the ground truth than other baselines.

Table 1 shows the template matching accuracy and BLEU score for SynPG and the unsupervised baselines. Notice that here we use the full parse trees as the syntactic specifications. We will discuss the influence of using the template as the syntactic specifications in Section 4.3.

Although we train SynPG on the reference sentences of ParaNMT-50M, we observe that SynPG performs well on Quora, PAN, and MRPC as well. This validates that SynPG indeed learns the syntactic rules and can transfer the learned knowledge to other datasets. CopyInput gets high BLEU scores; however, due to the lack of paraphrasing, it obtains low template matching scores. Compared to the unsupervised baselines, SynPG achieves higher template matching accuracy and higher BLEU scores on all datasets. This verifies that the syntactic specification is indeed helpful for syntactic control.

Next, we compare SynPG with Seq2seq-Syn and SIVAE. All models are given syntactic specifications; however, without the disentanglement, Seq2seq-Syn and SIVAE tend to copy the source sentence as the output and therefore get low template matching scores.

Table 2 lists some paraphrase examples generated by all models. Again, we observe that without syntactic specifications, the paraphrases generated by unsupervised baselines are similar to the source sentences. Without the disentanglement, Seq2seq-Syn and SIVAE always copy the source sentences. SynPG is the only model can generate paraphrases syntactically similar to the ground truths.

4.2 Human Evaluation

We perform human evaluation using Amazon Mechanical Turk to evaluate the quality of generated paraphrases. We follow the setting of previous work (Kok and Brockett, 2010; Iyyer et al., 2018; Goyal and Durrett, 2020). For each model, we randomly select 100 pairs of source sentence $\mathbf{x}$ and the corresponding generated paraphrase $\mathbf{y}$ from ParaNMT-50M test set (after being post-processed as mentioned in Section 2.3) and have three Turkers annotate each pair. The annotations are on a three-point scale: **0** means $\mathbf{y}$ is not a paraphrase of $\mathbf{x}$; **1** means $\mathbf{x}$ is paraphrased into $\mathbf{y}$ but $\mathbf{y}$ contains some grammatical errors; **2** means $\mathbf{x}$ is paraphrased into $\mathbf{y}$, which is grammatically correct.

The results of human evaluation are reported in Table 3. If paraphrases rated **1** or **2** are considered meaningful, we notice that SynPG generates meaningful paraphrases at a similar frequency to that of SIVAE. However, SynPG tends to generate more ungrammatical paraphrases (those rated **1**). We think the reason is that most of paraphrases generated by SIVAE are very similar to the source sentences, which are usually grammatically correct. On the other hand, SynPG is encouraged to

Model	**2**	**1**	**0**	**2+1**	Hit Rate
BackTrans	63.6	22.4	14.0	86.0	11.0
SIVAE	57.6	20.3	22.0	78.0	6.5
SynPG	44.3	32.0	23.7	76.3	28.9

Table 3: Human evaluation on a three-point scale (**0** = not a paraphrase, **1** = ungrammatical paraphrase, **2** = grammatical paraphrase). SynPG performs better on hit rate (defined as the percentage of generated paraphrase getting **2** and matching the target parse at the same time) than other unsupervised models.

use different syntactic structures from the source sentences to generate paraphrases, which may lead some grammatical errors.

Furthermore, we calculate the *hit rate*, the percentage of generated paraphrases getting **2** and matching the target parse at the same time. The hit rate measures how often the generated paraphrases follow the target parses and preserve the semantics (verified by human evaluation) simultaneously. The results show that SynPG gets higher hit rate than other models.

4.3 Target Parses vs. Target Templates

Next, we discuss the influence of generating paraphrase by using templates instead of using full parse trees. For each paraphrase pair $(\mathbf{x}_1, \mathbf{x}_2)$ in test data, we consider two ways to generate the paraphrase. 1) Generating the paraphrase with the target parse. We use SynPG to generate a paraphrase directly from $(\mathbf{x}_1, \mathbf{p}_2)$. 2) Generating the paraphrase with the target template. We first use the parse generator to generate a parse $\tilde{\mathbf{p}}$ from $(\mathbf{tag}_1, \mathbf{t}_2)$, where $\mathbf{tag}_1$ is the tag sequence of $\mathbf{x}_1$ and $\mathbf{t}_2$ is the template of $\mathbf{p}_2$. Then we use SynPG to generate a paraphrase from $(\mathbf{x}_1, \tilde{\mathbf{p}})$. We calculate the template matching accuracy to compare these two ways to generate paraphrases, as shown in Table 4. We also report the template matching accuracy of the generated parse $\tilde{\mathbf{p}}$.

We find that most of generated parses $\tilde{\mathbf{p}}$ indeed follow the target templates, which means that the parse generator usually generates good parses $\tilde{\mathbf{p}}$. Next, we observe that generating paraphrases with target parses usually performs better than with target templates. The results show a trade-off. Using templates proves more effortless during the generation process, but may compromise the syntactic control ability. In comparison, by using the target parses, we have to provide more detailed parses, but our model can control the syntax better.

Another benefit of generating paraphrase with target templates is that we can easily generate a lot of syntactically different paraphrases by feeding the model with different templates. Table 5 lists some paraphrases generated by SynPG with different templates. We can perceive that most generated paraphrases are grammatically correct and have similar meanings to the original sentence.

Model	Template Matching Accuracy			
	ParaNMT	Quora	PAN	MRPC
Paraphrases generated by target parses	71.0	82.6	66.3	74.0
Paraphrases generated by target templates	54.1	73.4	51.7	62.3
Parses $\tilde{\mathbf{p}}$ generated by parse generator	98.4	99.0	95.7	93.9

Table 4: Influence of using templates. Using templates proves more effortless during the generation process, but may compromise the syntactic control ability.

4.4 Training SynPG on Larger Dataset

Finally, we demonstrate that the performance of SynPG can be further improved and be even competitive to supervised models on some datasets if we consider more training data. The advantage of unsupervised paraphrase models is that we do not require parallel pairs for training. Therefore, we can easily boost the performance of SynPG by consider more unannotated texts into training.

We consider SynPG-Large, the SynPG model trained on the reference sentences of ParaNMT-50M as well as One Billion Word Benchmark (Chelba et al., 2014), a large corpus for training language models. We sample about 24 million sentences from One Billion Word and add them to the training set. In addition, we fine-tune SynPG-Large on only the reference sentences of the testing paraphrase pairs, called SynPG-FT.

From Table 6, We observe that enlarging the training data set indeed improves the performance. Also, with the fine-tuning, the performance of SynPG can be much improved and even is better than the performance of supervised models on some datasets. The results demonstrate the potential of unsupervised paraphrase generation with syntactic control.

4.5 Word Dropout Rate

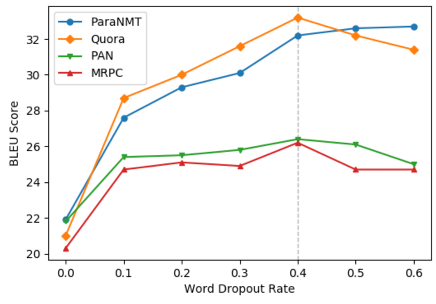
The word dropout rate plays an important role for SynPG since it controls the ability of SynPG to generate new words in paraphrases. We test differ-

Template	Generated Paraphrase
Original	can you adjust the cameras?
`(S(NP)(VP)(.))`	you can adjust the cameras.
`(SBARQ(ADVP)(,)(S)(,)(SQ)(.))`	well, adjust the cameras , can you?
`(S(PP)(,)(NP)(VP)(.))`	on the cameras, you can adjust them?
Original	she doesn't keep pictures from her childhood.
`(SBARQ(WHADVP)(SQ)(.))`	why doesn't she keep her pictures from childhood.
`(S(``)(NP)(VP)('')(NP)(VP)(.))`	" she doesn't keep pictures from her childhood " she said.
`(S(ADVP)(NP)(VP)(.))`	perhaps she doesn't keep pictures from her childhood.

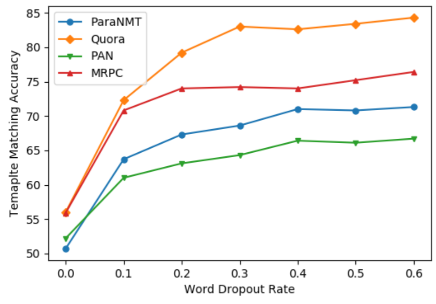
Table 5: Paraphrases generated by SynPG with different templates.

	Model	ParaNMT		Quora		PAN		MRPC	
		TMA	BLEU	TMA	BLEU	TMA	BLEU	TMA	BLEU
Ours	SynPG	71.0	32.2	82.6	33.2	66.3	26.4	74.0	26.2
	SynPG-Large	70.3	31.8	83.8	34.7	66.6	27.1	79.3	36.2
	SynPG-FT	–	–	86.3	**44.4**	66.4	34.2	**80.7**	**44.6**
Supervised Models	Seq2seq-Sup	40.2	19.6	54.0	11.3	29.2	13.1	44.3	16.3
	SCPN	**83.9**	**58.3**	**87.1**	41.0	**72.3**	**37.6**	80.1	41.8

Table 6: Training on larger dataset improves the performance of SynPG. Since training SynPG does not require annotated paraphrase pairs, it is possible to fine-tune SynPG on the texts in the target domain. With the fine-tuning, SynPG can have competitive or even better performance than supervised approaches.



(a) BLEU score



(b) Template matching accuracy

Figure 3: Influence of word drop out rate. Setting the word dropout rate to 0.4 can achieve the best BLEU score. However, higher word dropout rate leads to better template matching accuracy.

ent word dropout rates and report the BLEU scores and the template matching accuracy in Figure 3.

From Figure 3a, we can observe that setting the word dropout rate to 0.4 can achieve the best BLEU score in most of datasets. The only exception is ParaNMT, which is the dataset used for training. On the other hand, Figure 3b shows that higher word dropout rate leads to better template matching accuracy. The reason is that higher word dropout rate gives SynPG more flexibility to generate paraphrases. Therefore, the generated paraphrases can match the target syntactic specifications better. However, higher word dropout rate also make SynPG have less ability to preserve the meaning of source sentences. Considering all the factors above, we recommend to set the word dropout rate to 0.4 for SynPG.

5 Improving Robustness of Models

Recently, a lot of work show that NLP models can be fooled by different types of adversarial attacks (Alzantot et al., 2018; Ebrahimi et al., 2018; Iyyer et al., 2018; Tan et al., 2020; Jin et al., 2020). Those attacks generate *adversarial examples* by slightly modifying the original sentences without changing the meanings, while the NLP models change the predictions on those examples. However, a robust model is expected to output the same labels. Therefore, how to make NLP models not affected by the adversarial examples becomes an important task.

Since SynPG is able to generate syntactically different paraphrases, we can improve the robustness of NLP models by data augmentation. The models trained with data augmentation are thus more robust to the syntactically adversarial examples (Iyyer et al., 2018), which are the adversarial sentences that are paraphrases to the original sen-

Model	SST-2		MRPC		RTE	
	Acc.	Brok.	Acc.	Brok.	Acc.	Brok.
Base	91.9	46.7	84.1	52.8	63.2	58.3
SynPG	88.9	**39.6**	80.1	**35.5**	60.7	**33.9**

Table 7: Data augmentation improves the robustness of models. SynPG denotes the base classifier trained on augmented data generated by SynPG. Acc denotes the accuracy in the original dataset (the higher is the better). Brok denotes the percentage of examples changing predictions after attacking (the lower is the better).

tences but with syntactic difference.

We conduct experiments on three classification tasks covered by GLUE benchmark (Wang et al., 2019): SST-2, MRPC, and RTE. For each training example, we use SynPG to generate four syntactically different paraphrases and add them to the training set. We consider the setting to generate syntactically adversarial examples by SCPN (Iyyer et al., 2018). For each testing example, we generate five candidates of adversarial examples. If the classifier gives at least one wrong prediction on the candidates, we treat the attack to be successful.

We compare the model without data augmentation (Base) and with data augmentation (SynPG) in Table 7. We observe that with the data augmentation, the accuracy before attacking is slightly worse than Base. However, after attacking, the percentage of examples changing predictions is much less than Base, which implies that data augmentation indeed improves the robustness of models.

6 Related Work

Paraphrase generation. Traditional approaches usually require hand-crafted rules, such as rule-based methods (McKeown, 1983), thesaurus-based methods (Bolshakov and Gelbukh, 2004; Kauchak and Barzilay, 2006), and lattice matching methods (Barzilay and Lee, 2003). However, the diversity of their generated paraphrases is usually limited.

Recently, neural models make success on paraphrase generation (Prakash et al., 2016; Mallinson et al., 2017; Cao et al., 2017; Egonmwan and Chali, 2019; Li et al., 2019; Gupta et al., 2018). These approaches treat paraphrase generation as a translation task and design seq2seq models based on a large amount of parallel data. To reduce the effort to collect parallel data, unsupervised paraphrase generation has attracted attention in recent years. Wieting et al. (2017); Wieting and Gimpel (2018) use translation models to generate paraphrases via back-translation. Zhang et al. (2019); Roy and Grangier (2019) generate paraphrases based on variational autoencoders. Reinforcement learning techniques are also considered for paraphrase generation (Li et al., 2018).

Controlled generation. Recent work on controlled generation can be grouped into two families. The first family is doing end-to-end training with an additional trigger to control the attributes, such as sentiment (Shen et al., 2017; Hu et al., 2017; Fu et al., 2018; Peng et al., 2018; Dai et al., 2019), tense (Logeswaran et al., 2018), plots (Ammanabrolu et al., 2020; Fan et al., 2019; Tambwekar et al., 2019; Yao et al., 2019; Goldfarb-Tarrant et al., 2019, 2020), societal bias (Wallace et al., 2019; Sheng et al., 2020b,a), and syntax (Iyyer et al., 2018; Goyal and Durrett, 2020; Kumar et al., 2020). The second family controls the attributes by learning disentangled representations. For example, Romanov et al. (2019) disentangle the meaning and the form of a sentence. Chen et al. (2019b,a); Bao et al. (2019) disentangle the semantics and the syntax of a sentence.

7 Conclusion

We present syntactically controlled paraphrase generator (SynPG), an paraphrase model that can control the syntax of generated paraphrases based on the given syntactic specifications. SynPG is designed to disentangle the semantics and the syntax of sentences. The disentanglement enables SynPG to be trained without the need for annotated paraphrase pairs. Extensive experiments show that SynPG performs better syntactic control than unsupervised baselines, while the quality of the generated paraphrases is competitive to supervised approaches. Finally, we demonstrate that SynPG can improve the robustness of NLP models by generating additional training examples. SynPG is especially helpful for the domain where annotated paraphrases are hard to obtain but a large amount of unannotated text is available. One limitation of SynPG is the need for mannually providing target syntactic templates at inference time. We leave the automatic template generation as our future work.

Acknowledgments

We thank anonymous reviewers for their helpful feedback. We thank Kashif Shah and UCLA-NLP group for the valuable discussions and comments. This work is supported in part by Amazon Research Award.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B. Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *EMNLP*.

Prithviraj Ammanabrolu, Ethan Tien, W. Cheung, Z. Luo, W. Ma, Lara J. Martin, and Mark O. Riedl. 2020. Story realization: Expanding plot events into sentences. In *AAAI*.

Yu Bao, Hao Zhou, Shujian Huang, Lei Li, Lili Mou, Olga Vechtomova, Xin-Yu Dai, and Jiajun Chen. 2019. Generating sentences from disentangled syntactic and semantic spaces. In *ACL*.

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *NAACL*.

Igor A. Bolshakov and Alexander F. Gelbukh. 2004. Synonymous paraphrasing using wordnet and internet. In *NLDB*.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *CoNLL*.

Ziqiang Cao, Chuwei Luo, Wenjie Li, and Sujian Li. 2017. Joint copying and restricted generation for paraphrase. In *AAAI*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019a. Controllable paraphrase generation with a syntactic exemplar. In *ACL*.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019b. A multi-task approach for disentangling syntax and semantics in sentence representations. In *NAACL*.

Ning Dai, Jianze Liang, Xipeng Qiu, and Xuanjing Huang. 2019. Style transformer: Unpaired text style transfer without disentangled latent representation. In *ACL*.

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. Hotflip: White-box adversarial examples for text classification. In *ACL*.

Elozino Egonmwan and Yllias Chali. 2019. Transformer and seq2seq model for paraphrase generation. In *EMNLP*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2019. Strategies for structuring story generation. In *ACL*.

Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. 2018. Style transfer in text: Exploration and evaluation. In *AAAI*.

Seraphina Goldfarb-Tarrant, Tuhin Chakrabarty, Ralph Weischedel, and Nanyun Peng. 2020. Content planning for neural story generation with aristotelian rescoring. In *EMNLP*.

Seraphina Goldfarb-Tarrant, Haining Feng, and Nanyun Peng. 2019. Plan, write, and revise: an interactive system for open-domain story generation. In *NAACL system demonstration*.

Tanya Goyal and Greg Durrett. 2020. Neural syntactic preordering for controlled paraphrase generation. In *ACL*.

Ankush Gupta, Arvind Agarwal, Prawaan Singh, and Piyush Rai. 2018. A deep generative framework for paraphrase generation. In *AAAI*.

J. Edward Hu, Rachel Rudinger, Matt Post, and Benjamin Van Durme. 2019. PARABANK: monolingual bitext generation and sentential paraphrasing via lexically-constrained neural machine translation. In *AAAI*.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Toward controlled generation of text. In *ICML*.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First quora dataset release: Question pairs. *data.quora.com*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *NAACL*.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *AAI*.

Jerrold J Katz and Jerry A Fodor. 1963. The structure of a semantic theory. *language*, 39(2):170–210.

David Kauchak and Regina Barzilay. 2006. Paraphrasing for automatic evaluation. In *NAACL*.

Stanley Kok and Chris Brockett. 2010. Hitting the right paraphrases in good time. In *NAACL*.

Ashutosh Kumar, Kabir Ahuja, Raghuram Vadapalli, and Partha P. Talukdar. 2020. Syntax-guided controlled generation of paraphrases. *TACL*, 8:330–345.

Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. 2018. Paraphrase generation with deep reinforcement learning. In *EMNLP*.

Zichao Li, Xin Jiang, Lifeng Shang, and Qun Liu. 2019. Decomposable neural paraphrase generation. In *ACL*.

Lajanugen Logeswaran, Honglak Lee, and Samy Bengio. 2018. Content preserving text generation with attribute controls. In *NeurIPS*.

Nitin Madnani, Joel R. Tetreault, and Martin Chodorow. 2012. Re-examining machine translation metrics for paraphrase identification. In *NAACL*.

Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. 2017. Paraphrasing revisited with neural machine translation. In *EACL*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *ACL*.

Kathleen R. McKeown. 1983. Paraphrasing questions using given and new information. *Am. J. Comput. Linguistics*, 9(1):1–10.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair's WMT19 news translation task submission. In *WMT*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*.

Nanyun Peng, Marjan Ghazvininejad, Jonathan May, and Kevin Knight. 2018. Towards controllable story generation. In *NAACL Workshop*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Aaditya Prakash, Sadid A. Hasan, Kathy Lee, Vivek V. Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. 2016. Neural paraphrase generation with stacked residual LSTM networks. In *COLING*.

Alexey Romanov, Anna Rumshisky, Anna Rogers, and David Donahue. 2019. Adversarial decomposition of text representation. In *NAACL*.

Aurko Roy and David Grangier. 2019. Unsupervised paraphrasing without translation. In *ACL*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *NeurIPS*.

Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. 2020a. " nice try, kiddo": Ad hominems in dialogue systems. *arXiv preprint arXiv:2010.12820*.

Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. 2020b. Towards controllable biases in language generation. In *EMNLP-Findings*.

Pradyumna Tambwekar, Murtaza Dhuliawala, Lara J. Martin, Animesh Mehta, B. Harrison, and Mark O. Riedl. 2019. Controllable neural story plot generation via reward shaping. In *IJCAI*.

Samson Tan, Shafiq R. Joty, Min-Yen Kan, and Richard Socher. 2020. It's morphin' time! combating linguistic discrimination with inflectional perturbations. In *ACL*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. In *EMNLP*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

John Wieting and Kevin Gimpel. 2018. Paranmt-50m: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *ACL*.

John Wieting, Jonathan Mallinson, and Kevin Gimpel. 2017. Learning paraphrastic sentence embeddings from back-translated bitext. In *EMNLP*.

Yorick Wilks. 1975. An intelligent analyzer and understander of english. *Commun. ACM*, 18(5):264–274.

Zhao Yan, Nan Duan, Junwei Bao, Peng Chen, Ming Zhou, Zhoujun Li, and Jianshe Zhou. 2016. Docchat: An information retrieval approach for chatbot engines using unstructured documents. In *ACL*.

Lili Yao, Nanyun Peng, Weischedel Ralph, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-and-write: Towards better automatic storytelling. In *AAAI*.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. In *ICLR*.

Xinyuan Zhang, Yi Yang, Siyang Yuan, Dinghan Shen, and Lawrence Carin. 2019. Syntax-infused variational autoencoder for text generation. In *ACL*.

Sanqiang Zhao, Rui Meng, Daqing He, Andi Saptono, and Bambang Parmanto. 2018. Integrating transformer and paraphrase rules for sentence simplification. In *EMNLP*.