BoFGAN: Towards A New Structure of Backward-or-Forward Generative Adversarial Nets

M.K. Sophie Chen†
Institute of Computer Science and Technology, Peking University, Beijing, China
Department of Computer Science and Engineering, UC San Diego, La Jolla, USA

Xinyi Lin†
Institute of Computer Science and Technology, Peking University, Beijing, China

Chen Wei
Turing Robot, Beijing, China

Rui Yan*
Institute of Computer Science and Technology, Peking University, Beijing, China
ruiyan@pku.edu.cn

ABSTRACT

Natural Language Generation (NLG), as an important part of Natural Language Processing (NLP), has begun to take full advantage of recent advances in language models. Based on recurrent neural networks (RNNs), NLG has made ground breaking improvement and is widely applied in many tasks. RNNs typically learn a joint probability of words, and the additional information is usually fed to RNN's hidden layer using implicit vector representations. Still, there exists some problem unsolved. Standard RNN is not applicable when we need to impose *hard constraints* on the language generation tasks: for example, standard RNNs cannot guarantee designated word(s) to appear in a target sentence to generate. In this paper, we propose a Backward-or-Forward Generative Adversarial Nets model (BoFGAN) to address this problem. Starting from a particular given word, a generative model at every time step generates a new *preceding* or *subsequent* word conditioned on the generated sequence so far until both sides reach an end. To train the generator, we first model it as a stochastic policy using Reinforcement Learning; then we employ a discriminator to evaluate the quality of a complete sequence as the end reward; and lastly, we apply Monte Carlo (MC) search to estimate the long-term return and update the generator via policy gradient. Experimental results demonstrate the effectiveness and rationality of our proposed BoFGAN model.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Natural language generation**; • **Information systems** → *Web applications.*

*Corresponding author: Rui Yan (ruiyan@pku.edu.cn).
† indicates equal contributions.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313734

KEYWORDS

BoFGAN; constrained generation; reinforcement learning

ACM Reference Format:
M.K. Sophie Chen, Xinyi Lin, Chen Wei, and Rui Yan. 2019. BoFGAN: Towards A New Structure of Backward-or-Forward Generative Adversarial Nets. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3308558.3313734

1 INTRODUCTION

Natural Language Generation (NLG) is the process of producing high-quality texts from some underlying representation of key information. NLG is an important task in Natural Language Processing, and it is also a critical step in many applications, such as machine translation [1, 33], dialogue systems [36, 38], stylistic writing [6, 17, 35], storytelling [16, 37], and abstractive summarization [7, 28], etc. The task can be roughly divided into 2 steps: content determination and sentence structuring. Content determination involves choices of what semantic information to convey, which is closely related to the specific application and/or scenario. Although contents may vary, the way for sentence structuring of natural languages is similar even given different applications/scenarios. Thus, it is likely to propose a general framework to generate texts from given contents. There are three dominant approaches for sentence generation architectures: human-crafted templates [19, 27], grammar-based systems [3], statistical data-driven approaches [2, 20, 21, 26].

Template-based method is widely used in industry systems to generate articles by filling the templates with correct terms. It allows full control over the quality of the output, but it is quite labor-intensive, and rigid. It is only applicable when the application domains are small and variation is expected to be restricted. *Grammar*-based system is more flexible than template-based ones. However, the complexity and difficulty to design and to use these systems make them less popular. Recently, *statistical* approaches gain more attention in research work powered by deep learning. In particular, deep neural networks (DNNs) lead the way since they can learn dense, low-dimensional, and distributed representations at a more abstract level for sentence generation.

It is well-suited to get rid of hand-crafted templates and rules by capturing grammatical and semantic generalisations in a fixed size vector [18, 22, 25]. Thereinto, recurrent neural networks (RNNs) [21], a prevailing class of language models, has achieved notable successes in sequential modeling. It projects histories into a low-dimensional space and propagates the projection through the sequence to build up the implicit relations among the sequence. Compared to the standard *n*-gram model, RNNs can represent sequences of varying lengths, while avoiding both data sparseness and an explosion of parameter numbers.

There is a prominent drawback for RNNs: once the model is learned, the model generates autonomously. Standard RNN is not directly applicable when we need to impose *hard constraints* on the language generation tasks. In other words, for example, standard RNNs cannot guarantee designated word(s) to "explicitly appear" in the generated sentence. The designated word is the hard-constraint for sentence generation. The need for constrained generation is real.

Templates and grammar based systems are inflexible since different scenarios require different templates and grammars: the given word may not match any slots/templates. Existing language models, such as RNNs, can learn the joint probability of a sentence with an additional information. With a given word information fed into the hidden layer, RNN can generate a sentence semantically correlated with the word, but there is no policy to guarantee the word will appear.

Mou *et al.* [23] proposed a backward and forward language model (B/F LM) to tackle the problem. Given a specific word, the model generates previous words and subsequent words simultaneously or asynchronously. That is to say, once the generation direction is determined, the generation process will be executed until the end of the direction. A more natural way is that given the constraint word, we generate sentences incrementally, i.e., from words to word groups (e.g., phrases), from word groups to larger phrases, and from phrases to sentences. Such a process is to some extent analogous to human cognitions about natural language generation [8].

Here is a motivating case. Given a word "*convolutional*" and the target sequence is "*word order for text categorization with convolutional neural networks*", the B/F language model firstly generate backward words (*"with", "categorization", "text", "for", "order", "word"*) and then (*"neural", "networks"*). However, the entire sequence consists of two parts using *"with"* between *"word order for text categorization"* and *"convolutional neural networks"*. It is difficult to derive the first part from the single word *"convolutional"* since the word itself contains little information while it is more feasible to do so from the phrase *"convolutional neural networks"*.

In this paper, we propose a novel framework to tackle the problem of imposing hard constraints on sentence generation. Rather than *pre-defined* backward and then forward direction to generate words, we leave the choice to the model itself. The generative model is treated as an agent of Reinforcement Learning (RL): the state is the generated tokens so far and the action is the combination of what the next token is and which direction (backward, or forward) to generate it. We consider the generation procedure as a sequential decision making process: at each time, it decides to generate either a preceding or a subsequent word conditioned on generated tokens. One side will stop "growing" when it has generated the $\langle eos \rangle$ token. The reward is given by a discriminator, which evaluates the generated sequences and feeds the evaluation back to guide the learning of the generative model. Since the gradients can not pass back to the generator when the output is discrete, we follow SeqGAN [39] and regard the generative model as a stochastic parametrized policy. We employ Monte Carlo (MC) search to approximate the state-action value and directly train the policy via policy gradient. Theoretically, our model guarantees a sentence with a designated constraint word. The model does not plan a whole sentence with merely a single word in the first place; itself decides backward or forward to generate words, it has the chance to generate words into phrases, and to generate phrases into sentences.

Our contributions are as follows: we propose a novel *backward-or-forward* model with gradual information enrichment to generate a sentence given hard-constraints. The model generates text pieces to eliminate uncertainty and finally organize them into a complete sentence. We formulate the model under a Generative Adversarial Network framework and a backward/forward sequential decision making process. The model constructs sentences in a hierarchial generation process, from words to "phrases" and to sentences.

2 RELATED WORK

2.1 RNN and Sequence-to-Sequence

An RNN is widely used in language modeling, which is capable of capturing sequential dependency. It consists of an input layer, a hidden layer and an output layer. At each time t, a word is mapped to a fixed dimensional vector as an input x_t, which is called word embedding. The hidden state vector h_t depends on its previous hidden state h_{t-1} and the input vector x_t. The output layer predicts the generated word y_t from a vocabulary using a $softmax$ function. The formulas are as follows:

$$h_t = \text{RNN}(x_t, h_{t-1})$$

$$y_t = \text{softmax}(W_{out} h_t)$$

The vanilla RNN has the gradient vanishing or explosion problem. To solve this problem, Graves and Schmidhuber [10] proposed long short term memory (LSTM) unit, which contains a memory cell and three kinds of gates controlling what information is retained or forgotten.

Character-level RNN language model [32] has been applied to various generation tasks at early times. Later, a text-to-text NLG framework named sequence-to-sequence (Seq2Seq) [33] plays an important role in language generation. It combines the subtasks (i.e., content determination and sentence structuring) together through an encoder-decoder framework. The encoder encodes the source sentence into a fixed-size vector, acting as the content decision. The decoder decodes the vector to the target sentence, acting as the sentence realisation. The vector passed from encoder to decoder is an underlying representation of the content information of the generated sentence. LSTM-based RNN can be used as the encoder and the decoder.

2.2 B/F Language Model

The standard RNNs or LSTMs are able to formulate language models but they cannot deal with language models with constraints.

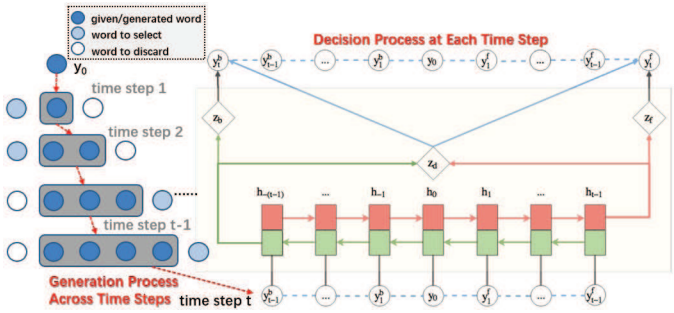

Figure 1: BoFGAN sequence generation at time step t: G_θ would generate the next two tokens conditioned on the previously generated sequence $Y_{0:t-1}$. Note that the red flow is forward h_f propagation, and the green flow is the backward h_b propagation. The diamonds are softmax layers: z_d represents the confidence for direction, and z_f, z_b choose the next token. Across time steps, the model repeats the process incrementally, starting from the constraint word, until the whole sentence is completed by $\langle eos \rangle$'s.

Under the scenarios that constraints exist, the B/F language model, introduced by [23], is a new language model towards the generation of constrained natural language. Given a specific word, the model generates preceding words and subsequent words simultaneously or asynchronously. In the simultaneous setting (syn-B/F model), there is a shared RNN sequence which emits words in the backward and forward directions at the same time. The backward language model and the forward language model share the same input layer and hidden layer. In the asynchronous setting (asyn-B/F model), the model generates the backward sequence firstly, which is then fed to another RNN to generate the forward sequence.

For both settings, the B/F language model pre-defines the directions to generate the sequence while we are seeking for a more flexible model that decides which direction to generate according to the current status and changes the generation strategies from time to time, dynamically.

2.3 GAN and SeqGAN

Generative adversarial networks (GAN) proposed by [9] is a state-of-the-art generative model, which has been successfully applied in generating picture samples [5, 11]. GAN includes a discriminator D classifying the given data as real or fake, and a generator G generating almost real data to confuse D. G and D finally reach a Nash equilibrium, which can be regarded as a win-win situation. However, problems arise when applying GAN in sequential language generation.

Originally, GAN is defined in real number domain for continuous conditions. G is updated via the gradient loss from D. In general, it makes sense only when we slightly change the synthetic data which is continuous, such as pixel values. When GAN generates discrete tokens, the slightly changed data may not have corresponding tokens from the vocabulary. Besides, we can only evaluate the quality of a sentence when it is completely generated. To this end, Yu *et al.* [39] proposed a model named SeqGAN considering the generation procedure as a sequential decision making process, where the generator is an agent of Reinforcement Learning (RL) and the discriminator is used to score the generated sequence and feedback the reward. They employ Monte Carol (MC) search to approximate the state-action value and train the policy via the policy gradient.

However, the GAN and SeqGAN models cannot deal with constrained information so that we can not guarantee the designated word will appear in the generated sentence.

3 BOFGAN

In this part, we introduce our proposed *Backward-or-Forward GAN* (BoFGAN) model in details. The model consists of two parts: a generator and a discriminator. The generator is an RL agent to generate a sentence from a given token. Starting from a single word, each time it needs to make two choices: where to put the new word and what the new word is. The generation process will reach an end when both sides have met an $\langle eos \rangle$ token or the sequence length exceeds limits. In other words, the state is the tokens generated so far, and the actions consist of 1) the position and 2) selection of the new word. Since it is hard to evaluate an incomplete sequence with an appropriate reward for each step, we also employ MC search to approximate the action value function and evaluate the complete sequence using a discriminator. We treat the agent's behaviour as a stochastic parameterized policy and train it using policy gradient method. The discriminator is trained to evaluate how likely a sequence is from real data or not by learning from real-world data and synthetic data from the generator. The generator and discriminator are trained to improve each other iteratively in an adversarial way.

3.1 Generator

G_θ is the θ-parameterized generative model. Γ is the vocabulary of candidate words plus the empty token ε. y_0 is the **hard-constraint** word, i.e., the start token given to the generator in the beginning and is required to exist in the generated sentence.

At timestep t, the state s_{t-1} is the previously generated sequence $Y_{0:t-1} = (y^b_{t-1}, y^b_{t-2}, ..., y^b_1, \mathbf{y_0}, y^f_1, ..., y^f_{t-2}, y^f_{t-1})$, where $y^f_i, y^b_i \in \Gamma$. The action a_t is to separately append new tokens (y^b_t, y^f_t) to two ends of $Y_{0:t-1}$, i.e., the next state s_t means $Y_{0:t} = (y^b_t, y^b_{t-1}, y^b_{t-2}, \ldots, y^b_1, y_0, y^f_1, \ldots, y^f_{t-2}, y^f_{t-1}, y^f_t)$. For the *backward or forward* language modeling, at least one of (y^b_t, y^f_t) is ε because the G_θ is allowed to append one valid word to the sequence at each timestep of generation.

To generate two tokens at a time, we use the bi-directional RNN [29] to model the policy $G_\theta(y^b_t, y^f_t | Y_{0:t-1})$. The bi-RNN maps $X_{0:t-1} = (x_{-t+1}, x_{-t+2}, ..., x_{-1}, x_0, x_1, ..., x_{t-1})$, the input embedding representation of $Y_{0:t-1}$, into two sequences of hidden state, by using the update function g recursively:

$$h_{0:t-1} = (\{h^b_{-t+1}, h^f_{-t+1}\}, \ldots, \{h^b_0, h^f_0\}, \ldots, \{h^b_{t-1}, h^f_{t-1}\})$$

$$\begin{aligned} h^f_i &= g_f(h^f_{i-1}, x_i) \\ h^b_i &= g_b(h^b_{i+1}, x_i) \end{aligned} \tag{1}$$

where h^f_{-t} and h^b_t, the initial vectors to feed into the hidden layer, are both zero vectors. Note that we allow the subscripts be negative for convenience and each subscript stands for the relative position to y_0. g_f and g_b are implemented by LSTM cells, since LSTM cells can faster converge and eliminate the problem of vanishing or exploding gradients in sequence via self-regularization [10].

Generator's action actually consists of two parts: 1) the position and 2) the selection of the new token, thus, we use three output units to parameterize the policy.

$$\begin{aligned} z_d &= \text{softmax}\left(b_d + W_d h\right) \\ z_f &= \text{softmax}\left(b_f + W_f h^f_{t-1}\right) \\ z_b &= \text{softmax}\left(b_b + W_b h^b_{-t+1}\right) \end{aligned} \tag{2}$$

where W_d, W_f, W_b are weight matrics and b_d, b_f, b_b are bias vectors. h is the concatenation of h^f_{t-1} and h^b_{-t+1}. z_d is a two-dimension vector to control the preference for which direction to append newly generated tokens, and z_f, z_b are used to represent the output token distribution separately:

$$\begin{aligned} p\left(y^f_t | Y_{0:t-1}\right) &= z_{d0} \cdot z_f \\ p\left(y^b_t | Y_{0:t-1}\right) &= z_{d1} \cdot z_b \end{aligned} \tag{3}$$

Note that

$$\sum_{y^f_t \in \Gamma} p\left(y^f_t | Y_{0:t-1}\right) + \sum_{y^b_t \in \Gamma} p\left(y^b_t | Y_{0:t-1}\right) = 1.$$

At timestamp t, the generator would only sample one word from $p\left(y | Y_{0:t-1}\right)$. If one side has obtained $\langle eos \rangle$ token, the corresponding direction of z_d would be forced to be 0 and the other to be 1. When both sides reach an end or the total length goes beyond the limit, the generation is terminated. The whole generation process is shown in Figure 1.

3.2 Discriminator

D_ϕ is the ϕ-parameterized discriminative model. It is a binary classifier that takes a sequence as the input and outputs a probability indicating how likely the input is generated by humans. Deep learning methods have been widely used in the task of text classification and achieves good performance, especially using the convolutional neural network (CNN)-based methods [12, 13, 15]. CNN is similar to the n-gram model to some degree as the filter window can be seen as the n-gram method. The use of convolution layer and pooling layer can largely reduce the number of parameters and extracts high-level information of text at the same time. We follow SeqGAN [39] and choose the same architectures as our discriminator.

The overall classification framework is based on CNN. The input sequence is represented as:

$$x_{1:T} = x_1 \oplus x_2 \oplus \cdots \oplus x_T$$

where $\oplus$ is the concatenation operator and $x_t \in \mathbb{R}^k$ is the k-dimensional word embedding corresponding to the t-th word in the sequence. The convolutional operation is then applied to a window size of m words with a kernel $w \in \mathbb{R}^{m \times k}$. We can use multiple kernels to extract different features and the window size varies when needed. Each feature map can be denoted as:

$$\mathbf{c} = [c_1, c_2, \cdots, c_{T-m+1}]$$

where

$$c_i = f(w \cdot x_{i:i+m-1} + b)$$

Here f is a non-linear function and $b \in \mathbb{R}$ is a bias term. Then, a max-over-time operation is applied to the feature map getting the most important feature $\tilde{c} = max\{\mathbf{c}\}$

We add highway networks [31] after the pooling operation. At last, we apply a fully connected layer to distinguish whether the input sequence is from human or not. The loss function is the cross entropy between the ground truth label and the classifier output.

3.3 BoFGAN via Policy Gradient

To guide the learning of the generator, we establish an adversarial reinforcement architecture. The key idea of the system is to encourage the G_θ to generate a semantically coherent and grammatically correct sentence with a constraint word. In other words, our objectives are that 1) the generated sequence includes the designated constraint word, and 2) is indistinguishable from human generated sentences.

We construct a bi-directionally growing generator. On one hand, the model guarantees that the constraint word will appear; on the other hand, it splits the whole generation task into sub-tasks, i.e., from words to word groups (e.g., phrases), from text pieces to sentences, making each sub-task easier than the task to generate a sentence as a whole. The process is quite similar to human cognitions [8].

We employ a discriminator to assess the quality of the generated sentence in the end. If the synthetic sentence gets a high probability from D_ϕ, that means the sentence is good enough to fool the discriminator and shows the effectiveness of the generator. However, it exists a problem applying traditional GAN to our model: it has difficulty to pass back the gradient to G_θ due to the discrete output. Thus, we adopt the policy gradient method to train the generator, and we use MC search to estimate the action-value function and the discriminator to provide the reward.

To improve the authority of the discriminator, we periodically re-train the discriminator by feeding real-world data and synthetic data generated by the upgraded Generator G_θ. The objective of the discriminative model is to minimize the cross entropy of the classification:

$$\min_\phi -\mathbb{E}_{Y \sim p_{\text{data}}}\left[\log D_\phi(Y)\right] - \mathbb{E}_{Y \sim G_\theta}\left[\log\left(1 - D_\phi(Y)\right)\right] \tag{4}$$

Each time when we obtain an upgraded discriminator D_ϕ, we continue the improvement of Generator G_θ. The objectives of G_θ is to maximize the state value of state s_0.

$$\begin{aligned} J(\theta) &= V_\theta(s_0) \\ &= \mathbb{E}\left[Q_\theta(s_0, a)\right] \\ &= \sum_{y^b_1, y^f_1 \in \Gamma} G_\theta\left(y^b_1, y^f_1 | s_0\right) \cdot Q^{G_\theta}_{D_\phi}\left(s_0, y^b_1, y^f_1\right) \end{aligned} \tag{5}$$

$V_\theta(s_0)$ is the state value of s_0, $Q_\theta(s_0, a)$ is the action value function, and the $Q^{G_\theta}_{D_\phi}\left(s_0, y^b_1, y^f_1\right)$ is the approximation of $Q_\theta\left(s_0, y^b_1, y^f_1\right)$

obtained from MC Search. Since there is no intermediate reward, we utilize the discriminator to provide the reward for the complete sequence, apply MC search with a roll-out policy G_β to sample the rest of the sequence, and estimate action value function with the empirical mean return. Note that, to improve the accuracy, we set β the same as θ.

To reduce variance, we run the roll-out policy for N times to get a batch of samples. Starting from sequence $Y_{0:t}$, the sampled sequences are represented as

$$\{Y_{0:T}^1, ..., Y_{0:T}^N\} = \mathrm{MC}^{G_\beta}(Y_{0:t}; N)$$

Thus, the $Q_{D_\phi}^{G_\theta}(s_{t-1}, y_t^b, y_t^f)$ can be written as:

$$Q_{D_\phi}^{G_\theta}(s_{t-1}, y_t^b, y_t^f) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y_{0:T}^n),\ Y_{0:T}^n \in \mathrm{MC}^{G_\beta}(Y_{0:t}; N) & \text{for } t < T \\ D_\phi(Y_{0:T}) & \text{for } t = T \end{cases} \tag{6}$$

We use policy gradient method to update G_θ. Following [34], the gradient of $J(\theta)$ can be derived as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_{t-1}\sim G_\theta}\Big[\sum_{y_t^b, y_t^f \in \Gamma} \nabla_\theta G_\theta(y_t^b, y_t^f | s_{t-1}) \cdot Q_{D_\phi}^{G_\theta}(s_{t-1}, y_t^b, y_t^f)\Big] \tag{7}$$

Thus, the equation above can be unbiasedly estimated using likelihood ratios as:

$$\begin{aligned} \nabla_\theta J(\theta) &\simeq \frac{1}{T}\sum_{t=1}^{T}\sum_{y_t^b, y_t^f \in \Gamma} \nabla_\theta G_\theta(y_t^b, y_t^f | s_{t-1}) \cdot Q_{D_\phi}^{G_\theta}(s_{t-1}, y_t^b, y_t^f) \\ &= \frac{1}{T}\sum_{t=1}^{T}\sum_{y_t^b, y_t^f \in \Gamma} G_\theta(y_t^b, y_t^f | s_{t-1}) \nabla_\theta \log G_\theta(y_t^b, y_t^f | s_{t-1}) \cdot Q_{D_\phi}^{G_\theta}(s_{t-1}, y_t^b, y_t^f) \\ &= \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{(y_t^b, y_t^f)\sim G_\theta(y_t^b, y_t^f | s_{t-1})}\Big[\nabla_\theta \log G_\theta(y_t^b, y_t^f | s_{t-1}) \cdot Q_{D_\phi}^{G_\theta}(s_{t-1}, y_t^b, y_t^f)\Big] \end{aligned} \tag{8}$$

For brevity, we omit several details of derivations. We can approximate the expected value with sampled means, thus, we can update the parameters θ as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta),$$

where α is learning rate. We can adopt gradient algorithms such as Adam [14] here.

To sum up, we first pretrain the generator and the discriminator. Then the generator and discriminator are trained alternately. For the discriminator, positive samples are from human generated sequences and negative samples are synthetic sequences generated by the generator. We need to periodically re-train the discriminator for several times as we get synthetic data of higher quality from the improved generator. In turn, the generator needs to be re-trained after we obtain an upgraded discriminator. In the end, the discriminator and the generator can both get improved.

4 EXPERIMENT

We conduct experiments to evaluate the effectiveness and to understand the inherent mechanism of BoFGAN.

4.1 Experimental Setups

Data and Pre-processing. We implement our experiments over the the Westbury Lab Wikipedia Corpus [30]. We carry out the data pre-processing, including data selection and data cleaning. First, we split the paragraphs into sentences and replace some entities with designated symbols, such as numbers, name mentions, temperature and so on. Since extremely short sentences are less meaningful, we extract one million sentences of length between 5 and 10 tokens as dataset $Data_1$, and one million sentences of length between 10 and 20 tokens as dataset $Data_2$. Both vocabulary sizes are set as 30,000, and the words out of vocabulary ($\leq$ 25 occurrences) were grouped as a single token, ⟨UNK⟩. Then we remove those sentences with more than three ⟨UNK⟩'s. Both $Data_1$ and $Data_2$ have 900,000 sentences, including 800,000 samples for training, and 100,000 for testing.

Baselines. We compare our model with traditional RNN language models as one of our baselines. Since standard RNNs can only generate sequence in one direction, it cannot handle our task scenario if a constraint word occurs in the middle of the target sentence. Therefore, for RNNs, we test the performance of **Forward-RNN** (F-RNN), starting from the constraint word, as well as the performance of **Backward-RNN** (B-RNN), ending by the constraint word. RNN is capable to deal with sequential generation in only one direction. We also include the B/F language models with RNNs. As mentioned above, B/F language model is designed for the same scenario as ours: to generate a sentence with the hard constraint word. The **asyn-B/F** language model is the best method as reported in [23].

In the pre-training process, asyn-BF language model is splitted into two part: the forward one and the backward one, each of which is trained as normal RNN language models. We feed the first word of each sentence as the start token, and the forward sequence reconstructs the original sentence. For the backward one, we reverse the sequence and train in the same way. The pretraining loss of asyn-BF is as:

$$-\frac{1}{M}\sum_{i=1}^{M}\left[\log p_f(Y_{1:T}|y_0) + \log p_b(Y_{T-1:0}|y_T)\right] \tag{9}$$

where p_f and p_b are the probability of forward generation and backward generation, and M is the batch size.

Implementation. Since BoFGAN is implemented based on the bi-directional architecture, thus, we use the same pretraining method mentioned above for the generator G_θ. At each time step, it actually acts like a bi-directional RNN if we ignore the direction decision layer z_d. After pretraining, the generator is preliminarily able to generate the start or end token for a sequence. As for the softmax layer of z_d, we leave it for next training stage.

We use LSTM architectures in both our generator and asyn-B/F language model. Actually, most of the RNN variants, such as the GRU [4] and soft attention mechanism [1], can also be applied. We adopt Adam gradient algorithms [14] for all the stochastic optimization, including the training of generator and discriminator. Adam is computationally efficient, has low memory cost, requires little tuning, and is invariant to diagonal rescaling of the gradients.

Evaluation Metric. Since our model is not learned by joint probability of word estimation during generation, it is infeasible to

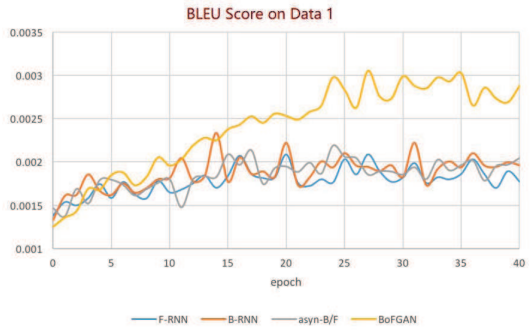

Figure 2: BLEU score curves on dataset $Data_1$

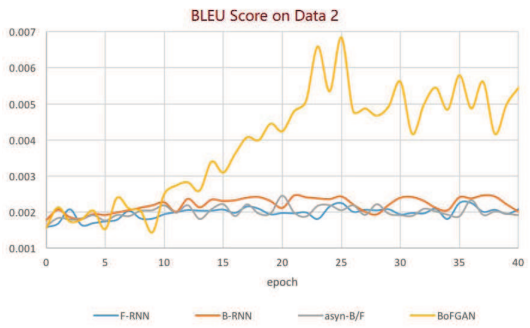

Figure 3: BLEU score curves on dataset $Data_2$

Table 1: Evaluation scores of different groups, measured in terms of BLEU evaluations.

Methods	BLEU of $Data_1$	BLEU of $Data_2$
F-RNN	1.82	2.04
B-RNN	1.96	2.37
asyn-B/F	1.84	2.02
BoFGAN	**2.83**	**4.73**

evaluate with *perplexity* as other language model studies do. *BLEU* is an algorithm to evaluate the quality of generation by comparing the overlap between the generated texts and the target references from humans [24]. BLEU is calculated by taking the geometric mean of the N-gram precision scores and then multiply the result by an exponential brevity penalty factor. Readers may refer to [24] for details. In this work, we regard all target sentences with the constraint word as references for BLEU calculation. N is set to 4 combining BLEU-1 to BLEU-4.

4.2 Experimental Results

The BLEU performance of all methods is provided in Table 1, measuring the overlap between the generated sentences and the targeted sentences as references. For each method, we use the best model in training stage to generate candidate samples and calculate their BLEU score. We can see that BoFGAN shows apparent advantages over other methods. BoFGAN outperforms baselines both on dataset $Data_1$ and $Data_2$, and it shows much larger gap in $Data_2$, which is trained to generate longer sentences. As mentioned above, when a sentence is longer, it is more likely to contain complex structures. This situation means that it would be more favorable for BoFGAN to perform its stronger capability. There is not so much difference among three baselines, since they are all based on the same RNN language model. Note that Forward RNN and Backward RNN has a "prior" knowledge because they are aware of the position of the constraint word (at the beginning or the end). Besides, they can only generate sequence in one way. Thus, if they are fed with a constraint word at a random position of the target sentence, they would perform much worse than BoFGAN and asyn-BF model.

In Figures 2-3, we show the learning curves, which strongly illustrate the effectiveness of BoFGAN. In the pretraining period—from epoch 0 to 10—BoFGAN is even slightly weaker than baselines since parameters of BoFGAN are not fully trained. In the training stage, i.e., from epoch 11 to 40, BoFGAN quickly outperforms baselines and enlarges the gap gradually, which demonstrates the effectiveness.

5 CONCLUSION

In this paper, we proposed a BoFGAN model for hard-constrained natural language generation. The model consists of two parts: a generator treated as a stochastic policy in reinforcement learning, and a discriminator to give a reward. Given the constraint word, we can generate a complete sentence containing it. The generation process is innovative. Starting from the given word, 1) which direction to generate and 2) what is the next token to generate are conditioned on previously generated words so far. Thus, we can form a word group (e.g., a phrase) from individual words and form a sentence from the word groups (or phrases).

Experimental results show that BoFGAN outperforms other baselines using BLEU scores. The generation process enlightens us another perspective of natural language generation paradigm: not only sequential, but also *backward-or-forward*. We will investigate how to organize multiple inconsecutive constraint words with our proposed model.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (No. 2017YFC0804001), the National Science Foundation of China (NSFC No. 61672058; NSFC No. 61876196), and Tencent Open Research Fund.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473* (2014).
[2] Srinivas Bangalore and Owen Rambow. 2000. Corpus-based lexical choice in natural language generation. In *ACL*. 464–471.
[3] John A. Bateman. 1997. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering* 3, 1 (1997), 15–55.
[4] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259* (2014).
[5] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. 2015. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. In *NIPS*. 1486–1494.
[6] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. 2018. Style transfer in text: Exploration and evaluation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[7] Shen Gao, Xiuying Chen, Piji Li, Zhaochun Ren, Lidong Bing, Dongyan Zhao, and Rui Yan. 2019. Abstractive Text Summarization by Incorporating Reader Comments. In *AAAI'19*.

[8] Merrill F Garrett. 1980. Levels of processing in sentence production. *Language production I* (1980), 177–220.

[9] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. In *NIPS*. 2672–2680.

[10] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5 (2005), 602–610.

[11] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. 2016. Generating images with recurrent adversarial networks. *arXiv:1602.05110* (2016).

[12] Rie Johnson and Tong Zhang. 2014. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *arXiv:1412.1058* (2014).

[13] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882* (2014).

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (2014).

[15] Ji Young Lee and Franck Dernoncourt. 2016. Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks. In *HLT-NAACL*. 515–520.

[16] Juntao Li, Lidong Bing, Lisong Qiu, Dongmin Chen, Dongyan Zhao, and Rui Yan. 2019. Learning to Write Creative Stories with Thematic Consistency. In *AAAI'19*.

[17] Juntao Li, Yan Song, Haisong Zhang, Dongmin Chen, Shuming Shi, Dongyan Zhao, and Rui Yan. 2018. Generating Classical Chinese Poems via Conditional Variational Autoencoder and Adversarial Training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3890–3900.

[18] Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. In *CoNLL*. 104–113.

[19] S. W Mcroy. 2003. An augmented template-based approach to text realization. *Natural Language Engineering* 9, 4 (2003), 381–420.

[20] Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. 2019. CGMH: Constrained Sentence Generation by Metropolis-Hastings Sampling. In *AAAI'19*.

[21] Tomas Mikolov, Martin KarafiÃąt, Lukas Burget, Jan CernockÃ¡, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*. 1045–1048.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[23] Lili Mou, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. 2016. Backward and Forward Language Modeling for Constrained Sentence Generation. *Computer Science* 4, 6 (2016), 473–482.

[24] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *ACL*. 311–318.

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. 1532–1543.

[26] Adwait Ratnaparkhi. 2000. Trainable Methods for Surface Natural Language Generation. *arXiv:cs.CL/0006028* (2000).

[27] Ehud Reiter, Chris Mellish, and Jon Levine. 1995. Automatic Generation of Technical Documentation. *Journal of Applied Artificial Intelligence* (1995).

[28] Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization.. In *EMNLP*. 379–389.

[29] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* (1997).

[30] C. Shaoul and Westbury C. 2010. The Westbury Lab Wikipedia Corpus, Edmonton, AB: University of Alberta.

[31] Rupesh Kumar Srivastava, Klaus Greff, and JÄijrgen Schmidhuber. 2015. Highway Networks. *arXiv:1505.00387* (2015).

[32] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. 2011. Generating Text with Recurrent Neural Networks. In *ICML*. 1017–1024.

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.

[34] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*. 1057–1063.

[35] Rui Yan. 2016. i, Poet: Automatic Poetry Composition through Recurrent Neural Networks with Iterative Polishing Schema.. In *IJCAI*. 2238–2244.

[36] Rui Yan. 2018. "Chitty-Chitty-Chat Bot": Deep Learning for Conversational AI.. In *IJCAI'18*. 5520–5526.

[37] Lili Yao, Nanyun Peng, Weischedel Ralph, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-And-Write: Towards Better Automatic Storytelling. In *AAAI'19*.

[38] Lili Yao, Yaoyuan Zhang, Yansong Feng, Dongyan Zhao, and Rui Yan. 2017. Towards implicit content-introducing for generative short-text conversation systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2190–2199.

[39] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *AAAI*. 2852–2858.