IMPROVING CONDITIONAL SEQUENCE GENERATIVE ADVERSARIAL NETWORKS BY EVALUATING AT EVERY GENERATION STEP

Anonymous authors
Paper under double-blind review

ABSTRACT

Conditional sequence generation is a widely researched topic. One of the most important tasks is dialogue generation, which is famously composed of input-output pairs with one-to-many property. Recently, the success of generative adversarial network (GAN) has attributed researchers to apply GAN on sequence generation. However, there are still limited researches on the conditional sequence generation. We then investigate the influence of GAN on conditional sequence generation with three artificial grammars which share similar properties with dialogue generation. We compare the state-of-the-art GAN related algorithms by evaluation indexes: the accuracy and the coverage of answers. Moreover, we proposed every step GAN (ESGAN) for conditional sequence generation, which predicts reward at each time-step. ESGAN can be considered as the general version of SeqGAN including MCMC and REGS in the model. We also explore energy-based ESGAN (EBESGAN). These then solve the sparse rewards problem in reinforcement learning. In addition, with weighted rewards for every steps, the advantage of curriculum learning is also included. The experimental results show that ESGAN and EBESGAN are usually comparable with the state-of-the-art algorithms, and sometimes outperform them.

1 INTRODUCTION

Conditional sequence generation aims at generating correspondent responses given the input sequence. One of the most important applications is dialogue generation. Dialogue generation has one-to-many property. That is, given a specific input, there are various reasonable responses. Previous works train sequence-to-sequence based dialogue generation model by using maximum likelihood estimation and achieve promising results for both meaningful and coherence (Vinyals & Le, 2015). Despite the success, the generated responses given the inputs are still sometimes broken and are often general (egs. I don't know). Reinforcement learning is therefore proposed to preserve the sequence-level quality instead of predicting each words given the sequence history (Ranzato et al., 2015; Kandasamy et al., 2017; Bahdanau et al., 2016).

More recently, generative adversarial networks have been applied on sequence generation, especially on natural language. Since random variable of natural language is discrete, this leads the difficulty for back-propagation. To solve this problem, several approaches have been proposed, such as policy gradient (Yu et al., 2017; Li et al., 2017), gumbel-softmax (Kusner & Hernández-Lobato, 2016), MaliGAN (Che et al., 2017), directly connected WGAN-GP (Gulrajani et al., 2017; Rajeswar et al., 2017; Press et al., 2017), etc. In most of the previous works, the researchers use multiple assistant methods to stabilize the training and often introduce improvements, for example, Monte-Carlo search (Yu et al., 2017; Li et al., 2017; Che et al., 2017) and curriculum learning (Rajeswar et al., 2017; Press et al., 2017). Furthermore, the modifications of objective function in GAN are proposed to improve the quality for text generation (Zhang et al., 2017; Lin et al., 2017).

In this paper, we propose every step GAN (ESGAN) and energy-based every step GAN (EBESGAN). Both approaches predict a score for each time-step when decoding. In particular, ESGAN includes weighted factors that can change the relative importance of each time-step. By the factor, ESGAN is actually the *general version of SeqGAN*. Furthermore, step-time-decreased

weighted factor can simulate *curriculum learning* by paying attention on generating the head of a whole sequence. After the first subsequence are fit, the further improvement will be found from the later subsequece. We experimentally find that step-time-decreased weighted factor indeed improve the performance. On the other hand, EBESGAN use cross-entropy to evaluate the *energy* of generated sequence at each time-step. The discriminator here is actually the same as generator. Therefore the generator and discriminator can share the same pretrained model.

To find the assistance of GANs on conditional sequence generation, we constructed artificial grammars to help our realization of assistance of GANs. For these tasks, the accuracy and the coverage of the generated conditioned sequence are considered to evaluate the quality of the model. The coverage is the percentage of the conditioned sequences sampled from model distribution over all the probable responses. While the accuracy can measure the coherence and meaningfulness, coverage can measure the diversity of responses. Further, we present the examples of dialogue generation and the results of human evaluation. Our proposed models can be comparable with or even outperform the state-of-the-art algorithms.

2 Related Works

Conditional sequence generation using seq2seq model (Sutskever et al., 2014; Vinyals & Le, 2015) has been widely studied, especially for dialogue generation. The model can be learned by maximum-likelihood estimation (MLE) which minimizes the cross-entropy between the true data distribution and the generated approximation at the word level. This method indeed ensures reasonable responses, however, suffers from exposure bias and lack of considering sequence-level structure (Ranzato et al., 2015). *Exposure bias* is introduced because of inconsistent conditions between training and testing stages. While the ground-truth words are fed into seq2seq model in training stage, the generated words are fed into the model in testing stage.

To solve problems of MLE, (Ranzato et al., 2015) proposed to apply REINFORCE and MIXER on sequence generation. By providing a task-specific score of the generated sequence, the REINFORCE algorithm (Williams, 1992) can guide the seq2seq model to reach higher score. Because the score is evaluated based on the whole generated sequence, both disadvantages of MLE are solved. However, REINFORCE algorithm can not easily trained from scratch, so the MIXER is proposed, which integrates MLE and REINFORCE. The detailed process is first training the whole sequence by MLE, then accumulating the number of last words trained by REINFORCE. For further improvement, (Bahdanau et al., 2016) adopted another reinforcement learning (Sutton & Barto, 1998) based approach - the actor-critic architecture. They trained a critic to predict the expected value of each time-step to guide the actor. These algorithms outperform the original MLE algorithm on the task-specific score (egs. BLEU) for text generation. Nonetheless, there is no evidence that the task-specific scores strongly related with the prior knowledge of human. Especially, the relation between the scores and human evaluation has been proved weak on dialogue generation (Liu et al., 2016).

Recently, the significant success of generative adversarial network (GAN) on image processing has intrigued researchers to apply GAN on natural language. However, it's rather restricted because of the difficulty of back-propagation through discrete random variables. To address this problem, (Yu et al., 2017) used policy gradient on text generation. The reward is provided by the discriminator with Monte-Carlo search. In addition, (Li et al., 2017) adopted the same idea on dialogue generation task. They also proposed Reward for Every Generation Step (REGS), which is more time efficient but worse performed than Monte-Carlo search. Another solution to apply GAN on natural language tasks is using gumbel-softmax (Kusner & Hernández-Lobato, 2016). It can simulate the discrete argmax outputs, and be directly backpropagated from discriminator. Other than these, MaliGAN (Che et al., 2017) derives the objective function irrelevant to backpropagation through the discrete outputs of generator. More recently, the improved Wasserstein GAN (WGAN-GP) (Gulrajani et al., 2017) successfully trained text generation task by directly feeding softmax layer to discriminator and even without pre-training. This breakthrough then inspired (Press et al., 2017) and (Rajeswar et al., 2017) to further investigate WGAN-GP for achieving better performance on text generation.

3 CONDITIONAL SEQUENCE GENERATION

The *conditional sequence generation* is to generate an output sequence $x \in X$ given an input condition $y \in Y$. When the output sequence is generated from a model, it is called x^G. On the other hand, when the output sequence is from real data (training examples), it is annotated as x^R. In dialogue generation, both x and y are sequences composed of words. They can be written as:

$$\begin{aligned} x &= \{x_t\}_{t=1}^{T}, x_t \in V \\ y &= \{y_t\}_{t=1}^{T}, y_t \in V \end{aligned} \tag{1}$$

The x_t and y_t represent the word at time-step t in the interval T of the specific sequences, x and y. The set V is the vocabulary set where words are selected from. The generator G used to generate x given y in this paper is seq2seq model (Sutskever et al., 2014; Vinyals & Le, 2015). From sections 3.1 to 3.3, we are going to introduce maximum likelihood estimation, REINFORCE algorithm for sequence generation and GAN for sequence generation. In section 4, we will introduce the proposed approaches.

3.1 MAXIMUM LIKELIHOOD ESTIMATION

The basic idea of maximum likelihood estimation (MLE) in conditional sequence generation is to find parameters for model G that maximize the *likelihood* of generating the training data. When using MLE to train generator model G, the objective function is shown as below:

$$G^* = \arg\max_G E_{(x^R,y)\sim P^R(X,Y)}[\sum_{t=1}^{T} log(P^G(x_t^R|y, x_{1...t-1}^R))], \tag{2}$$

where $P^R(X,Y)$ is the joint distribution of the (x,y) pairs in the training data. T is the length of x^R. In the training stage, in equation (2), the prediction is learned based on $< y, x_{1...t-1}^R >$, but the condition in the testing stage is $< y, x_{1...t-1}^G >$. This problem is called *exposure bias*. Moreover, the likelihood is only estimated at word-level (Ranzato et al., 2015) instead of the whole sequence.

3.2 REINFORCE

The conditional sequence generation can be formulated as reinforcement learning. Similar to (Ranzato et al., 2015), we described it by Markov decision process MDP, where state s is the history consists of the condition and previous words sequence, in our case, $s = < y, x_{1...t-1}^G >$, and an action a is the generated word conditioned on the current state, in our case, a is a word in the vocabulary. Each action is generated according to the policy. The policy is the generator model G. There is a reward function $R(x,y)$ evaluating the goodness of generating x given y. The generator G is learned to maximize the expected reward,

$$G^* = \arg\max_G E_{y\sim P^R(Y), x^G\sim P^G(X|y)}[R(x^G, y)], \tag{3}$$

where $P^R(Y)$ is the probability distribution of condition y in the training data, $P^G(X|y)$ is the probability of generating the sequence x^G given the generator G and condition y, and $R(x,y)$ is the reward function. Here $R(x,y)$ is assumed to be the summation of a sequence of step evaluation reward r_t, where r_t is the reward given to the current state-action pair. r_t evaluates the goodness of generating a specific word x_t^G given the conditions and the generated words $< y, x_{1...t-1}^G >$.

$$R(x^G, y) = \sum_{t=1}^{T} r_t, \tag{4}$$

where T is the length of x^G. It is worth noted that the main difference between equation (2) and equation (3) is that the condition sequence here is $x_{1...t-1}^G$ rather than $x_{1...t-1}^R$. Moreover, each x_t^G here is sampled from softmax rather than argmax over vocabulary set V. As (Ranzato et al., 2015), we use a policy gradient algorithm REINFORCE (Williams, 1992) to tackle the task. The parameters of the generator θ_G is updated as below:

$$\theta_G \leftarrow \theta_G + \eta(\sum_{t'=t}^{T} r_t - b_t)\nabla log(p_G(x_t^G|y, x_{1...t-1}^G)), \tag{5}$$

where b_t is the baseline to reduce variance of training (Sutton & Barto, 1998; Ranzato et al., 2015), and η is the learning rate.

3.3 Generative Adversarial Network

Generative adversarial network (GAN) is composed of a generator and a discriminator (Goodfellow et al., 2014). The aim for discriminator is to discriminate the fed data is from real data or from generator. In the mean time, the generator aims to generate plausible data to confuse the discriminator. Here we use GAN on conditional sequence generation by considering our model G as the generator and constructing a discriminator D sequentially fed with the input-output pair y and x (Mirza & Osindero, 2014; Li et al., 2017).

3.3.1 SeqGAN

Since the vocabulary V in generated sequence is discrete variable, it introduces the difficulty for backpropagation through generator. SeqGAN (Yu et al., 2017; Li et al., 2017) therefore formulates the generation task as reinforcement learning scenario, which is similar to the description in subsection 3.2, and replace the reward function with the discriminator in regular GAN. The discriminator D is then updated through backpropagation, while generator G is updated through policy gradient with reward given by D. The optimization functions for D and G are given as:

$$\begin{aligned} D^* &= \arg\max_D E_{y\sim P^R(Y),x^R\sim P^R(X|y)}[log(D(x^R|y))] + E_{y\sim P^R(Y),x^G\sim P^G(X|y)}[log(1-D(x^G|y))] \\ G^* &= \arg\max_G E_{y\sim P^R(Y),x^G\sim P^G(X|y)}[D(x^G|y)] \end{aligned} \tag{6}$$

G in (6) is optimized by the REINFORCE algorithm. The formulation for optimizing G in (6) is the same as (3), except that $R(x^G, y)$ is replaced with $D(x^G|y)$.

3.3.2 Wasserstein GAN with Gradient Penalty (WGAN-GP)

Recent researches have successfully use WGAP-GP on sequence generation (Gulrajani et al., 2017; Rajeswar et al., 2017; Press et al., 2017). Instead of using more complicate methods, such as policy gradient, they directly fed the softmax layer into discriminator. The generator can therefore be updated through backpropagation. We then formulate the conditional version of WGAN-GP as below:

$$\begin{aligned} D^* &= \arg\max_D E_{y\sim P^R(y),x^R\sim P^R(X|y)}[D(x^R|y)] - E_{y\sim P^R(y),x^G\sim P(X|y)}[D(x^G|y)] \\ G^* &= \arg\max_G E_{y\sim P^R(Y),x^G\sim P^G(X|y)}[D(x^G|y)] \end{aligned} \tag{7}$$

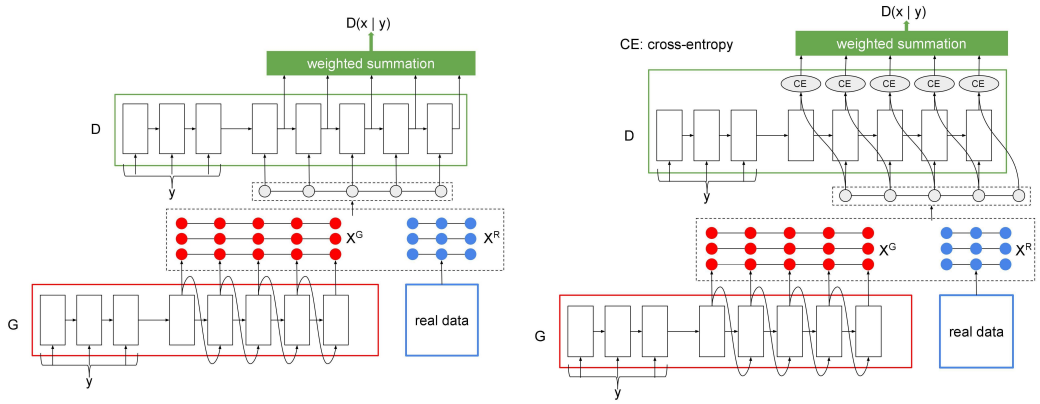
Different from (6), G in (7) is directly optimized by backpropagation.

4 Proposed Approach

Inspired by the actor-critic architecture in (Bahdanau et al., 2016), we thought of predicting D value at each time-step. We proposed two different approaches to accomplish this goal: Every Step GAN (ESGAN) and Energy-based ESGAN (EBESGAN).

4.1 Every Step GAN

The basic idea of every step GAN (see Figure 1a) is to construct a sequence-to-sequence model as discriminator D. At each time-step of the D's decoder, the hidden vector would be passed to a fully-connected layer. The discriminator D then outputs the evaluation score for each subsequence $\langle y, x_{1\ldots t}\rangle$, denoted as $D(x_{1\ldots t}|y)$. The aim for discriminator D is to minimize the summation of $D(x^G_{1\ldots t}|y)$ over the time steps when fed with generated sequences x^G. Simultaneously, D has to maximize the summation of $D(x^R_{1\ldots t}|y)$ when fed with real data x^R. The optimization of D is



(a) Every Step GAN

(b) Energy-based Every Step GAN

Figure 1: Illustration of Every Step GAN (ESGAN) and Energy-based Every Step GAN (EBESGAN).

described as below:

$$D(x|y) = \sum_{t=1}^{T} \alpha_t^D D(x_{1\ldots t}|y)$$
$$D^* = \arg\max_D E_{y\sim P^R(y), x^R\sim P^R(X|y)}[D(x^R|y)] - E_{y\sim P^R(y), x^G\sim P(X|y)}[D(x^G|y)] \tag{8}$$

where α_t^D is a weighted factor with $\sum_{t=1}^{T} \alpha_t^D = 1$. We set $\alpha_t^D = \frac{1}{T}$ in the experiments, but it is possible to give different steps different weights. ESGAN would be a generalized version of SeqGAN. If we set $\alpha_T^D = 1$, and $\alpha_t^D = 0$ for $t < T$, then the proposed approach is equivalent to SeqGAN without MCMC. If we consider $D(x_{1\ldots t}|y)$ as the value obtained by MCMC in SeqGAN, then the proposed approach is the same as SeqGAN with MCMC. However, the proposed approach is more efficient than MCMC because sampling which is very time consuming is not needed in the proposed approach. Also, the REGS algorithm (Li et al., 2017) can also be induced by set $\alpha_t^D = 1$ at a randomly chosen time-step.

The parameters of the generator G, θ_G, is updated by the REINFORCE algorithm. If the objective function of G is as below,

$$G^* = \arg\max_G E_{y\sim P^R(Y), x^G\sim P^G(X|y)}[D(x^G|y)]. \tag{9}$$

Then the formulation for parameter update is as below,

$$\theta_G \leftarrow \theta_G + \eta \alpha_t^G (\sum_{t'=t}^{T} D(x_{1\ldots t'}|y)) \nabla log(p_G(x_t^G|y, x_{1\ldots t-1}^G)). \tag{10}$$

By the factor α_t^G, we can diverse the training by arbitrarily weighting the importance of each time-step[1]. We will explore the influence of different setting of α_t^G in the experiments. We call this ESGAN-Seq below in the experiments. There is another formulation to update θ_G.

$$\theta_G \leftarrow \theta_G + \eta \alpha_t^G D(x_{1\ldots t}|y) \nabla log(p_G(x_t^G|y, x_{1\ldots t-1}^G)), \tag{11}$$

We will compare the two update strategies in the experiments.

4.2 Energy-based ESGAN

The other type of GAN we proposed to predict value for each time-step is energy-based ESGAN (EBESGAN), which is mainly inspired by energy-based GAN (Zhao et al., 2016) but has completely different structure. As depicted in Figure 1b, the discriminator D here has the same architecture as

[1] α_t^G does not have to be equivalent to α_t^D.

the generator G, which is a seq2seq model. The benefit of EBESGAN is that we can use a seq2seq model pre-trained by MLE to initialize the discriminator D. The discriminator predicts probability distribution over vocabulary V. Its aim is to find model D that maximize the likelihood of real data and minimize the likelihood of generated sequence. The generator G, in the mean time, struggle to maximize the likelihood of discriminator D. Therefore, the optimization functions can be written as:

$$\begin{aligned} D^* = &\arg\max_D E_{(x^R,y)\sim P^R(X,Y)}[\sum_{t=1}^{T} log(P^D(x_t^R|y, x_{1\ldots t-1}^R))] \\ &+maximum(0, \beta - E_{y\sim P^R(Y), x^G\sim P^G(X|y)}[\sum_{t=1}^{T} log(P^D(x_t^G|y, x_{1\ldots t-1}^G))]) \end{aligned} \tag{12}$$

$$G^* = \arg\max_G E_{y\sim P^R(Y), x^G\sim P^G(X|y)}[\sum_{t=1}^{T} log(P^D(x_t^G|y, x_{1\ldots t-1}^G))] \tag{13}$$

where β is the threshold for preventing the discriminator D distinguishes generated data from real data too easily. $P^D(x_t^G|y, x_{1\ldots t-1}^G)$ in (4.2) is the probability of generating x_t^G given $< y, x_{1\ldots t-1}^G >$ based on the seq2seq model of D. While $P(X^G|Y)$ is far from $P(X^R|Y)$, the threshold would then turn off the second term in equation . The aim of discriminator D is therefore the same as maximum-likelihood estimation. Equation then lead the generator to reproduce the distribution that has been learned by discriminator.

5 EXPERIMENTS

We use recurrent neural network for both the discriminator and generator due to its strong sequential correlation (Press et al., 2017; Rajeswar et al., 2017). Specifically, gated recurrent unit (GRU) (Chung et al., 2014) is used in our experiments. The noise feature in GAN is considered as the random process of sampling from softmax layer distribution.

5.1 ARTIFICIAL GRAMMARS

To better analyze GANs on conditional sequence generation, we define three artificial grammars: sequence, counting and addition. The description of the three grammars are listed in Table 1. For **sequence** grammar, the aim is to generate a continued consecutive number sequence behind the input Y. For example, when the input is $\langle 1, 2, 3\rangle$, the answer would be any length of consecutive number sequence started with 4, such as $\langle 4, 5, 6, 7, 8\rangle$. The target of **counting** grammar is more complicated. The generated sequence should contain exactly 3 words, where the median is a random selected word from input sequence. The first generated word should be the number of words on the left-hand side of the selected median, while the last generated word should be the number of words on the right-hand side. For example, when the input is $\langle 5, 9, 2, 8, 3, 2, 9, 1\rangle$, one of the generated sequence can be $\langle 0, 5, 7\rangle$. Last, the task of **addition** grammar aims to generate the addition of two number that are randomly segmented from input sequence. That is, if input is $\langle 8, 1, 3, 4\rangle$, then the output can be the addition of 8 and 134 and therefore $\langle 1, 4, 2\rangle$.

The purpose of these design is to imitate some major properties in dialogue generation, such as: variable-length, repeated beginning-sequence, the same sequence space, one-to-many, and many-to-one. The variable-length property means there is no fixed length for the input and output sequences. The repeated beginning-sequence property means the beginning subsequences are usually shared by many data. Such as *"What"* and *"I am"* in natural language. The same sequence space here means that the input and output have the same structure and therefore are sampled from the same space. Finally, **One-to-many** and **many-to-one** are quite common in dialogue generation. For example, when asking *"How are you?"*, the response can varied from *"I'm fine."* to *"Great! How do you do?"*. Also, the same response can be paired with multiple question, such as that *"My name is Paul."* can answer *"What's your name?"* and *"Who are you?"*.

We then randomly generate 100,000 samples as training data, 10,000 as development data, and 10,000 samples as testing data. The architectures are set to 1 layer with 128 hidden units. Since the

Table 1: Simple definition and examples.

GRAMMAR	DEFINITION	EXAMPLES
Sequence	continue the sequence with a random length	123 : 4, 45, ...
Counting	randomly choose a digit, and then count the left-hand length and right-hand length	123 : 012, 121, 230
Addition	random partition, and then add the two number	123 : 15, 24

Table 2: ESGAN with different weighted factor. The dash(-) here notes that the weighted factor do not introduce improvement nor deterioration.

		VLGAN-RL			VLGAN-VRL		
		Acc	**AccS**	**Cov**	**Acc**	**AccS**	**Cov**
Sequence	uniform	97.19	75.92	66.08	97.11	74.85	67.49
	increase	97.17	75.33	66.75	97.18	73.30	68.19
	decrease	97.16	74.73	67.14	97.09	74.34	67.48
Counting	uniform	77.23	71.43	70.18	74.91	70.70	69.84
	increase	74.50	69.66	70.86	74.54	**70.74**	**70.11**
	decrease	**81.98**	**72.24**	69.02	**75.47**	70.64	69.82
Addition	uniform	44.26	32.39	**31.91**	44.77	32.34	31.65
	increase	-	-	-	-	-	-
	decrease	**44.94**	**32.19**	31.67	**45.55**	**32.49**	**32.01**

artificial grammars are easy to evaluate, we use three index to evaluate our results listed in Table 3. The first one is the accuracy of samples generated from argmax policy (Acc). The second one is the accuracy of samples generated from softmax probability (AccS). The last one is the coverage of softmax samples over all the probable answers of the specific grammar (Cov). To all inputs in testing data, we generate 50 softmax samples for evaluation. Among the 50 samples, the correct ones would be collected for evaluating the coverage. We observe that the last two indexes haven't been report in previous works. We report them for the purpose to evaluate if the one-to-many property can be learned. For fair comparison, all the algorithms are based on the same pre-trained model, which is the MLE model listed in Table 3[2].

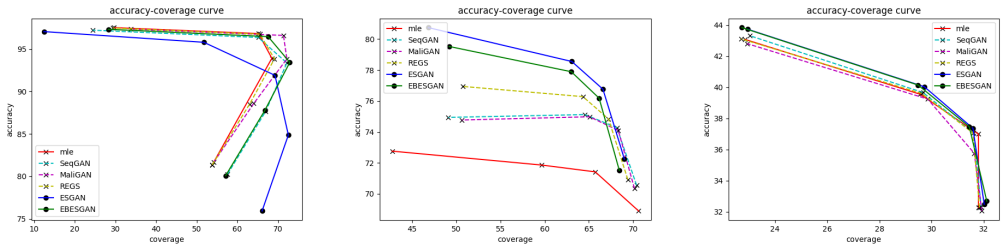
The result listed in Table 2 is the comparison of different weighted factor for ESGAN. In practice, we have found that the setting of α_t^D can actually be included in α_t^G. Therefore, we set $\alpha_t^D = \frac{1}{T}$ for all the presented cases. As depicted in Table 2, all the three grammars are trained using three sorts of weighted factors: uniform ($\alpha_t^G = 1$), increase($\alpha_t^G = t$), decrease($\alpha_t^G = T - t + 1$). The results clearly present that time-step-decreased weighted factor can positively affect the training. We consider the reason is that the training spirit of decreased weighted factor start from first correcting beginning subsequence, which is similar to curriculum learning. We have also conducted a weighted version of EBESGAN, but the results turn out having few impacts on all the three grammars.

The best weighted factors for ESGAN and ESGAN-Seq are then selected to compare with other algorithms in Table 3. We found that in our architecture, using SGD optimizer for generator and Adam optimizer for discriminator would facilitate both the training speed the performance. In sequence grammar, all the GAN-based methods introduce higher coverage but simultaneously reduce the accuracy. To investigate their influences of the accuracy and coverage, we plot the accuracy-coverage curve by sharpening the softmax layer. The Figure 2a shows that they have inconsistent advantages along the curve. Therefore, we summarize there are no positive or negative impacts on sequence grammar, but only changed to another curve. On the other hand, we can easily find that ESGAN (include ESGAN-Seq), and EBESGAN can outperform the previous algorithms, especially in argmax accuracy, directly from Tabel 3. To better understand the influence of coverage, one can see the curves in Figure 2b and Figure 2c. In both grammars, the differences are more significant

[2]The accuracy of REINFORCE here is considered to be the upper bound, since the given reward is the true accuracy.

Table 3: Results of artificial grammars with different back-propagation methods. We define the evaluation label Acc(%) as the accuracy of argmax samples, AccS(%) as the accuracy of softmax samples, and Cov(%) as the coverage of softmax samples over probable answers. The dash(-) here note that the algorithm can not introduce improvement based on the pre-trained model.

	Sequence			**Counting**			**Addition**		
	Acc	**AccS**	**Cov**	**Acc**	**AccS**	**Cov**	**Acc**	**AccS**	**Cov**
MLE	**97.43**	81.32	53.77	73.48	68.89	**70.63**	44.57	32.28	31.79
REINFORCE	*99.81*	*97.30*	*4.54*	*99.97*	*99.36*	*16.99*	*79.98*	*75.60*	*18.32*
WGAN-GP	-	-	-	-	-	-	-	-	-
MaliGAN	97.34	**81.66**	54.19	74.12	70.35	70.15	44.27	32.05	31.92
SeqGAN	97.20	80.28	57.61	74.59	70.56	70.39	44.87	32.30	31.90
MC-SeqGAN	97.20	80.98	55.49	72.96	68.10	70.54	44.72	32.28	31.83
REGS	97.42	81.36	53.82	75.99	70.93	69.40	44.64	32.32	32.01
ESGAN	97.19	75.92	66.08	**81.98**	**72.24**	69.02	44.94	32.19	31.67
ESGAN-Seq	97.11	74.85	**67.49**	75.47	70.64	69.82	45.55	32.49	32.01
EBESGAN	97.32	80.09	57.12	79.78	71.52	68.44	**45.74**	**32.68**	**32.11**



(a) Sequence (b) Counting (c) Addition

Figure 2: Sampled accuracy and coverage curve.

when sampling from a sharper distribution. Moreover, the coverage and softmax accuracy are both apparently improved.

5.2 Dialogue Generation

We use OpenSubtitle (Tiedemann, 2009) as the training data and with a vocabulary of top 4,000 frequently occurring words. Both the generator and discriminator are 1-layer GRU with hidden dimension set to 1024. To compare the improvements introduced by all the algorithms, we first pre-train the generator with MLE. And then randomly choose 32,000 examples from the training data for further modifying the model. Time-comparable algorithms such as SeqGAN, ESGAN and EBESGAN are then trained upon the pre-train model using the training data subset. The discriminator of SeqGAN and ESGAN are pre-trained by real data and generated data from the pre-trained generator. The discriminator of EBESGAN specifically uses the same pre-trained model as generator. Note that these models are all trained without any other auxiliary tricks such like: MIXER, curriculum learning, teacher forcing.

For human evaluation, we randomly select 20 inputs from development set. Each model then generate 1 argmax sample and 2 softmax samples according to an input. We present both an input and generated outputs to 16 judges. They are asked to score each generated output from 1 to 5. We then normalize each judge's scores, and calculate two indexes from the results: Acc and AccS. Acc is the average scores that the argmax samples obtain. The AccS is the average scores of all the 3 generated samples obtained. In addition, we evaluate how smooth the softmax layer is by averaging the log probability of 10 softmax samples for each of 1,000 inputs randomly selected from development set. That is, a lower average log probability means the distribution of softmax layer tend to be more smooth. We also want to evaluate their coverage, though there do not exist complete reference set for dialogue generation. Specifically, that a model simultaneously has a lower average log probability

Table 4: Argmax examples of dialogue generation.

Input	I don' t wanna know your name , your story .
MLE	I ' m not a _UNK .
SeqGAN	You ' re a _UNK _UNK .
ESGAN	You wanna know what I mean ?
EBESGAN	I ' m not interested .
Input	We decide when we die and how .
MLE	We don ' t know .
SeqGAN	We don ' t want to die .
ESGAN	No , you don ' t .
EBESGAN	We ' re not leaving .

Table 5: Human evaluation and average log probability of dialogue generation.

	Acc(%)	**AccS(%)**	**AvgLogProb**
MLE	46.92	52.61	-2.201
SeqGAN	48.20	56.89	-2.383
ESGAN	**54.38**	**60.92**	-2.281
EBESGAN	43.83	46.61	-2.652

and higher softmax accuracy points out this model do have a wider coverage. That's because while it generates a wide range of samples, most of it's coverage is actually good.

In Table 5, the ESGAN outperforms the other three models in both Acc and AccS while having a lower AvgLogProb than MLE. This indicates that SeqGAN and ESGAN can find a better coverage than MLE. When interacting with the model, it can generate more number of proper responses to an input. On the other side, although the EBESGAN achieves better performance in artificial grammars, it diverges from the MLE model in dialogue generation task with lower AccS and much lower AvgLogProb.

6 Conclusion

The focus of this paper is to demonstrate the essence of GANs on conditional sequence generation. Therefore the experiments conducted without other auxiliary approaches, such as the notable teacher forcing and curriculum learning. We then clearly find that our proposed models ESGAN and EBESGAN can equally perform or outperform the current GANs using policy gradient scheme on the artificial grammars. And ESGAN significantly outperforms others of the human evaluation on dialogue generation task.

In addition, our proposed artificial grammars can not only accurately evaluate the coverage and accuracy, but also have the advantage for clearly distinguishable style. For example, the style in sequence can be the length, the style in counting can be the selected position of digit, the style in addition can be the selected position for partition. This property is suitable for investigating style transferring of sequences, which would be one of our aims in future works.

References

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

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

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

Kirthevasan Kandasamy, Yoram Bachrach, Ryota Tomioka, Daniel Tarlow, and David Carter. Batch policy gradient methods for improving neural conversation models. *arXiv preprint arXiv:1702.03334*, 2017.

Matt J Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. Adversarial ranking for language generation. *arXiv preprint arXiv:1705.11001*, 2017.

Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*, 2016.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, and Lior Wolf. Language generation with recurrent generative adversarial networks without pre-training. *arXiv preprint arXiv:1706.01399*, 2017.

Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. Adversarial generation of natural language. *arXiv preprint arXiv:1705.10929*, 2017.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Jörg Tiedemann. News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov (eds.), *Recent Advances in Natural Language Processing*, volume V, pp. 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria, 2009. ISBN 978 90 272 4825 1.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pp. 2852–2858, 2017.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. *arXiv preprint arXiv:1706.03850*, 2017.

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.