Text Generation Based on Generative Adversarial Nets with Latent Variable

Heng Wang[1], Zengchang Qin[1*], Tao Wan[2*]

[1]Intelligent Computing and Machine Learning Lab
School of ASEE, Beihang University, Beijing, 100191, China
[2]School of Biological Science and Medical Engineering
Beihang University, Beijing, 100191, China
*{zcqin, taowan}@buaa.edu.cn

Abstract. In this paper, we propose a model using generative adversarial net (GAN) to generate realistic text. Instead of using standard GAN, we combine variational autoencoder (VAE) with generative adversarial net. The use of high-level latent random variables is helpful to learn the data distribution and solve the problem that generative adversarial net always emits the similar data. We propose the VGAN model where the generative model is composed of recurrent neural network and VAE. The discriminative model is a convolutional neural network. We train the model via policy gradient. We apply the proposed model to the task of text generation and compare it to other recent neural network based models, such as recurrent neural network language model and SeqGAN. We evaluate the performance of the model by calculating negative log-likelihood and the BLEU score. We conduct experiments on three benchmark datasets, and results show that our model outperforms other previous models.

Keywords: Generative Adversarial Net; Variational Autoencoder; VGAN; Text Generation

1 Introduction

Automatic text generation is important in natural language processing and artificial intelligence. For example, text generation can help us write comments, weather reports and even poems. It is also essential to machine translation, text summarization, question answering and dialogue system [1]. One popular approach for text generation is by modeling sequence via recurrent neural network (RNN) [1]. However, recurrent neural network language model (RNNLM) suffers from two major drawbacks when used to generate text. First, RNN based model is always trained through maximum likelihood approach, which suffers from exposure bias [12]. Second, the loss function used to train the model is at word level but the performance is typically evaluated at sentence level. There are some research on using generative adversarial net (GAN) to solve the problems. For example, Yu *et al.* [9] applies GAN to discrete sequence generation by directly optimizing the discrete discriminator's rewards. Li *et al.* [22] applies GAN to open-domain dialogue generation and generates higher-quality responses. Instead of directly optimizing the GAN objective, Che *et al.* [21] derives a novel and low-variance objective using the discriminator's output that follows corresponds to the log-likelihood. In GAN, a discriminative net D is learned to distinguish whether a given data instance is real or not, and a generative net G is learned to confuse D by generating highly realistic data. GAN has achieved a great success in computer vision tasks [13], such as image style transfer [24], super resolution and imagine generation [23]. Unlike image data, text generation is inherently discrete, which makes the gradient from the discriminator difficult to back-propagate to the generator [19]. Reinforcement learning is always used to optimize the model when GAN is applied to the task of text generation [9].

Although GAN can generate realistic texts, even poems, there is an obvious disadvantage that GAN always emits similar data [14]. For text generation, GAN usually uses recurrent neural network as the generator. Recurrent neural network mainly contains two parts: the state transition and a mapping from the state to the output, and two parts are entirely deterministic. This could be insufficient to learn the distribution of highly-structured data, such as text [3]. In order to learn generative models of sequences, we propose to use high-level latent random variables to model the observed variablity. We combine recurrent neural network with variational autoencoder (VAE) [2] as generator G.

In this paper, we propose a generative model, called VGAN, by combining VAE and generative adversarial net to better learn the data distribution and generate various realistic text. The paper is structured as the following: In Section 2, we give the preliminary of this research. In Section 3, we introduce the VGAN model and adversarial training of VGAN. Experimental results are given and analyzed in Section 4. In Section 5, we conclude this research and discuss the possible future work.

2 Preliminary

2.1 LSTM Architecture

A recurrent neural network is a class of artificial neural network where connections between units form a directed cycle [1]. This allows it to exhibit dynamic temporal behavior. Long short-term memory (LSTM) is an improved version of recurrent neural network considering long-term dependency in order to overcome the vanishing gradient problem. It has been successfully applied in many tasks, including text generation and speech recognition [20]. LSTM has a architecture consisting of a set of recurrently connected subnets, known as memory blocks. Each block contains memory cells and three gate units, including the input, output and forget gates. The gate units allow the network to learn when to forget previous information and when to update the memory cells given new information.

Given a vocabulary V and an embedding matrix $W \in R^{q \times |V|}$ whose columns correspond to vectors; $|V|$ and q denote the size of vocabulary and the dimension of the token vector, respectively. The embedding matrix W can be initialized randomly or pretrained. Let x denote a token with index k, $e(x) \in R^{|V| \times 1}$ is a vector with zero in all positions except $R_k^{|V| \times 1} = 1$. Given an input sequence $X_s = (x_1, x_2, \cdots, x_T)$, we compute the output sequence of LSTM $Y_s = (y_1, y_2, \cdots, y_T)$. When the input sequence passes through the embedding layer, each token is represented by a vector: $v_i = W \cdot e(x_i) \in R^{q \times 1}$. The relation between inputs, memory cells and outputs are defined by the following equations:

$$i^{(t)} = \sigma(W_{ix} v^{(t)} + W_{ih} h^{(t-1)} + W_{ic} c^{(t-1)}) \tag{1}$$

$$f^{(t)} = \sigma(W_{fx} v^{(t)} + W_{fh} h^{(t-1)} + W_{fc} c^{(t-1)}) \tag{2}$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot tanh(W_{cx} v^{(t)} + W_{ch} h^{(t-1)}) \tag{3}$$

$$o^{(t)} = \sigma(W_{ox} v^{(t)} + W_{oh} h^{(t-1)} + W_{oc} c^{(t-1)}) \tag{4}$$

$$h^{(t)} = o_{(t)} \odot tanh(c^t) \tag{5}$$

where $i^{(t)} \in R^{l \times 1}$, $f^{(t)} \in R^{l \times 1}$, $o^{(t)} \in R^{l \times 1}$ and $h^{(t)} \in R^{l \times 1}$ represent the input gate, forget gate, output gate, memory cell activation vector and the recurrent hidden state at time step t; l is the dimension of LSTM hidden units, σ and $tanh$ are the logistic sigmoid function and hyperbolic tangent function, respectively. $\odot$ represents element-wise multiplication [20].

2.2 Variational Autoencoder

An autoencoder (AE) is an unsupervised learning neural network with the target values to be equal to the inputs. Typically, AE is mainly used to learn a representation for the input data, and extracts features and reduces dimensionality [2]. Recently, autoencoder has been widely used to be a generative model of image and text. The variational autoencoder is an improved version based on the standard autoencoder. For variational autoencoder, there is a hypothesis that data is generated by a directed model and some latent variables are introduced to capture the variations in the observed variables. The directed model $p(x) = \int p(x|z)p(z)\, dz$ is optimized by using a variational upper bound:

$$\begin{aligned} -\log p(x) &= -\log \int p(x|z)p(z)\, dz \\ &\leq -KL(q(z|x)||p(z)) + E_{z \sim q(z|x)}[\log p(x|z)] \end{aligned} \tag{6}$$

where $p(z)$ is a prior distribution over the latent random variable z; The prior distribution is unknown, and we generally assume it to be a normal distribution. $p(x|z)$ denotes a map from the latent variables z to the observed variables x, and given z it produces a distribution over the possible corresponding values of x. $q(z|x)$ is a variational approximation of the true posterior distribution. let $KL(q(z|x)||p(z))$ denote the Kullback-Leibler divergence between $q(z|x)$ and $p(z)$; The introduction of latent variables makes it intractable to optimize the model directly. We minimize the upper bound of the negative log-likelihood to optimize VAE. The training algorithm we use is Stochastic Gradient Variational Bayes (SGVB) proposed in [2].

2.3 Generative Adversarial Nets

For generative adversarial nets, there is a two-sided zero-sum game between a generator and a discriminator. The training objective of the discriminative model is to determine whether the data is from the fake data generated by the generative model or the real training data. For the generative model, its objective is to generate realistic data, which is similar to the true training data and the discriminative model can't distinguish. For the standard generative adversarial networks, we train the discriminative model D to maximize the probability of giving the correct labels to both the samples from the generative model and training examples. We simultaneously train the generative model G to minimize the estimated probability of being true by the discriminator. The objective function is:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (7)$$

where z denotes the noises and $G(z)$ denotes the data generated by the generator. $D(x)$ denotes the probability that x is from the training data with emperical distribution $p_{data}(x)$.

3 Model Description

3.1 The Generative Model of VGAN

The proposed generative model contains a VAE at every timestep. For the standard VAE, its prior distribution is usually a given standard normal distribution. Unlike the standard VAE, the current prior distribution depends on the hidden state h_{t-1} at the previous moment, and adding the hidden state as an input is helpful to alleviate the long term dependency of sequential data. It also takes consideration of the temporal structure of the sequential data [3] [4]. The model is described in Fig. 1. The prior distribution $z_t = p_1(z_t|h_{t-1})$ is:

$$z_t \sim N(\mu_{0,t}, \sigma^2_{0,t}), \quad where \quad [\mu_{0,t}, \sigma^2_{0,t}] = \varphi^{prior}(h_{t-1}) \quad (8)$$

where $\mu_{0,t}$, $\sigma^2_{0,t}$ are the mean and variance of the prior Gaussian distribution, respectively. The posterior distribution depends on the current hidden state. For the appriximate posterior distribution, it depends on the current state h_t and the current input x_t: $z'_t = q_1(z_t|x_t, h_t)$.

$$z'_t \sim N(\mu_{1,t}, \sigma^2_{1,t}), \quad where \quad [\mu_{1,t}, \sigma^2_{1,t}] = \varphi^{posterior}(x_t, h_t) \quad (9)$$

where $\mu_{1,t}$, $\sigma^2_{1,t}$ are the mean and variance of the approximate posterior Gaussian distribution, respectively. φ^{prior} and $\varphi^{posterior}$ can be any highly flexible functions, for example, a neural network.

The derivation of the training criterion is done via stochastic gradient variational Bayes. We achieve the goal of minimizing the negative log-likelihood in

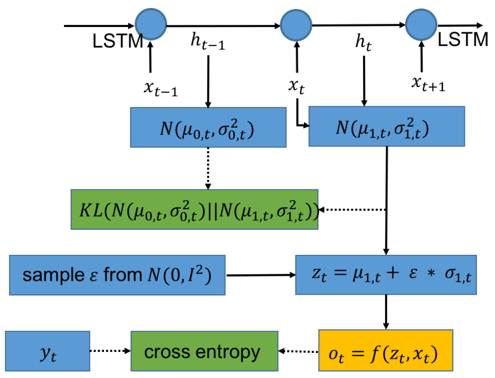

LSTM
h_{t-1}
h_t
LSTM
x_{t-1}
x_t
x_{t+1}
$N(\mu_{0,t}, \sigma_{0,t}^2)$
$N(\mu_{1,t}, \sigma_{1,t}^2)$
$KL(N(\mu_{0,t}, \sigma_{0,t}^2)||N(\mu_{1,t}, \sigma_{1,t}^2))$
sample ε from $N(0, I^2)$
$z_t = \mu_{1,t} + \varepsilon * \sigma_{1,t}$
y_t
cross entropy
$o_t = f(z_t, x_t)$

Fig. 1. The structure of the generator G_θ. The generator is composed of LSTM and VAE. x_t denotes the input at timestep t; h_t denotes the LSTM hidden state; $N(\mu_{0,t}, \sigma_{0,t}^2)$ denotes the prior distribution; $N(\mu_{1,t}, \sigma_{1,t}^2)$ denotes the appriximate posterior distribution; y_t denotes the target output at timestp t, which is the same as x_{t+1}. o_t denote the estimated result. The dotted line denotes the optimization process in the pre-training stage.

the pre-training stage by minimizing $L(x_{1:T})$:

$$L(x_{1:T}) = -\log p(x_{1:T}) = -\log \int_{z_{1:T}} \frac{q_1(z_{1:T}|x_{1:T}, h_{1:T})}{q_1(z_{1:T}|x_{1:T}, h_{1:T})} \prod_{t=0}^{T-1} p(x_{t+1}|x_{1:t}, z_{1:t})\, dz_{1:T}$$
$$\leq -KL(q_1(z_{1:T}|x_{1:T}, h_{1:T})||p_1(z_{1:T}|x_{1:T-1}, h_{1:T-1}))$$
$$+E_{z_{1:T}\sim q_1(z_{1:T}|x_{1:T}, h_{1:T})}\left[\sum_{t=0}^{T-1} \log p(x_{t+1}|x_{1:t}, z_{1:t})\right] \tag{10}$$

where p_1 and q_1 represent the prior distribution and the approximate posterior distribution.

If we directly use the stochastic gradient descent algorithm to optimize the model, there will be a problem that some parameters of the VAE are not derivable. In order to solve the problem, we introduce the "reparametrization trick" [2]. For example, if we want to get the samples from the distribution $N(\mu_{1,t}, \sigma_{1,t}^2)$, we will sample from a standard normal distribution $\epsilon \sim N(0, I^2)$ and get the samples z'_t via $z'_t = \mu_{1,t} + \sigma_{1,t}\epsilon$. Before the adversarial training, we need to pre-train the generative model via SGVB. For example, given the input $X_s = (\mathbf{S}, i, like, it)$, and the target output is $Y_s = (i, like, it, \mathbf{E})$, where $\mathbf{S}$ and $\mathbf{E}$ are the start token and the end token of a sentence, respectively.

3.2 Adversarial Training of VGAN

In this paper, we choose convolutional neural network as the discriminative model, which has shown a great success in the task of text classification [5] [6].

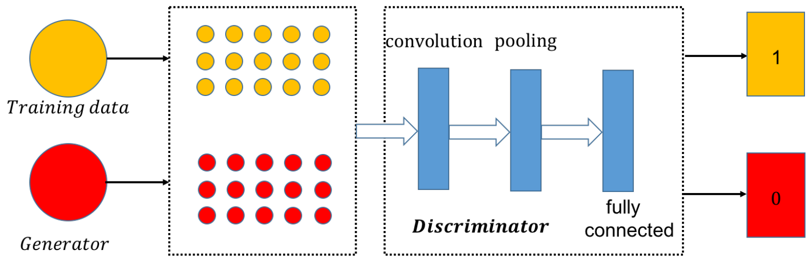


Fig. 2. The illustration of the discriminator. The discriminator D_ϕ is trained by using the true training data and the fake data generated by generator. The discriminator contains the convolution layer, the max-pooling layer and the fully connected layer.

Let $v_i \in R^k$ denote a k-dimesion vector corresponding to the i-th word in the sentence. $v_{1:n} = v_1 \oplus v_2 \oplus \cdots \oplus v_n$ denote a sentence of length n, where $\oplus$ is the concatenation operator and $v_{1:n} \in R^{n \times k}$ is a matrix. Then a filter $w_1 \in R^{h \times k}$ is applied to a window of h words to produce a new feature. For example, $c_i = f(w_1 \cdot v_{i:i+h-1} + b)$ where c_i is a feature generated by convolution operation. f denotes a nonlinear function such as the hyperbolic tangent or sigmoid; b is a bias term. When the filter is applied to a sentence $v_{1:n}$, a feature map $\mathbf{c} = [c_1, c_2, \cdots, c_{n-h+1}]$ is generated. We can use a variety of convolution kernels to obtain a variety of feature maps. We apply a max-over-time pooling operation to the features to get the maximum value $c' = \max\{\mathbf{c}\}$. Finally, all features are used as input to a fully connected layer for classification. After the generator is pre-trained, we use the generator to generate the negative examples. The negative samples and the true training data are combined as the input of the discriminator. The training process of discriminator is showed in Fig. 2. In the adversarial training, the generator G_θ is optimized via policy gradient, which is a reinforcement learning algorithm [7] [8]. The training process is showed in Fig. 3. G_θ can be viewed as an *agent*, which interacts with the environment. The parameters θ of this *agent* defines a policy, which determines the process of generating the sequences.

Given a start token **S** as the input, G_θ samples a token from the generating distribution. And the sampled word is as the input at the next time. A whole sentence is generated word by word until an end token **E** has been generated or the maximum length is reached. For example, given a start token **S** as the input, the sequence $Y_{1:T} = (y_1, y_2, \cdots, y_T)$ is generated by G_θ. In timestep t, the state s_t is the current produced tokens $(y_1, y_2, \cdots, y_{t-1})$ and the action a_t is to select the next token in the vocabulary. After taking an action a_t, the agent updates the state $(y_1, y_2, \cdots, y_t)$. If the agent reaches the end, the whole sequence has been generated and the reward will be assigned. During the training, we choose the next token according to the current policy and the current state. But there is a problem that we can observe the reward after a whole sequence. When G_θ generates the sequence, we actually care about the expected accumulative

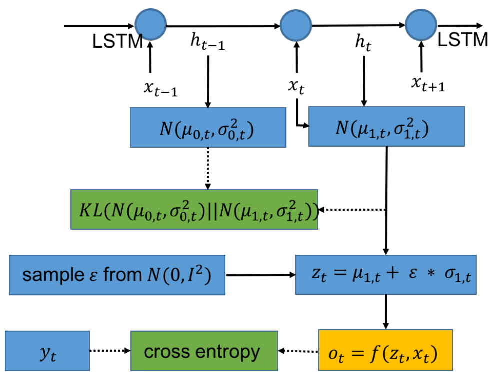

LSTM
h_{t-1}
h_t
LSTM
x_{t-1}
x_t
x_{t+1}
$N(\mu_{0,t},\sigma_{0,t}^2)$
$N(\mu_{1,t},\sigma_{1,t}^2)$
$KL(N(\mu_{0,t},\sigma_{0,t}^2)||N(\mu_{1,t},\sigma_{1,t}^2))$
sample ε from $N(0,I^2)$
$z_t = \mu_{1,t} + \varepsilon * \sigma_{1,t}$
y_t
cross entropy
$o_t = f(z_t, x_t)$

Fig. 3. The training process of the generator via policy gradient. The dotted line denotes sampling a token from the output distribution. The sampled token is as the input at the next time. **START** denotes a start token.

reward from start to end and not only the end reward. At every timestep, we consider not only the reward brought by the generated sequence, but also the future reward.

In order to evaluate the reward for the intermediate state, Monte Carlo search has been employed to sample the remaining unknown tokens. In result, for the finished sequences, we can directly get the rewards by inputting them to the discriminator D_ϕ. For the unfinished sequences, we first use the Monte Carlo search to get estimated rewards [25]. To reduce the variance and get more accurate assessment of the action value, we employ the Monte Carlo search for many times and get the average rewards. The objective of the generator model G_θ is to generate a sequence from the start token to maximize its expected end reward:

$$\max_\theta J(\theta) = E[R_T|s_0] = \sum_{y_1 \in V} G_\theta(y_1|s_0) \cdot Q_{G_\theta}^{D_\phi}(s_0, y_1) \tag{11}$$

where R_T denotes the end reward after a whole sequence is generated; $Q_{G_\theta}^{D_\phi}(s_i, y_i)$ is the action-value function of a sequence, the expected accumulative reward starting from state s_i, taking action $a = y_i$; $G_\theta(y_i|s_i)$ denotes the generator chooses the action $a = y_i$ when the state $s_i = (y_1, y_2, \cdots, y_{i-1})$ according to the policy. The gradient of the objective function $J(\theta)$ can be derived [9] as:

$$\nabla_\theta J(\theta) = E_{Y_{1:t-1} \sim G_\theta}\left[\sum_{y_t \in V} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q_{G_\theta}^{D_\phi}(Y_{1:t-1}, y_t)\right] \tag{12}$$

we then update the parameters of G_θ as:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta) \tag{13}$$

where α is the learning rate.

After training the generator by using policy gradient, we will use the negative samples generated by the updated generator to re-train the discriminator D_ϕ as follows:

$$\min_{\phi} -E_{Y\sim p_{data}}[\log D_\phi(Y)] - E_{Y\sim G_\theta}[\log(1 - D_\phi(Y))] \tag{14}$$

The pseudo-code of the complete training process is shown in Algorithm **1**.

Algorithm 1. Generative Adversarial Network with Latent Variable

```
Require: generaror: $G_\theta$;
      discriminator: $D_\phi$;
      True training dataset: $X$;
1 :  Pre-train $G_\theta$ on $X$ by Eq. (10);
2 :  Generate negative samples Y by using $G_\theta$;
3 :  Pre-train $D_\phi$ on $X$ and $Y$ by minimizing the cross entropy;
4 :  Repeat:
5 :    for $1 \sim$ m do
6 :      generate the data $Y_1$ by using $G_\theta$;
7 :      use $D_\phi$ to get the reward of $Y_1$;
8 :      update the parameters $\theta$ of $G_\theta$ by Eq. (13);
9 :    end for
10:    for $1 \sim$ n do
11:      use $G_\theta$ to generate negative samples $Y_2$;
12:      update the parameters $\phi$ of $D_\phi$ on $X$ and $Y_2$ by Eq. (14);
13:    end for
14:  Until: VGAN converges.
```

In this paper, we propose the VGAN model by combining VAE and GAN for modelling highly-structured sequences. The VAE can model complex multimodal distributions, which will help GAN to learn structured data distribution.

4 Experimental Studies

In our experiments, given a start token as input, we hope to generate many complete sentences. We train the proposed model on three datasets: Taobao Reviews, Amazon Food Reviews and PTB dataset [15]. Taobao dataset is crawled on taobao.com. The sentence numbers of Taobao dataset, Amazon dataset are 400K and 300K, respectively. We split the datasets into 90/10 for training and test. For PTB dataset is relatively small, the sentence numbers of training data and test data are 42,068 and 3,370.

4.1 Training Details

In our paper, we compare our model to two other neural models RNNLM [1] and SeqGANs [9]. For these two models, we use the random initialized word embeddings, and they are trained at the level of word. We use 300 for the dimension of LSTM hidden units. The size of latent variables is 60. For Taobao Reviews and PTB dataset, the sizes of vocabulary are both 5K. For Amazon Food Reviews, the size is 20K. The maximum lengths of Taobao Reviews, PTB dataset and Amazon Food Reviews are 20, 30 and 30, respectively. We drop the LSTM hidden state with the dropout rate 0.5. All models were trained with the Adam optimization algorithm [10] with the learning rate 0.001. First, we pre-train the generator and the discriminator. We pre-train the generator by minimizing the upper bound of the negative log-likelihood. We use the pre-trained generator to generate the negative data. The negative data and the true training data are combined to be the input of the discriminator. Then, we train the generator and discriminator iteratively. Given that the generator has more parameters and is more difficult to train than the discriminator, we perform three optimization steps for the discriminator for every five steps for the generator. The process is repeated until a given number of epochs is reached.

Table 1. BLEU-2 score on three benchmark datasets. The best results are highlighted.

Numbers of generated sentence	200	400	600	800	1000
RNNLM (Taobao)	0.965	0.967	0.967	0.967	0.967
SeqGAN (Taobao)	0.968	0.970	0.970	0.968	0.968
VGAN-pre (Taobao)	0.968	0.968	0.967	0.968	0.968
VGAN (Taobao)	**0.969**	**0.972**	**0.970**	**0.969**	**0.969**
RNNLM (Amazon)	0.831	0.842	0.845	0.846	0.848
SeqGAN (Amazon)	0.846	0.851	0.852	0.853	0.856
VGAN-pre (Amazon)	0.842	0.849	0.854	0.849	0.848
VGAN (Amazon)	**0.876**	**0.874**	**0.866**	**0.868**	**0.868**
RNNLM (PTB)	0.658	0.650	0.654	0.655	0.662
SeqGAN (PTB)	0.712	0.705	0.701	0.702	0.681
VGAN-pre (PTB)	0.680	0.690	0.694	0.695	0.671
VGAN (PTB)	**0.715**	**0.709**	**0.714**	**0.715**	**0.695**

4.2 Results and Evaluation

In this paper, we use the BLEU score [17] and negative log-likelihood as the evaluation metrics. BLEU score is used to measure the similarity degree between the generated texts and the human-created texts. We use the whole test data as the references when calulating the BLEU score via nature language toolkit (NLTK). For negative log-likelihood, we calulate the value by inputting the test data. Table 2 shows the NLL values of the test data. VGAN-pre is the pretrained model of VGAN.

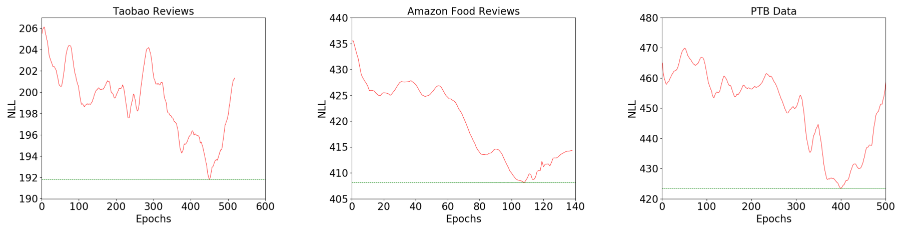
$$NLL = -E\left[\sum_{t=1}^{T} \log G_\theta(y_t|Y_{1:t-1})\right] \tag{15}$$

where NLL denotes the negative log-likelihood; Table 1 shows the experimental results of BLEU-2 score, and numbers in Table 1 denote the numbers of the sentence generated. We calulate the average BLEU-2 score between the generated sentences and the test data. The descent processes of NLL values during the adversarial training are showed in the Fig 4. Here, we give some examples generated by the proposed model. Due to the page limit, only some of generated comments from Amazon Food Reviews are shown in Table 3 and more results will be available online in the final version of the paper.

Table 2. The comparison results (NLL) of VGAN to other models.

Dataset	Taobao Dataset	Amazon Dataset	PTB Dataset
RNNLM [1]	219	483	502
SeqGAN [9]	212	467	490
VGAN-pre	205	435	465
VGAN	**191**	**408**	**423**

From Table 1 and Table 2, we can see the significant advantage of VGAN over RNNLM and SeqGAN in both metrics. The results in the Fig 4 indicate that applying adversarial training strategies to generator can breakthrough the limitation of generator and improve the effect.



(a) Taobao Reviews (b) Amazon Food Reviews (c) PTB Data

Fig. 4. (a) NLL values of Taobao Reviews. (b) NLL values of Amazon Food Reviews. (c) NLL values of PTB Data.

5 Conclusions

In this paper, we proposed the VGAN model to generate realistic text based on classical GAN model. The generative model of VGAN combines generative adversarial nets with variational autoencoder and can be applied to the sequences

Table 3. Generated examples from Amazon Food Reviews using different models.

RNNLM	1. this is a great product , if you liked canned jerky but this is probably okay because your taste is great too . 2. we 'll left never eating a bit of more bars . will definitely buy again . 3. but my friends and i love it mixes and they are hard in colored tap up ; 4. we love this product , and our purchase was fast too . 5. the soup is quite very good , its flax flavoring for the taste ... it is pronounced and yummy .
SeqGAN	1. how good it is that i 'll really cost again , this was my favorite . 2. service was super fast and good timely manner on each order . 3. the risk is very important , but truly the best so , you 'll probably love it ! 4. each bag is n't lower and use . 5. i found that the seller was good to my own from amazon bags practically very frozen .
VGAN	1. you just ate in first , but that is the best thing . 2. but that did give me a much healthier and healthy . 3. the tea powder in need is based on the label and is very well as well . 4. chips ahoy cookies are very hard . this is just not very tasty . not what they say ? 5. this is very nice and a little sweet . the red 's very fresh 6. i found that the coffee was not bold . 7. you are not sure why loves these cookies . i will be ordering these again

of discrete tokens. In the process of training, we employ policy gradient to effectively train the generative model. Our results show that VGAN outperforms two strong baseline models for text generation and behaves well on three benchmark datasets. In the future, we plan to use deep deterministic policy gradient [11] to train the generator better. In the addition, we will choose other models as the discriminator such as recurrent convolutional neural network [16] and recurrent neural network [18].

References

1. Mikolov, Tomas and Karafiat, Martin and Burget, Lukas and Cernock, Jan and Khudanpur, Sanjeev: Recurrent neural network based language model. Interspeech, 1045–1048, 2010.
2. Kingma, Diederik P and Welling, Max: Auto-encoding variational bayes. arXiv: 1312.6144, 2014.
3. Chung, Junyoung and Kastner, Kyle and Dinh, Laurent and Goel, Kratarth and Courville, Aaron C and Bengio, Yoshua: A recurrent latent variable model for sequential data. NIPS, 2980–2988, 2015.
4. Serban, Iulian Vlad and Sordoni, Alessandro and Lowe, Ryan and Charlin, Laurent and Pineau, Joelle and Courville, Aaron C and Bengio, Yoshua: A hierarchical latent variable encoder-decoder model for generating dialogues. arXiv: 1605.06069, 2016.

5. Kim, Yoon: Convolutional neural networks for sentence classification. EMNLP, 1746–1751, 2014.
6. Kalchbrenner, Nal and Grefenstette, Edward and Blunsom, Phil: A convolutional neural network for modelling sentences. ACL, 655–665, 2014.
7. Sutton, Richard S and Mcallester, David A and Singh, Satinder P and Mansour, Yishay: Policy gradient methods for reinforcement learning with function approximation. NIPS, 1057–1063, 2000.
8. Ranzato, Marcaurelio and Chopra, Sumit and Auli, Michael and Zaremba, Wojciech: Sequence level training with recurrent neural networks. arXiv: 1511.06732, 2016.
9. Yu, Lantao and Zhang, Weinan and Wang, Jun and Yu, Yong: SeqGAN: sequence generative adversarial nets with policy gradient. arXiv:1609.05473, 2016.
10. Kingma, Diederik P and Ba, Jimmy Lei: Adam: a method for stochastic optimization. arXiv:1412.6980, 2015.
11. Lillicrap, Timothy P and Hunt, Jonathan J and Pritzel, Alexander and Heess, Nicolas and Erez, Tom and Tassa, Yuval and Silver, David and Wierstra, Daniel Pieter: Continuous control with deep reinforcement learning. arXiv:1509.02971, 2016.
12. Bengio, Samy and Vinyals, Oriol and Jaitly, Navdeep and Shazeer, Noam: Scheduled sampling for sequence prediction with recurrent Neural networks. NIPS, 1171–1179, 2015.
13. Denton, Emily L and Chintala, Soumith and Szlam, Arthur and Fergus, Robert D: Deep generative image models using a Laplacian pyramid of adversarial networks. NIPS, 1486–1494, 2015.
14. Salimans, Tim and Goodfellow, Ian J and Zaremba, Wojciech and Cheung, Vicki and Radford, Alec and Chen, Xi: Improved techniques for training GANs. NIPS, 2234–2242, 2016.
15. Marcus, Mitchell P and Marcinkiewicz, Mary Ann and Santorini, Beatrice: Building a large annotated corpus of english: the penn treebank. Computational Linguistics, 313–330, 1993.
16. Lai, Siwei and Xu, Liheng and Liu, Kang and Zhao, Jun: Recurrent convolutional neural networks for text classification. AAAI, 2267–2273, 2015.
17. Papineni, Kishore and Roukos, Salim and Ward, Todd and Zhu, Weijing: Bleu: a method for automatic evaluation of machine translation. ACL, 311–318, 2002.
18. Liu, Pengfei and Qiu, Xipeng and Huang, Xuanjing: Recurrent neural network for text classification with multi-task learning. IJCAI, 2873–2879, 2016.
19. Huszar, Ferenc: How (not) to train your generative model: scheduled sampling, likelihood, adversary? arXiv:1511.05101, 2015.
20. Hochreiter, S and Schmidhuber, J: Long short-term memory. Neural Computation 9(8), 1735–1780, 1997.
21. Che, Tong and Li, Yanran and Zhang, Ruixiang and Hjelm, R Devon and Li, Wenjie and Song, Yangqiu and Bengio, Yoshua: Maximum-likelihood augmented discrete generative adversarial networks. arXiv:1702.07983, 2017.
22. Li, Jiwei and Monroe, Will and Shi, Tianlin and Jean, Sbastien and Ritter, Alan and Jurafsky, Dan: Adversarial learning for neural dialogue generation. arXiv:1701.06547, 2017.
23. Radford, Alec and Metz, Luke and Chintala, Soumith: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv: 1511.06434, 2016.
24. Gatys, Leon A and Ecker, Alexander S and Bethge, Matthias: A neural algorithm of artistic style. arXiv: 1508.06576, 2015.

25. Chaslot, Guillaume and Bakkes, Sander and Szita, Istvan and Spronck, Pieter: Monte-carlo tree search: a new framework for game AI. Artificial Intelligence and Interactive Digital Entertainment Conference, 216-217, 2008.