Generating Text via Adversarial Training

Yizhe Zhang, Zhe Gan, Lawrence Carin
Department of Electronical and Computer Engineering
Duke University, Durham, NC 27708
{yizhe.zhang,zhe.gan,lcarin}@duke.edu

Abstract

Generative Adversarial Networks (GANs) have achieved great success in generating realistic synthetic real-valued data. However, the discrete output of language model hinders the application of gradient-based GANs. In this paper we propose a generic framework employing Long short-term Memory (LSTM) and convolutional neural network (CNN) for adversarial training to generate realistic text. Instead of using standard objective of GAN, we match the feature distribution when training the generator. In addition, we use various techniques to pre-train the model and handle discrete intermediate variables. We demonstrate that our model can generate realistic sentence using adversarial training.

1 Introduction

Learning sentence representations is central to many natural language applications. The aim of a model for such task is to learn fixed-length feature vectors that encode the semantic and syntactic properties of sentences. One popular approach to learn a sentence model is by encoder-decoder framework via recurrent neural network (RNN) [1]. Recently, several approaches has been proposed. The skip-thought model of [2] describes an encoder-decoder model to reconstruct the surrounding sentences of an input sentence, where both the encoder and decoder are modeled as RNN. The sequence autoencoder of [3] is a simple variant of [2], in which the decoder is used to reconstruct the input sentence itself.

These types of models enjoyed great success in many aspects of language modeling tasks, including sentence classification and word prediction. However, autoencoder-based methods may fail when generating realistic sentences from arbitrary latent representations [4]. The reason behind this is that when mapping sentences to their hidden representations using an autoencoder, the representations of these sentences may often occupy a small region in the hidden space. Thereby, most of regions in the hidden space do not necessarily maps to a realistic sentence. Consequently, using a randomly generated hidden representation from a prior distribution would usually leads to implausible sentences. [4] attempt to use a variational auto-encoding framework to ameliorate this problem, however in principle the posterior of the hidden variables would not cover the hidden space, rendering difficulties to randomly produce sentences.

Another underlying challenge of generating realistic text relates to the nature of RNN. Suppose we attempt to generate sentences from certain latent codes, the error will accumulate exponentially with the length of the sentence. The first several words can be relatively reasonable, however the quality of sentence deteriorates quickly. In addition, the lengths of sentences generated from random latent representations could be difficult to control.

In this paper we propose a framework to generate realistic sentences with adversarial training scheme. We adopted LSTM as generator and CNN as discriminator, and empirically evaluated various model training techniques. Due to the nature of adversarial training, the generated text is discriminated with real text, thus the training is from a holistic perspective, rendering generated sentences to maintain

Workshop on Adversarial Training, NIPS 2016, Barcelona, Spain.

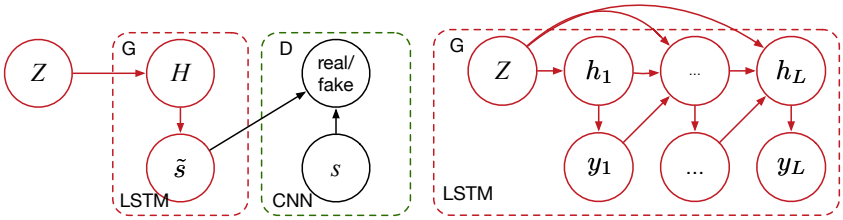

Figure 1: Left: Illustration of the textGAN model. The discriminator is a CNN, the sentence decoder is an LSTM. Right: the structure of LSTM model

high quality from the start to the end. As a related work, [5] proposed a sentence-level log-linear bag-of-words (BoW) model, where a BoW representation of an input sentence is used to predict adjacent sentences that are also represented as BoW. CNNs have recently achieved excellent results in various supervised natural language applications [6, 7, 8]. However, CNN-based *unsupervised* sentence modeling has previously not been explored.

We highlight that our model can: (*i*) learn a continous hidden representation space to generate realistic text; (*ii*) generating high quality sentence in a holistic manner; (*iii*) take advantages of several training techniques to improve convergence of GAN; (*iv*) be potentially applied to unsupervised disentangling learning and transferring literary styles.

2 Model description

2.1 TextGAN

Assume we are given a corpus $\mathcal{S} = \{s_1, \cdots, s_n\}$, where n is the total number of sentense. Let w^t denote the t-th word in sentences s. Each word w^t is embedded into a k-dimensional word vector $\boldsymbol{x}_t = \mathbf{W}_e[w^t]$, where $\mathbf{W}_e \in \mathbb{R}^{k \times V}$ is a word embedding matrix (to be learned), V is the vocabulary size, and notation $[v]$ denotes the index for the v-th column of a matrix. Next we describe the model in three parts: CNN discriminator, LSTM generator and training strategies.

CNN discriminator The CNN architecture in [7, 9] is used for sentence encoding, which consists of a convolution layer and a max-pooling operation over the entire sentence for each feature map. A sentence of length T (padded where necessary) is represented as a matrix $\mathbf{X} \in \mathbb{R}^{k \times T}$, by concatenating its word embeddings as columns, *i.e.*, the t-th column of $\mathbf{X}$ is $\boldsymbol{x}_t$.

A convolution operation involves a filter $\mathbf{W}_c \in \mathbb{R}^{k \times h}$, applied to a window of h words to produce a new feature. According to [9], we can induce one feature map $\boldsymbol{c} = f(\mathbf{X} * \mathbf{W}_c + \boldsymbol{b}) \in \mathbb{R}^{T-h+1}$, where $f(\cdot)$ is a nonlinear activation function such as the hyperbolic tangent used in our experiments, $\boldsymbol{b} \in \mathbb{R}^{T-h+1}$ is a bias vector, and $*$ denotes the convolutional operator. Convolving the same filter with the h-gram at every position in the sentence allows the features to be extracted independently of their position in the sentence. We then apply a max-over-time pooling operation [9] to the feature map and take its maximum value, *i.e.*, $\hat{c} = \max\{\boldsymbol{c}\}$, as the feature corresponding to this particular filter. This pooling scheme tries to capture the most important feature, *i.e.*, the one with the highest value, for each feature map, effectively filtering out less informative compositions of words. Further, this pooling scheme also guarantees that the extracted features are independent of the length of the input sentence.

The above process describes how one feature is extracted from one filter. In practice, the model uses multiple filters with varying window sizes. Each filter can be considered as a linguistic feature detector that learns to recognize a specific class of n-grams (or h-grams, in the above notation). Assume we have m window sizes, and for each window size, we use d filters; then we obtain a md-dimensional vector $\boldsymbol{f}$ to represent a sentence. Above this md-dimensional vector feature layer, we use a softmax layer to map the input sentence to an output $D(\mathbf{X}) \in [0, 1]$, represents the probability of $\mathbf{X}$ is from the data distribution, rather than from adversarial generator.

There exist other CNN architectures in the literature [6, 8, 10]. We adopt the CNN model in [7, 9] due to its simplicity and excellent performance on classification. Empirically, we found that it can extract high-quality sentence representations in our models.

LSTM generator We now describe the LSTM decoder that translates a latent vector $\boldsymbol{z}$ into the synthetic sentence $\tilde{s}$. The probability of a length-T sentence $\tilde{s}$ given the encoded feature vector $\boldsymbol{z}$ is defined as

$$p(\tilde{s}|\boldsymbol{z}) = p(w^1|\boldsymbol{z}) \prod_{t=2}^{T} p(w^t|w^{<t}, \boldsymbol{z}) \tag{1}$$

Specifically, we generate the first word w^1 from $\boldsymbol{z}$, with $p(w^1|\boldsymbol{z}) = \text{argmax}(\mathbf{V}\boldsymbol{h}_1)$, where $\boldsymbol{h}_1 = \tanh(\mathbf{C}\boldsymbol{z})$. Bias terms are omitted for simplicity. All other words in the sentence are then sequentially generated using the RNN, until the end-sentence symbol is generated. Each conditional $p(w^t|w^{<t}, \boldsymbol{z})$, where $< t = \{1, \ldots, t-1\}$, is specified as $\text{argmax}(\mathbf{V}\boldsymbol{h}_t)$, where $\boldsymbol{h}_t$, the hidden units, are recursively updated through $\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{z})$. The LSTM input $\boldsymbol{y}_{t-1}$ for t-th step is the embedding vector of the previous word that maximize the w^{t-1}

$$\boldsymbol{y}_{t-1} = \mathbf{W}_e[w^{t-1}]. \tag{2}$$

$\mathbf{V}$ is a weight matrix used for computing a distribution over words. The synthetic sentence is obtained by $s = [\text{argmax}(w^1), \cdots, \text{argmax}(w^L)]$.

The transition function $\mathcal{H}(\cdot)$ is implemented with an LSTM [1]. Each LSTM unit has a cell containing a state $\boldsymbol{c}_t$ at time t. Reading or writing the memory unit is controlled through sigmoid gates, namely, input gate $\boldsymbol{i}_t$, forget gate $\boldsymbol{f}_t$, and output gate $\boldsymbol{o}_t$. The hidden units $\boldsymbol{h}_t$ are updated as follows:

$$\boldsymbol{i}_t = \sigma(\mathbf{W}_i\boldsymbol{y}_{t-1} + \mathbf{U}_i\boldsymbol{h}_{t-1} + \mathbf{C}_i\boldsymbol{z}) \qquad \boldsymbol{f}_t = \sigma(\mathbf{W}_f\boldsymbol{y}_{t-1} + \mathbf{U}_f\boldsymbol{h}_{t-1} + \mathbf{C}_f\boldsymbol{z}) \tag{3}$$

$$\boldsymbol{o}_t = \sigma(\mathbf{W}_o\boldsymbol{y}_{t-1} + \mathbf{U}_o\boldsymbol{h}_{t-1} + \mathbf{C}_o\boldsymbol{z}) \qquad \tilde{\boldsymbol{c}}_t = \tanh(\mathbf{W}_c\boldsymbol{y}_{t-1} + \mathbf{U}_c\boldsymbol{h}_{t-1} + \mathbf{C}_c\boldsymbol{z}) \tag{4}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{5}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function, and $\odot$ represents the element-wise multiply operator (Hadamard product). $\mathbf{W}_{\{i,f,o,c\}}$, $\mathbf{U}_{\{i,f,o,c\}}$, $\mathbf{C}_{\{i,f,o,c\}}$, $\mathbf{V}$ and $\mathbf{C}$ are the set of parameters. Note that $\boldsymbol{z}$ is used as an explicit input at each time step of the LSTM to guide the generation of $\tilde{s}$.

3 Training Techniques

GAN objective Given the sentence corpus $\mathcal{S}$, instead of directly minimizing the objective function from standard GAN [11], we adopted an approach similar to feature matching [12]. The iterative optimization schemes consists of two steps:

$$\text{minimizing: } L_D = -\mathbb{E}_{s\sim\mathcal{S}} \log D(s) - \mathbb{E}_{z\sim p_z(z)} \log[1 - D(G(z))] \tag{6}$$

$$\text{minimizing: } L_G = \text{tr}(\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_r + \boldsymbol{\Sigma}_r^{-1}\boldsymbol{\Sigma}_s) + (\boldsymbol{\mu}_s - \boldsymbol{\mu}_r)^T(\boldsymbol{\Sigma}_s^{-1} + \boldsymbol{\Sigma}_r^{-1})(\boldsymbol{\mu}_s - \boldsymbol{\mu}_r) \tag{7}$$

where $\boldsymbol{\Sigma}_s, \boldsymbol{\Sigma}_r$ represents the covariance matrices of real and synthetic sentence feature vector $\boldsymbol{f}_s, \boldsymbol{f}_r$, respectively. $\boldsymbol{\mu}_s, \boldsymbol{\mu}_r$ denote the mean vector of $\boldsymbol{f}_s, \boldsymbol{f}_r$, respectively. $\boldsymbol{\Sigma}_s$,$\boldsymbol{\Sigma}_r$,$\boldsymbol{\mu}_s$ and $\boldsymbol{\mu}_r$ are empirically estimated on minibatch. By setting $\boldsymbol{\Sigma}_s = \boldsymbol{\Sigma}_r = \mathbf{I}$, this reduces to the feature matching technique from [12]. Note that this second loss L_G is the Jensen-Shannon divergence between two multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_r, \boldsymbol{\Sigma}_r)$ and $\mathcal{N}(\boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$. Instead of capturing the first moment similarity, we cast more stringent criteria of matching the feature covariance of real and synthetic data. Despite the fact that the feature vector is not necessarily Gaussian distributed, empirically this loss (7) works well. Intuitively, this technique provides stronger signal for modifying the generator to make the synthetic data more realistic.

Unfortunately, using small subset of data would yield inaccurate estimation of the empirical covariance matrix $\boldsymbol{\Sigma}_s$ and $\boldsymbol{\Sigma}_r$. Furthermore, if the subset size is less than the feature dimensionality, estimating (7) would result in singular matrices. To remedy this issue, we use a sliding window of most recent m mini-batches for estimating both $\boldsymbol{\Sigma}_s$ and $\boldsymbol{\Sigma}_r$.

Approximated discretization To train the generator G which contains discrete variable, direct application of gradient estimation would fails. Score function based algorithms, such as REINFORCE [13] algorithm, obtains unbiased gradient estimation for discrete variables by using Monte Carlo estimation. However the variance of the gradient estimation could be large [14]. Here we consider a soft-argmax function when performing the inference as an approximation to the (2):

$$\boldsymbol{y}_{t-1} = \mathbf{W}_e\text{softmax}(\mathbf{V}\boldsymbol{h}_{t-1} \odot L). \tag{8}$$

where $\odot$ represents element-wise product. When $L \to \infty$, this approximation would becomes (2). The w^{t-1} is constrained to be either 0 or have an absolute value greater than 1.

Confusion pre-training for feature learning Previous literature [11, 12] has discussed the fundamental difficulty in training GAN model using gradient-based method. In general, gradient descent optimization scheme would fail to converges to the Nash equilibrium by moving along orbit trajectory among saddle points. Presumably, a good initialization would encourage convergence by reducing the orbit movement. To achieve good initialization, we initialize the LSTM parameters for the generator by pre-training a standard auto-encoder LSTM model. For the discriminator, we use a confusion training strategy. For each sentence in the corpus, we randomly swap two words to construct a tweaked counterpart sentence. The discriminator is pre-trained to classify the tweaked sentences from the true sentence. The swapping operation, rather than adding/deleting is used because we attempt to train the CNN discriminator to learn the *sentence structure* feature, rather than absence/presence of certain words (*words presence* feature). This strategy helps avoid the generator to produce repeated "realistic" words that rewards the *words presence* feature to achieve a higher score of being classified as from real data.

4 Experiments

Our model is trained using a combination of two datasets: 1). BookCorpus dataset [15], which consists of 70 million sentences from over 7000 books; 2). ArXiv dataset, which consists of 5 million sentences from various subjects, scrawled from arXiv website. The purpose of including two different corpus is to investigate whether the model can generate sentences that integrates both scientific writing style and vocabulary with informal writing style and vocabulary. We randomly choose 1 million sentences from BookCorpus and 1 million sentences from arXiv dataset to construct our training dataset.

We train the generator and discriminator iteratively. Given that the LSTM generator typically involves more parameters and is more difficult to train than the CNN discriminator, we perform one optimization step for the discriminator for every $K = 5$ steps for the generator. When optimizing θ_D, we consider using the unrolling techniques similar to [16]. Instead of optimizing θ_D with fixed θ_G, we obtain the optimal θ_D by,

$$\theta_D^* = \text{argmin}_{\theta_D} L_D(\theta_D, \theta_G^M(\theta_D, \theta_G)) \tag{9}$$

where the gradient with respect to θ_D is obtained by

$$\frac{dL_D}{d\theta_D} = \frac{\partial L_D(\theta_D, \theta_G^K(\theta_D, \theta_G))}{\partial \theta_D} + \frac{\partial L_D(\theta_D, \theta_G^K(\theta_D, \theta_G))}{\partial \theta_G^K(\theta_D, \theta_G)} \frac{d\theta_G^K(\theta_D, \theta_G)}{d\theta_D} \tag{10}$$

Both of the generator and discriminator are per-trained with the strategies described in Section 2.

For the CNN encoder, we employ filter windows (h) of sizes {3,4,5} with 300 feature maps each, hence each sentence is represented as a 900-dimensional vector. For both, the LSTM sentence decoder and paragraph generator, we use one hidden layer of 500 units.

Gradients are clipped if the norm of the parameter vector exceeds 5 [17]. The Adam algorithm [18] with learning rate 1×10^{-4} for both discriminator and generator is utilized for optimization. We use mini-batches of size 128. All experiments are implemented in Theano [19], using a NVIDIA GeForce GTX TITAN X GPU with 12GB memory. The model was trained for roughly one week.

We first examine how well the generator can produce similar feature distributions to mimic the real data. We calculate the empirical expectation and covariance of 900 CNN top layer features over a minibatch of 128 real sentences and 128 synthetic sentences. As shown in Figure 2, the expectation of these 900 features from synthetic sentences matches well with the feature expectation from the real sentences. The covariances of 900 features against the first feature (*i.e.*, $Cov(f_i, f_1)$) also demonstrate consistency between the real sentences and synthetic sentences.

We further empirically evaluate whether the latent variable space can densely encode sentences with appropriate gramma structure, and to visualize the transition from sentence to sentence. We construct a linear path between two randomly selected points in the latent space, and generate the intermediate sentences along the linear trajectory. The results are presented in Table 1. Generally the produced sentence is grammatically correct and semantically reasonable. The transition demonstrate smoothness and interpretability, however the wording choices and sentence structure showed dramatical changes in some regions in the latent space. Presumably the local “transition smoothness” can varies from region to region.

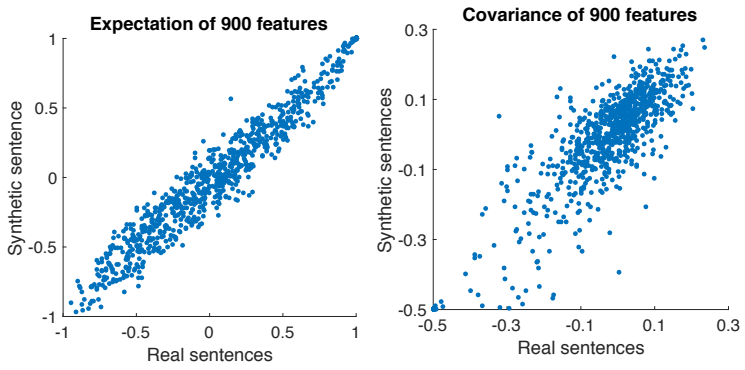

Figure 2: Left: the scatter plot of the expectations of features from real data against the expectations of features from synthetic data. Right: the scatter plot of the covariances of features (*v.s.* 1st feature, $Cov(f_i, f_1)$) from real data against the covariances of features from synthetic data.

Table 1: Intermediate sentences produced from linear transition between two random points in the latent space. Each sentence is generated from a latent point on a linear grid with equidistance

A	we show the efficacy of our new solvers , making it up to identify the optimal random vector .
-	we made the price of such new solvers and reduce its time series pairs .
-	we realized and minimize the best of such tasks in terms of multidimensional high dimensional kernels .
-	we realized and minimize the best of such tasks are invariant on such a sparse linear program .
-	we realized and minimize the price of <UNK> 's plans , by drawing up a constant .
-	we realized the price used to pay , whatever they are .
-	i realized mrs. <UNK> was surprised that most of them got safe .
B	dylan realized the distance of the walls to them .

Note that the generated text demonstrates some entangling of both scientific style of writing and informal style of writing, because the training is on the combination of two corpus. The generator can conjure novel sentence by leveraging the grammatical rule and property of words.

We observed that the discriminator can still sufficiently distinguish the synthetic sentences from the real ones (the probability to predict synthetic data as real is around 0.08), even if the synthetic sentences seems to perserve reasonable grammatical structure and use proper wording. Probably the CNN is able to sophisticatedly characterize the semantic meaning and differentiate sentences in a holistic perspective, while the generator may stuck into a sweet point where any slight modification would render a higher loss (7) for the generator.

5 Conclusion

We presented a text generation model via adversarial training and discussed several techniques for training such a model. We demonstrate that the proposed model can produce realistic sentences by mimicking the input real sentences, and the learned latent representation space can continuously encode plausible sentences.

In future work, we attempt to disentangle the latent representations for different writing style in an unsupervised manner. This would enable a smooth lexical and grammatical transition between different writing styles. The model also will be fine-tuned in order to achieve more compelling results, such as improved sentences with better semantical interpretation. Furthermore, a more comprehensive quantitative comparison will be performed.

References

[1] S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural computation*, 1997.

[2] R. Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *NIPS*, 2015.

[3] A. Dai and Q. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[4] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[5] F. Hill, K. Cho, and A. Korhonen. Learning distributed representations of sentences from unlabelled data. In *NAACL*, 2016.

[6] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. In *ACL*, 2014.

[7] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[8] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, 2014.

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. In *JMLR*, 2011.

[10] R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. In *NAACL HLT*, 2015.

[11] 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*, pages 2672–2680, 2014.

[12] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016.

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

[14] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[15] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, 2015.

[16] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[17] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[18] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[19] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv:1211.5590*, 2012.