Language Modeling with Generative Adversarial Networks

Mehrad Moradshahi
Department of Electrical Engineering
Stanford University
mehrad@stanford.edu

Utkarsh Contractor
Department of Computer Science
Stanford University
utkarshc@stanford.edu

Abstract

Generative Adversarial Networks (GANs) have been promising in the field of image generation, however, they have been hard to train for language generation. GANs were originally designed to output differentiable values, so discrete language generation is challenging for them which causes high levels of instability in training GANs. Consequently, past work has resorted to pre-training with maximum-likelihood or training GANs without pretraining with a WGAN[6] objective with a gradient penalty. In this study, we present a comparison of those approaches. Furthermore, we present the results of some experiments that indicate better training and convergence of Wasserstein GANs (WGANs) when a weaker regularization term is enforcing the Lipschitz constraint.

1 Introduction

Generative Adversarial Networks (GANs) [3] are used to train generative models in an adversarial setup, with a generator generating images that are trying to fool a discriminator whose role is to discriminate between real and synthetic images. GANs have shown good results and gained much attention when it comes to producing new images, however, language generation is an active and challenging area of research with GANs, mostly due to the non-differentiable nature of generating discrete symbols. Although GANs can accurately model complex distributions, they are known to be difficult to train due to instabilities caused by a difficult minimax optimization problem in a game theoretic situation, where the generator and discriminator aim to find a Nash Equilibrium.

Formally, the game between the generator G and the discriminator D is the minimax objective:

$$min_G max_D \, \mathbb{E}_{x\sim \, P_r}[log(D(x))] + \mathbb{E}_{\hat{x}\sim \, P_g}[log(1-D(\hat{x}))] \tag{1}$$

where Pr is the data distribution and Pg is the model distribution implicitly defined by $\hat{x} = G(z), z = p(z)$ (the input z to the generator is sampled from some simple noise distribution p, such as the uniform distribution or a spherical Gaussian distribution). Wasserstein GANs (WGANs) with gradient penalty provide stable training of GANs and provide a better approach to reasonably train GANs. [4]

In this paper, we will be evaluating the use of Generative Adversarial Networks (GANs) for text generation and language modeling. In particular we look at two implementations [7] and [8] using WGAN with gradient penalty. Furthermore, we provide some experimental results by training GANs with weaker regularization[2] and a modified gradient penalty that provides indications of better training stability of WGANs.

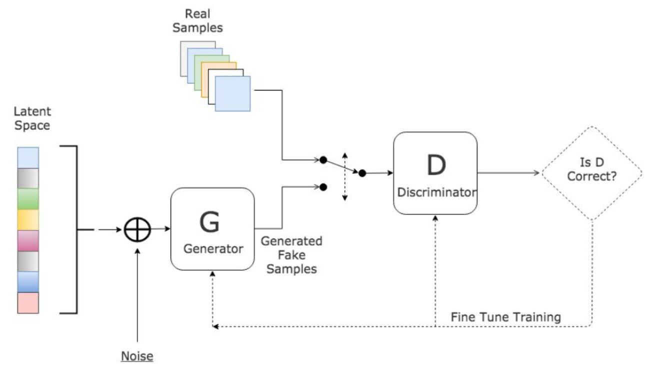


Figure 1: GAN general architecture

2 Related work

Sai et. al [8] have introduced a simple baseline that addresses the discrete output space problem without relying on gradient estimators and shows that it is able to achieve state-of-the-art results on a Chinese poem generation dataset and presented quantitative results on generating sentences from context-free and probabilistic context-free grammars, and qualitative language modeling results. A conditional version is also described that can generate sequences conditioned on sentence characteristics.

Ofir et. al [7] have shown that recurrent neural networks can be trained to generate text with GANs from scratch using curriculum learning, by slowly teaching the model to generate sequences of increasing and variable length. They empirically show that their approach vastly improves the quality of generated sequences compared to a convolutional baseline.

Henning et. al [2] present theoretical arguments why using a weaker regularization term enforcing the Lipschitz constraint is preferable. These arguments are supported by experimental results on several data sets. For stable training of Wasserstein GANs, they propose to use the following penalty term to enforce the Lipschitz constraint that appears in the objective function:

$$\mathbb{E}_{\hat{x}\sim\ \tau}[(max\{0, \left\|\nabla f(\hat{x})\right\|_2 - 1\})^2] \tag{2}$$

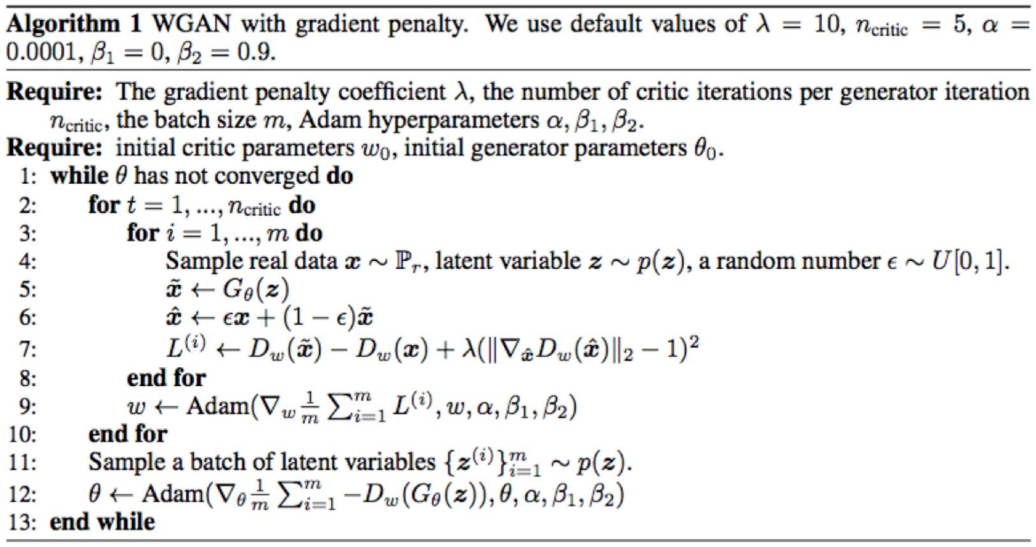
3 Data

We have used the One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling[1]. The project makes available a standard corpus of 0.8 billion words to train and evaluate language models. Duplicate sentences were removed, dropping the number of words from about 2.9 billion to about 0.8 billion. Words outside of the vocabulary were mapped to [unk] token, also part of the vocabulary. Sentence order was randomized, and the data was split into 100 disjoint partitions One such partition (1%) of the data was chosen as the held-out set. The benchmark is available as a code.google.com project at `https://code.google.com/p/1-billion-word-language-modeling-benchmark/`; besides the scripts needed to rebuild the training/held-out data, it also makes available log-probability values for each word in each of ten held-out data sets, for each of the baseline n-gram models.

4 Methods

We have used the implementation in [7] and [8] as our baseline. In both cases, we have used the 1 billion word dataset. GAN based methods have often been critiqued for lacking a concrete evaluation strategy [9] and resort to some sort or subjective evaluation. For our evaluations, in [7] we generate 640 sequences from each model and measure %-IN-TEST-n, that is, the proportion of word n-grams from generated sequences that also appear in a held-out test set. We evaluate these metrics for n 1, 2, 3, 4., however for [8] we just rely on human understanding and thus is qualitative.

In both cases, we have used multiple hyper-parameter variations to find the most optimal set. For [7], we were able to improve the results by increasing the batch-size and increasing the number of epochs. Recently, Wasserstein GAN (WGAN) have been introduced which solve the vanishing gradient problem by clipping the weight values to lie in a cube. However, it has been shown that the gradient signals that generator receives might become unstable. An improved version uses weaker regularization for gradient penalty instead of clipping to force that double-sided gradient approaches. We have implemented this method and used it with a model trained based on [7]. Training duration for GANs is unreasonably long, considering its reaches convergence at all. For [7] we have trained the network on four K80 GPUs for up to 14 days to run the variable length curriculum learning with teacher helping to generated sequences of length up to 25 with multiple hyper parameters selections.



Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

```
Require: The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration
    $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
Require: initial critic parameters $w_0$, initial generator parameters $\theta_0$.
 1: while $\theta$ has not converged do
 2:     for $t = 1, ..., n_{\text{critic}}$ do
 3:         for $i = 1, ..., m$ do
 4:             Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
 5:             $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
 6:             $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
 7:             $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
 8:         end for
 9:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:     end for
11:     Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
12:     $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
13: end while
```

Figure 2: Improved WGAN algorithm

5 Experiments/ Results/ Discussion

We used Amazon Elastic Compute Cloud and Microsoft Azure virtual machines to run our experiments. They provided the necessary CPU and GPU power to scale up our algorithms and run parallel processes. We evaluated our first model by generating sequences from the model and measuring "percent in Test-N", i.e. the proportion of word n-grams from generated sequences that also appear in a held-out test set. The goal is to measure the extent to which the generator is able to generate real words with local coherence. In the second model, we use human perception to decide on the quality of the produced text.

5.1 Language Generation with Recurrent Generative Adversarial Networks without Pre-training

This section follows the implementation on the lines of [7] where we train GRU based RNNs [5] to generate text using WGANs from scratch using curriculum learning, by teaching the model to generate sequences of increasing and variable length on the Billion word dataset [1]. We trained our model for up to 25 sequences on variable batch sizes, epochs, network layers, sequence lengths and observed the following results (Table 1)

Generated Samples	Unigrams	Bigrams	Trigrams	Quadgrams
”I have more than the man was” ”The NBC said so , Most of the” ”Reid Maybe you got the more than”	0.84	0.57	0.27	0.03

Table 1: Experiment 1 results

The proposed idea was to augment the WGAN loss by a regularization term that penalizes the deviation of the gradient norm of the critic with respect to its input from one (leading to a variant referred to as WGAN-GP, where GP stands for gradient penalty).[7]. However we observed that increasing the batch size to 128 and using a weaker regularization term enforcing the Lipschitz constraint[2], led to highest accuracy across our experiments, and as early as sequence length 11 resulting to a Unigrams and Bigrams %-in-N accuracy of 85% and 57% respectively.

We also noticed an improvement in the stability and convergence of the GAN training when we used a modified gradient penalty [2] with the WGAN objective, as seen in (Figure 4) below. More specifically when we use equation $\mathbb{E}_{\hat{x}\sim\ \tau}[(max\{0, \left\|\nabla f(\hat{x})\right\|_2 - 1\})^2]$ to enforce the Lipschitz constraint that appears in the objective function instead of the previously considered approaches of clipping weights and of applying the stronger gradient penalty $\mathbb{E}_{\hat{x}\sim\ \tau}[(\left\|\nabla f(\hat{x}\ )\right\|_2 - 1)^2]$.

In addition to more stable learning behavior, the proposed regularization term leads to lower sensitivity to the value of the penalty weight λ (demonstrating smooth convergence and well-behaved critic scores throughout the whole training process for different values of λ)

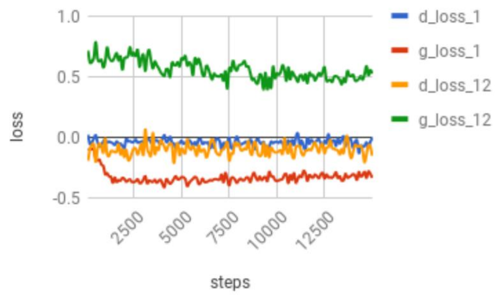


Figure 3: WGAN with original GP

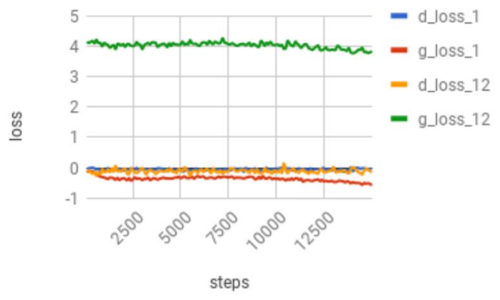


Figure 4: WGAN with modified GP

5.2 Adversarial Generation of Natural Language

This section follows the implementation on the lines of [8]. We have access to a variety of generator and discriminator architectures including LSTM based RNN and 1D CNN. So far we got the best results by using LSTM for both generator and discriminator. We trained our model for different objective functions to generate 64 batches of 5-word sequences after every 50 iterations. As you see in Table 2, WGAN outperform GAN, and WGAN with GP outperforms WGAN.
We used both quantitative and qualitative measures to compare the results. As you see GAN makes frequent grammatical mistakes and repeats same words in a phrase. We see these errors a lot in the earlier stages of training, but even after many iterations, GAN keeps making those mistakes. WGAN has two advantages. It trains faster than GAN and produces more coherent results. Even though it still produce illegible phrases occasionally, it learns much faster not to repeat the same mistakes again. Adding gradient penalty makes WGAN more robust to errors and training instability. If not being set properly, the training process can be highly unstable and the loss starts bouncing around. We Used

cross-validation on dev set to tune the hyper-parameters. Considering it takes a day to run each experiment, we used a small set of candidate values for hyper-parameters and used cross-validation on dev set to tune them.
The score indicates the fraction of all sentences produced during the training, which are not in the corpus. It is a rough estimate of how well the generator is performing.
After running the code for 5 epoch we got the following results:

Objective function	Samples	Score
GAN	”he only [unk] hard ?” ”about about in month ?” ”the are are murder !”	0.85
WGAN	”like last doing monday arrested” ”to all businesses won ...” ”they have were arrested ?”	0.88
WGAN with GP	”your even example killed it” ”he could have started .” ”finally cannot do parties ?”	0.90

Table 2: Experiment 2 results

6 Conclusion/ Future Work

In conclusion, we showed that GANs can produce original text with local coherency. Use of WGAN[6] is preferred over GAN as it trains faster, with higher stability, and produce more coherent phrases. Furthermore, we also show that using a weaker regularization term enforcing the Lipschitz constraint in the WGAN gradient penalty, has promised for more stable GAN training and smoother convergence.
In future work, we would like to explore more algorithms and techniques such as MasKGAN [10], where we would like to extend the results of our experiments and evaluate its performance on MASKGAN which introduces an actor-critic conditional GAN that fills in missing text conditioned on the surrounding context. We would also like to train GANs on more coherent text corpus such as the Wikipedia dataset to generate language that is more globally coherent. As previously mentioned, GANs are challenging when it comes to training stability and convergence and the hyper-parameters are sensitive to small changes. Given more time and computation power, we can run these algorithms for longer time and more variations.

Acknowledgments

We would like to thank Sandeep Subramanian and Sai Rajeswar who helped us in implementing the model and kindly provided help and guidance during the course of this project.

References

[1] Mike Schuster Qi Ge Thorsten Brants Phillipp Koehn-Tony Robinson Ciprian Chelba, Tomas Mikolov. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005v3*, 2013.

[2] Denis Lukovnicov Henning Petzka, Asja Fischer. On the regularization of wasserstein gans. *arXiv preprint arXiv:1709.08894v2*, 2017.

[3] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial networks. *arXiv preprint arXiv:1406.2661v1*, 2014.

[4] Martin Arjovsky Vincent Dumoulin Aaron Courville Ishaan Gulrajani, Faruk Ahmed. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028v3*, 2017.

[5] Dzmitry Bahdanau Kyunghyun Cho, Bart Van Merrienboer and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Lon Bottou Martin Arjovsky, Soumith Chintala. Wasserstein gan. *arXiv preprint arXiv:1701.07875v3*, 2017.

[7] Ben Bogin Jonathan Berant Lior Wolf Ofir Press, Amir Bar. Language generation with recurrent generative adversarial networks without pre-training. *arXiv preprint arXiv:1706.01399v3*, 2017.

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

[9] Wojciech Zaremba Vicki Cheung Alec Radford Xi Chen Tim Salimans, Ian Goodfellow. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498v1*, 2016.

[10] Andrew M. Dai William Fedus, Ian Goodfellow. Maskgan: Better text generation via filling in the ______. 2018.