End-to-End Differentiable GANs for Text Generation

Sachin Kumar **Yulia Tsvetkov**
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{sachink,ytsvetko}@cs.cmu.edu

Abstract

Despite being widely used, text generation models trained with maximum likelihood estimation (MLE) suffer from known limitations. Due to a mismatch between training and inference, they suffer from exposure bias. Generative adversarial networks (GANs), on the other hand, by leveraging a discriminator, can mitigate these limitations. However, discrete nature of text makes the model non-differentiable hindering training. To deal with this issue, the approaches proposed so far, using reinforcement learning or softmax approximatons are unstable and have been shown to underperform MLE. In this work, we consider an alternative setup where we represent each word by a pretrained vector. We modify the generator to output a sequence of such word vectors and feed them directly to the discriminator making the training process differentiable. Through experiments on unconditional text generation with Wasserstein GANs, we find that while this approach, without any pretraining is more stable while training and outperforms other GAN based approaches, it still falls behind MLE. We posit that this gap is due to autoregressive nature and architectural requirements for text generation as well as a fundamental difference between the definition of Wasserstein distance in image and text domains.

1 Introduction

Under the most popular paradigm, text generation models are trained with maximum likelihood estimation (MLE) via teacher forcing [1]. Training neural networks under MLE does not model sequence probabilities, since, at inference, the model is conditioned on sequences that may have never been observed at training time. Indeed, generated texts using this approach are often degenerate [2].

A natural alternative, to mitigate MLE limitations, is to frame the problem under the Generative Adversarial Network (GAN) paradigm [3]. GANS have been used successfully for generating images [4], audio [5] and videos [6], learning interpretable representation of images [7], controlling attributes of the generated data [8]. For text, modeled as a sequence of discrete symbols, a naive computation of the gradients backpropagating from the discriminator to the generator is however intractable. Hence, Language GANs are based on gradient estimation via RL-based techniques [4, 9–11] or by differentiable approximation of Softmax using Gumbel-Softmax or similar variants [12, 13].

Most of the said approaches suffer from issues like training instability (due to high variance of gradients with RL) or mode collapse. Moreover, almost *all* of them require a considerable amount of pre-training using MLE followed by GAN based fine tuning (often with low learning rates) [14]. As a result, the best performing solution tend to stay close to the one obtained by MLE [15]. This gives a false impression of the efficacy of GAN training, but recent work has shown that a properly tuned and temperature-controlled language model trained with MLE using teaching forcing outperforms most of these models [14, 15]. In this work, we thus posit that to fairly evaluate the effectiveness and stability of training language GANs, they should be trained and evaluated without pretraining.

1st I Can't Believe It's Not Better Workshop (ICBINB@NeurIPS 2020), Vancouver, Canada.

Inspired by recently proposed continuous-output language generation [16, 17] which has shown promise in machine translation, we propose the following: Feed a noise vector z as input to the generator and instead of predicting a softmax distribution at every step to obtain a token from, generate a low dimensional vector (representing the embedding of the token). Input this sequence of generated vectors, rather than corresponding tokens, to the discriminator. This facilitates computing exact gradients which can flow from the discriminator to the generator. We explore ways for training this model with Wasserstein GAN objective [18, 19] which is known to stabilise training and reduce mode collapse. After the training is complete, to sample text, we generate a sequence of vectors from the generator and find their nearest neighbor in the embedding table.

We train and evaluate our proposed model on the task unconditional text generation on a simple dataset COCO Image Captions [20]. We show that this approach results in stable training and performs on par with or better than the prior work *without pretraining* in terms of quality and diversity. But we find that this approach, albiet closer to a traditional GAN framework, fails to perform better than MLE. We posit that this gap in performance results from the following: (1) Fundamental differences between the nature of tasks, text generation which is inherently autoregressive in nature with long term dependencies in many cases, versus tasks like image generation where the entire image is usually generated at once. Domains where GANs have shown to be successful rely on simpler feed-forward architectures compared to recurrent architectures required for text generation (2) Most optimization objectives for training GANs (like WGAN) are defined based on distances between samples (like Wasserstein distance). While this distance is easy to define for fixed dimensional data like images, for sequential data like text this distance is not well defined.

2 Generative Models for Text

Most language generation systems are defined over a fixed vocabulary $\mathcal{V}$ where each token $w \in \mathcal{V}$ is represented as a discrete unit by a one hot vector ($o(w)$) of length $|\mathcal{V}|$ (we consider only word-level models in this work). Text generation lends itself to autoregressive modeling, feeding the output of step t to the input of step $t+1$. A natural choice of architecture of this model is recurrent networks (RNNs) like LSTMs [21], GRUs [22] or more recently relational memory networks [23] and transformers [24]. If the start word is w_0, we define,

$$\begin{aligned} \mathbf{e}_{w_t} &= o(w_t)\mathbf{W}_i \\ \mathbf{h_t} &= \mathrm{RNN}(\mathbf{e}_{w_t}, \mathbf{h_{t-1}}) \\ w_{t+1} &\sim f(\mathrm{softmax}(\mathbf{W}_o \mathbf{h}_t/\tau)) \end{aligned}$$

where $t > 0$ and $\mathbf{h}_0$ is the initial hidden state. In other words, at time step t, we first encode the input token x_t into a embedding $\mathbf{e}(w_t)$ using an embedding matrix $\mathbf{W}_i$. We input the embedding to the RNN network along with the hidden vector of the previous step, $\mathbf{h_{t-1}}$ to obtain a hidden vector $\mathbf{h}_t$ We then transform the output $\mathbf{h}_t$ into a probability distribution over the vocabulary through a linear transform using a weight matrix $\mathbf{W}_o$ (and an optional temperature parameter τ) followed by softmax. The next word w_{t+1} can be obtained from (a function f) of this probability distribution by a sampling procedure [] which along with the temperature τ controls the trade-off between the diversity and quality of the generated samples [25].

The simplest way to train this generative model is by minimizing the negative log-likelihood of a training corpus with respect to the output multinomial distribution, which is essentially training it like a language model. Training is usually done by passing as input at timestep t, the correct token from the training example w_t^* rather than the output of the previous step w_{t-1} (teacher forcing). Teaching forcing creates a disconnect between training and inference and leads to *exposure bias* where the model only sees correct input while training. This has been shown to lead to bad sample quality [26, 12] leading to research into GANs for text generation. Additionally, naive sampling from the softmax output often leads to repetitions or hallucinations leading to degenerate samples [25]. Sampling strategies like top-k sampling or nucleus sampling [25] have been used to address this issue.

2.1 Generative Adversarial Networks

GANs [3] learn the distribution $P_r^*(x)$ of a given text corpus through a two player adversarial game between a generator G_θ and a discriminator D_ϕ. The discriminator is trained to distinguish between

real sentences and fake ones (generated by G_θ). The original formulation proposes using a minimax objective, where the generator tries to generate better examples to fool the discriminator. While loss functions to train GANs are under active research [27], training GANs in most settings require that gradients can flow from the discriminator to the generator. This works well when the output of the generator is real valued, like pixels in images or videos.

However, due to the sampling process involved in text generation as described above, the output of the generator is discrete which makes it non-trivial to train G_θ. Most of the research in text GANs tackles this issue by one of the following two methods: (1) RL based [26, 28–30, 10, 11]. The idea behind all of these works is similar: use the REINFORCE algorithm [31] to get an unbiased gradient estimator for the generator, and apply a roll-out policy to obtain the reward for the discriminator. (2) Approximate the sampling process with tricks like Gumbel-softmax [32] and soft-argmax [12] to make the model differentiable. The latter uses MMD to match the features of real and generated text.

Despite the differences in training or architecture, most text GAN models except [10] follow a common paradigm: Initialize the generator with MLE followed by fine-tuning with a GAN objective. [14] shows that as a result such models remain close to and do not perform better than a carefully sampled pretrained model. Additionally, they rely on sampling from the softmax output as a source of diversity. Irrespective of the sampling strategy, with the auto-regressive factorization of text generation, all uncertainty is represented in the generator output distribution, making it difficult to search for multiple modes in the data distribution [**?**]. On the contrary, traditional GANs rely on an initial input (using a random vector z) to get diverse samples.

3 CONtinous Output Text GAN (ConGAN)

3.1 Generator with Continuous Outputs

Following [16], we propose the following model architecture: instead of treating each word discretely and learning input and output embedding matrices, W_i and W_o respectively (§2), we represent each word in the vocabulary as a pre-trained and fixed word embedding $e_w^{(p)} \in \mathbb{R}^k$ where $k \ll |\mathcal{V}|$. These representations can be learned by training a word embedding model on the training corpus (or any other corpus) [33–36]. We modify the generation process as follows:

$$
\begin{aligned}
\mathbf{h}_t &= \mathrm{RNN}(\hat{\mathbf{e}}_t, \mathbf{h}_{t-1}) \\
\mathbf{v}_t &= \mathrm{FF}(\mathbf{h}_t) \\
\hat{\mathbf{e}}_t &= \frac{\mathbf{v}_t}{\|\mathbf{v}_t\|}
\end{aligned}
$$

Where $t > 1$, $\hat{\mathbf{e}}_0 = \mathbf{e}_{\text{start}}$ is the embedding of the start of sequence token (`bos`) and $\mathbf{h}_0 = \mathrm{FF}(z), z \sim \mathcal{N}(0, 1)$, where FF is a fully connected network which transforms the hidden vector to an output vector. Simply put, at each time step t, the model gets a L_2 normalized vector as input and outputs another L_2 normalized vector which is then fed to the next step. The output of this generator can be represented as a sequence of normalized vectors $(\hat{\mathbf{e}}_1, \ldots, \hat{\mathbf{e}}_L)$ or more concisely by a matrix $\tilde{\mathbf{X}} \in \mathbb{R}^{k \times L}$. We feed this sequence (of continuous vectors) into a discriminator whose task is to distinguish between a sequence of pretrained (and $L2$ normalized) embeddings (coming from real text) and generated vectors. Since there is no sampling or argmax involved, the model maintains differentiability. The goal of the discriminator is two fold: (1) Differentiate the individual word vectors from the generated vectors and (2) differentiate the sequence of real word vectors with a sequence of generated vectors[1].

To sample text from a trained generator, we first sample the matrix $\mathbf{X}$ as described and find the nearest neighbor of each of the vectors (based on cosine similarity) in the pretrained embedding table to generate the sequence of words. Since we use a deterministic nearest neighbor search for predicting the word from the embedding, all the randomness in this model is clamped into the noise vector z. This change better aligns with generative models in other domains as well as aims to solve degenerate results obtained from different sampling methods.

[1] We also experiment with another setup where we find the nearest neighbor of $\hat{\mathbf{e}}_t$ in the embedding table and feed that to the model. This still allows gradient flow through a straight through estimator. This performs similar to the proposed setup.

3.2 Discriminator

We use a convolutional neural network [37] as our discriminator D parameterized by ϕ which contains convolutional and max-pooling layers. A convolution filter $\mathbf{W}_{\text{conv}} \in \mathbb{R}^{k \times l}$ is applied to a window of l words to produce new features. After applying a non-linear activation, we use max-pooling over time [38] to the feature maps. Convolution operator finds features independent of their position while max-pooling operator intends to capture the most salient ones. Our model uses multiple filters for different window sizes as described in §4.3. Consider K filters of sizes $l_1, \ldots, l_K$ each, then the pre-final layer has $\sum_{i=1}^{K} l_i$ dimensions. We add a fully connected layer on top on this feature vector to get a scalar value. Let the embeddings obtained from a sentence r sampled from the real data set be denoted as $\mathbf{X}_r$, while that generated by the model be $\tilde{\mathbf{X}}_g$. We train the discriminator with the following objective,

$$\mathcal{L}_D = \mathbb{E}_{\tilde{\mathbf{X}}_r \sim P_g, \mathbf{X}_r \sim P_r}[g(D(\mathbf{X}_r), D(\tilde{\mathbf{X}}_g))]$$

where g is a distance function dependent on the specific GAN objective used to train this model as we see next.

3.3 Training Loss

Leaning on prior work and initial experiments with different GAN objectives g [3, 18, 39, 19], we train our proposed model with the Wasserstein GAN objective [18] with gradient penalty (WGAN-GP) [19]. Minimizing Wasserstein distance between data distribution (P_r) and the generator distribution (P_g) has been shown to be helpful both for avoiding mode collapse as well as improving training stability [18]. This distance is given as:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{x,y \in \gamma}[\|x - y\|] \tag{1}$$

Where $\Pi(p_r, p_g)$ is the set of all possible joint distributions whose marginals are P_r and P_g respectively and $\| \cdot \|$ is ℓ^2 distance (we discuss the impact of this choice §6)). Since it is intractable to compute this quantity, [19] propose using Kantorovich-Rubinstein duality [40] to transform this problem into one which minimizes the following function by adding a Lipschitz continuity constraint on the discriminator. They approximate this constraint by constraining the norm of its gradient to be less than 1. The loss function to train the discriminator is given as,

$$\mathcal{L}_D^{(\text{wgan-gp})} = \mathbb{E}_{\tilde{\mathbf{X}}_g \sim P_g}[D(\tilde{\mathbf{X}}_g)] - \mathbb{E}_{\mathbf{X}_r \sim P_r}[D(\mathbf{X}_r)] + \mathbb{E}_{\hat{\mathbf{X}}_r \sim P_{\hat{X}}}[(\|\nabla_{\hat{\mathbf{X}}} D(\hat{\mathbf{X}})\|_2 - 1)^2], \tag{2}$$

where $\hat{\mathbf{X}} = \alpha \mathbf{X}_r + (1 - \alpha)\tilde{\mathbf{X}}_g$, $\alpha \sim \mathbb{U}[0, 1]$. This loss function is minimized with respect the discriminator parameters ϕ. The loss function to train the generator is given as

$$\mathcal{L}_G^{(\text{wgan-gp})} = -\mathbb{E}_{\tilde{\mathbf{X}}_g \sim P_g}[D(\tilde{\mathbf{X}}_g)].$$

This loss function has been shown to perform well on different kinds of data like images [] and even sequential data like video and audio]. But as we discuss in §6, this success has only been shown with feed-forward architectures or cases where maintaining local consistencies produces acceptable outputs. **[note to myself: add citations –SK]**

3.4 End-to-End Training of ConGAN

During the experiments, we observed that, since in the initial training phase the embedding predicted by the generator at step t is erroneous, passing it to step $t + 1$ compounds this error as the sequence progresses, resulting in the training not being able to converge. To enable better convergence, we propose a curriculum for training this model similar to the one described in [41]

We begin by training the model to generate short sequences and increase the length gradually as the training progresses. That is, we first train the model to generate sequences of length 1 by training with real sentences truncated to length 1. After a certain number of train steps, we increase the generated length to 2 and so on. Since we move to length $T + 1$, only after training the model to sentences of

length T effectively, error compounding is drastically reduced. The training schedule of ConGAN is more formally described in algorithm 1.

Algorithm 1: Training Procedure for ConGAN with WGAN with gradient penalty

Result: G_θ
Require: Learning rate α, Optimizer parameters β_1, β_2, gradient penalty coefficient λ, batch size m, number of discriminator steps n_c, pretrained embedding table $\mathbf{W}_e$ of dimension d, a sequence of steps describing the curriculum S.
Require: Initial values of generator and discriminator parameters, θ_0 and ϕ_0 respectively.
$\ell \leftarrow 1$;
for $c \leftarrow 0 \ldots \mathrm{len}(S)$ **do**
  for $i \leftarrow 0 \ldots S[c]$ **do**
    for $j \leftarrow 0 \ldots m$ **do**
      Sample a sentence from the corpus $x \sim \mathbb{P}_r$, truncate its length to ℓ;
      $\mathbf{X}_r \leftarrow W_e x$, $\mathbf{X}_r \in \mathbb{R}^{\ell \times d}$;
      $z \sim \mathcal{N}(0, I_d)$, $\mathbf{X}_g \leftarrow G_\theta(z)$;
      $L^{(j)} \leftarrow \mathcal{L}_D^{\text{wgan-gp}}$
    end
    $\phi \leftarrow \mathrm{Adam}(\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} L^{(j)}, \alpha, \beta_1, \beta_2)$;
    Sample a batch of noise vectors $\mathbf{Z} = \{z^{(1)}, \ldots z^{(m)}\} \sim \mathcal{N}(0, I_d)$;
    $\theta \leftarrow \mathrm{Adam}(\nabla_\theta \mathcal{L}_G^{\text{wgan-gp}}(\mathbf{Z}))$
  end
  $\ell \leftarrow \ell + 1$;
end

4 Experiments

We apply ConGAN on the task of unconditional text generation. That is, given a human-generated text corpus, we aim to train a model to generate language matching the distribution of the text in the corpus. We experiment with a standard dataset of COCO Image Captions [20]

4.1 Dataset

COCO Image Captions COCO Image Captions is a dataset containing simple sentences which are collected by asking users to write captions for images (we train the model just on the captions, not images). Both training and test set contain 10,000 sentences with longest sentence being 37 words and a vocabulary of size 4682.

4.2 Evaluation

Text generation GAN models are usually evaluated along two dimensions: quality and diversity. Quality measures how fluent, grammatical and coherent the generated text is. And diversity measures the coverage of the model. Ideally we want a generative model that can generate very high quality and diverse text. But it has been studied in [13, 14] that there is always a trade-off between the two, especially for softmax-based models where one can use temperature parameter to increase quality at the cost of diversity and increase diversity at the cost of quality. We evaluate our proposed model on four metrics consistently used in prior work, two of which are measuring quality: BLEU [42] and LM-score and the other two are measuring diversity: self-BLEU and Reverse LM-score [14]; they are described in the appendix.

4.3 Implementation Details

For representing each word, we use 300-dimensional fasttext embeddings [36] trained on a large monolingual English corpus as used by [16] with around more than 4 billion words. Most of the documents used in this training are news articles.

Table 1: The quality scores (BLEU and LM) obtained on the test sets by training the above models with the GAN objectives with (right) and without pretraining (left). BLEU is higher the better and LM-Score is lower the better. MLE and RelMLE indicate models which were only pretrained their scores are shown on the right side in each cell. Our proposed model consists of only GAN training for which we report the performance on the right side of each cell. There is no training in "Random", we just initialize the generator randomly and decode from it

	BLEU-2		**BLEU-3**		**BLEU-4**		**BLEU-5**		**LM-SCORE**	
Random	0.041	-	0.017	-	0.011	-	0.008	-	14.98	-
SeqGAN	0.043	0.745	0.016	0.498	0.012	0.294	0.006	0.18	13.05	4.09
RankGAN	0.040	0.743	0.014	0.467	0.012	0.264	0.007	0.156	13.43	5.16
LeakGAN	0.045	0.746	0.017	0.528	0.014	0.355	0.005	0.230	12.45	8.44
RelGAN	0.590	**0.849**	0.280	**0.687**	0.141	**0.502**	0.094	**0.331**	7.4	**3.98**
MLE	-	0.748	-	0.656	-	0.456	-	0.320	-	4.6
RelMLE	-	0.810	-	0.602	-	0.457	-	0.293	-	4.2
ConGAN	**0.712**	-	**0.487**	-	**0.315**	-	**0.210**	-	**4.51**	-

For the RNN-based generator, we have three components: (1) A multilayer perceptron to convert the noise vector z to the initial hidden representation $\mathbf{h}_0$. z is 300 dimensional and the multilayer perceptron has the following structure: `(Linear(300, 300), LeakyReLU(0.2), Linear(300, 300), LeakyReLU(0.2), Linear(300, 300))`. (2) The generator is a 2-layered LSTM with a word embedding and hidden vector of size 300. (3) The hidden vector is transformed into an output vector using another multilayer perceptron with the following structure: `(Linear(300, 300), LeakyReLU(0.2), Linear(300, 300), LeakyReLU(0.2), Linear(300, 300))`

For the CNN-based discriminator, we use filter windows of sizes 3, 4, 5 and 6 and 300 feature maps for each. We use leaky ReLU as our activation function with a leak value of 0.2. The final feature representation is 1200 dimensional which is then transformed to a single dimensional scalar using a fully connected layer.

For training, we use Adam optimizer [43] with a learning rate of 0.0001 for both the generator and the discriminator with $(\beta_1 = 0.5, \beta_2 = 0.9)$. We apply gradient clipping if the norm of the gradients exceeds 5. We train the model for 30000 steps without any pretraining. This involves 5 steps of discriminator updates followed by one generator update. For $\ell in\{1 \ldots 20\}$ (see 1), we train the model for 1000 steps each. After that, we train the model with the full length sequences.

5 Results

5.1 Baselines

We compare against 4 previously proposed GAN models which can be divided into two categories: (1) Trained with REINFORCE – SeqGAN, TextGAN, and RankGAN and (2) Differentiable training using Gumbel-softmax – RelGAN (with hyperparameter $\beta = 100$). Note that all these models require pretraining with a language model objective before even starting GAN training. We compare against all these models in three settings (1) GAN training without pretraining, (2a) only pretraining, and (2b) GAN training after pretraining, to measure how much the models are improved after pretraining. For LSTM (with softmax) based models, the pretraining setup is the same for SeqGAN, TextGAN and RankGAN and we report it as MLE. For RelGAN, the generator architecture is a relational memory augmented RNN [23], which we refer to as RelMLE. We stress that in our proposed model, we report results without pretraining. This is because in softmax based models where sampling is the source of diversity, initial hidden state is usually fixed and initialized to $\mathbf{0}$ which makes the model architecture identical to a language model and suitable for pre-training the way language models are trained. But in our proposed model, we follow a more traditional GAN approach of feeding in a noise vector z which initializes the initial hidden state.

Table 2: The diversity scores (self-BLEU, shown as SBLEU, and RLM score) obtained on the test set by training the above models with the GAN objectives with and without pretraining. In both scores lower is better.

	SBLEU-2	**SBLEU-3**	**SBLEU-4**	**SBLEU-5**	**RLM-Score**
SeqGAN	0.88/0.95	0.85/0.84	0.82/0.67	0.75/0.49	10.45/5.64
RankGAN	0.87/0.96	0.81/0.88	0.80/0.76	0.78/0.62	11.09/5.40
LeakGAN	0.94/0.97	0.90/0.91	0.86/0.85	0.85/0.78	13.75/5.16
RelGAN	0.97/0.94	0.92/0.81	0.85/0.70	0.80/0.69	9.30/6.44
ConGAN	0.97/-	0.94/-	0.85/-	0.79/-	7.29/-

5.2 Quality

We train ConGAN as well as all the baselines 6 times with different random initializations. Table 1 show the mean quality metrics obtained. For softmax based models, we generate multiple sets of samples by considering the temperature parameter $\tau \in \{0.01, 0.1, 1.0, 2.0, 5.0, 20.0\}$ and report the best scores. For each metric, the score on the left is without pretraining and the score on the right is with pretraining with a language model. Clearly, our proposed model, ConGAN, outperforms all the other models across all the metrics of quality without pretraining. The models based on RL perform no better than random. If we compare the prior models after pretraining the language model, LeakGAN and RelGAN perform better than the proposed model. But as has been previously studied [14], LeakGAN performs worse than a properly tuned MLE model. Also we observe that RelMLE performs quite well if not better than RelGAN. This suggests that the gains obtained in RelGAN are mainly due to replacing the generator with with a better language model rather than due to GAN training itself. Despite the gains obtained with our proposed model without pretraining, the model overall performs worse than a MLE trained model as well RelGAN. In the next section, we examine reasons for this performance as well as suggest directions of future research in this direction.

We observe the except for RelGAN, the performance of all the baseline models remains around their initialization points and does not seem to converge. Conversely, both RelGAN and the proposed ConGAN are much more stable across 6 runs with a standard deviation in the range of 0.01 to 0.02 for all BLEU-X scores.

5.3 Diversity

Table 2 shows the comparison of diversity scores. The numbers shown are means of 6 different runs. Although the diversity numbers look better without pretraining, it is because of high variance of the RL models and the difficulty in convergence resulting in very poor quality albeit diverse samples. Over the 6 runs, we observe a standard deviation of 0.2-0.5 in BLEU scores in all the compared models whereas it was around 0.02 for our proposed model. The source of diversity in ConGAN is the noise vector z as opposed to the sampling process in softmax based models. But all the diversity scores leave something to be desired; leading to the conclusion that mode collapse is still very much an issue.

6 Discussion

By manually inspecting several samples generated by ConGAN, we find that many of them are fluent and grammatical, but they produce sentences that are not meaningful; the latter being true for majority of text GANs. Additionally, as indicated in table 1 and table 2, our model doesn't show improvement over MLE. We attribute the poor performance to the following major factors:

6.1 Granularity of signal given by the discriminator and architectural choices

We train ConGAN with a binary signal from the discriminator: is the generated sample real? While this has shown to work well for continuous data like images, it has been argued that such a signal is not dense enough for text generation where in the beginning of training when the discriminator can easily distinguish between real and fake samples [44]. A generator is more amenable to training with a token or phrase level loss [10]. By designing a curriculum to generate good prefixes, we already

incorporate sub-sentence information in the training process. To explicitly incorporate these signals, we additionally explore two changes to the discriminator architectures:

CNN with multiple outputs In the described CNN discriminator, we add a fully connected layer after every CNN layer (filter + max pooling) to predict whether the sequence spanned by the current layer is real or not. We sum all such signals to get the final discriminator output [45].

LSTM discriminator Prior work has shown that the most suitable discriminator is the one with the same architecture and capacity as the generator [46]. Hence, we encode the sample with an LSTM and predict at every step whether the sequence so far is real. We again sum all the signals to get the final discriminator output. A similar loss is explored by [10] where they train the model using RL.

We train the model using the loss described in (2) as well as standard GAN loss [3]. With the standard GAN loss, we observe the model suffers severe mode collapse in both cases. With WGAN-GP loss, we didn't observe any significant improvement in performance with the new CNN discriminator beyond what we already report in table 1. We speculate this is likely because we design the training curriculum to predict good prefixes anyway. We see that the LSTM model fails to train altogether. Our analysis as well as previous observations [47] indicate that lipschitz continuity is difficult to enforce on recurrent architectures using gradient penalty (or otherwise) as the gradient values become too small leading to no updates.

6.2 Wasserstein distance metric

As shown in (1), Wasserstein distance between two distributions is function of a distance metric between its samples. Most work on GANs assume this distance to be ℓ^2. This assumption has held well for fixed dimensional data like images but it is clearly not well defined for two sequences of word (embeddings) which can be of different lengths. In fact, defining distance between two input text is an active area of research [48]. Consequently, the dual form of Wasserstein distance with the the gradient penalty term in (2), which we use to train our proposed model, penalizes the ℓ^2 norm of the gradient, which corresponds to choosing ℓ^2 distance as the distance between samples. Some research has been done on generalizing GAN theory to other spaces [49], but in its current form WGAN with gradient penalty does not extend to arbitrary choices of underlying spaces. This fact along with difficulty of ensuring lipschitz continuity of a recurrent discriminator, explains the poor performance of our proposed method and calls for exploration of extending WGANs beyond ℓ_2 norm [50]

Along with the above mentioned major factors, there are several other potential reasons for poor performance of text GANs in general such as (1) the generator not being expressive enough and cannot deal with longer term dependencies in the text and the (2) the discriminator is not being grounded in commonsense knowledge. Such issues are orthogonal to the ones we aimed to address in this work and need additional exploration.

7 Conclusion

In this work, we introduce ConGAN: a modification to neural language generation systems to predict word embeddings instead of softmax distribution which makes GANs for text generation differentiable and the training more stable. We show that all the previous GAN models draw most of their power from language model pretraining, and degrade substantially when using GANs objective. Recognizing that ConGANs are still outperformed by softmax-based GANs which rely on the language modeling power, we highlight that it is GAN training objectives are limited by their simplying assumptions about the data and model architecture. We plan to explore and address these issues in the future.

References

[1] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks, 1989.

[2] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *Proc. ICLR*, 2020.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. NeurIPS 2014*, 2014.

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *Proc. ICLR*, 2016.

[5] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *Proc. ICLR*, 2019.

[6] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proc. CVPR*, 2018.

[7] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Proc. NeurIPS*, 2016.

[8] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *Proc. ECCV*, 2016.

[9] Sidi Lu, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu. Neural text generation: past, present and beyond. *arXiv preprint arXiv:1803.07133*, 2018.

[10] Cyprien de Masson dAutume, Shakir Mohamed, Mihaela Rosca, and Jack Rae. Training language gans from scratch. In *Proc. NeurIPS*. 2019.

[11] Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. Coldgans: Taming language gans with cautious sampling strategies, 2020.

[12] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. In *Proc. ICML*, 2017.

[13] Weili Nie, Nina Narodytska, and Ankit Patel. RelGAN: Relational generative adversarial networks for text generation. In *Proc. ICLR*, 2019.

[14] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. *arXiv preprint arXiv:1811.02549*, 2018.

[15] Stanislau Semeniuta, Aliaksei Severyn, and Sylvain Gelly. On accurate evaluation of gans for language generation. *arXiv preprint arXiv:1806.04936*, 2018.

[16] Sachin Kumar and Yulia Tsvetkov. Von mises-fisher loss for training sequence to sequence models with continuous outputs. In *Proc. of ICLR*, 2019.

[17] Gayatri Bhat, Sachin Kumar, and Yulia Tsvetkov. A margin-based loss with synthetic negative samples for continuous-output machine translation. In *Proc. WNGT*, 2019.

[18] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proc. ICML*, 2017.

[19] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Proc. NeurIPS*, 2017.

[20] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[22] 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.

[23] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Proc. NeurIPS*. 2018.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proc. NeuRIPS*. 2017.

[25] Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. 2020.

[26] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proc. AAAI*, 2017.

[27] Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 2019.

[28] 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.

[29] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. Adversarial ranking for language generation. In *Proc. NeurIPS*, 2017.

[30] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*, 2018.

[31] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *JMLR*, 1992.

[32] 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.

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

[34] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proc. EMNLP*, 2014.

[35] Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proc. NAACL-HLT 2015*, 2015.

[36] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5, 2017.

[37] Yoon Kim. Convolutional neural networks for sentence classification. In *Proc. EMNLP*, 2014.

[38] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *JMLR*, 2011.

[39] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. In *Proc. ICLR*, 2019.

[40] S.T. Rachev and R.M. Shortt. *Duality Theorems for Kantorovich-Rubinstein and Wasserstein Functionals*.

[41] 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.

[42] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proc. ACL*, 2002.

[43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[44] Zichao Yang, Zhiting Hu, Chris Dyer, Eric P Xing, and Taylor Berg-Kirkpatrick. Unsupervised text style transfer using language models as discriminators. In *Proc. NuerIPS*, 2018.

[45] Mikołaj Bińkowski, Jeff Donahue, Sander Dieleman, Aidan Clark, Erich Elsen, Norman Casagrande, Luis C. Cobo, and Karen Simonyan. High fidelity speech synthesis with adversarial networks. In *Proc. ICLR*, 2020.

[46] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. In *Proc. NeurIPS*. 2019.

[47] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of neural networks by enforcing lipschitz continuity, 2020.

[48] Weiwei Hu, Anhong Dang, and Ying Tan. A survey of state-of-the-art short text matching algorithms. In Ying Tan and Yuhui Shi, editors, *Data Mining and Big Data*, 2019.

[49] Calvin Seward, Thomas Unterthiner, Urs Bergmann, Nikolay Jetchev, and Sepp Hochreiter. First order generative adversarial networks, 2018.

[50] Jonas Adler and Sebastian Lunz. Banach wasserstein gan, 2019.

Appendix: Evaluation Metrics

BLEU BLEU score [42] is a standard evaluation metric in machine translation which measures the quality of the generated translation based on counts of n-grams that match with a set of reference translations which are known in advance (this is n-grams precision measure). We use the same setting but consider the entire test set as the reference. We compute BLEU score with 2, 3, 4 and 5-grams.

Language Model (LM) Score BLEU score is a surface level metric and limited by short n-grams, and sometimes give an overestimate of the actual quality of the model. Hence, we also evaluate on LM score, which is the negative log-likelihood of the generated samples on a pretrained language model on the training corpus. We use a simple LSTM based language model with the same settings as that of softmax based generator used in the baselines.

Self-BLEU Both BLEU score and LM score measure the quality of the generated samples but even if the model generates one high quality sample every time, it would score on both the said metrics. Hence, we use self-BLEU to measure the diversity of the generated samples. In the set of generated samples, consider each sentence as hypothesis and the others as reference, calculate BLEU score for every generated sentence. Self-BLEU is defined as their average over the generated corpus. A low self-BLEU suggests better diversity. We compute self-BLEU with 2, 3, 4 and 5-grams

Reverse Language Model (RLM) Score Although, self-BLEU measures diversity to an extent but it is again a surface level metric. Moreover, consider a corpus of $10,000$ generated sentences where each sentence occurs exactly twice and no more. If we compute self-BLEU on this corpus, we will get a value of 1.00 even though there are 5000 unique sentences in the corpus. Reverse LM score was proposed to handle such cases [14]. It is defined as the negative log likelihood obtained on the real test data using a language model trained on the generated samples. This is a measure of both quality as well as diversity. A generated sample which mimics the real test set both in quality as well diversity will achieve high RLM score.