Boundary-Seeking Generative Adversarial Networks

R Devon Hjelm [*] [1 2 3] **Athul Paul Jacob** [*] [2 4] **Tong Che** [1 2] **Kyunghyun Cho** [5] **Yoshua Bengio** [6 1 2 3]

Abstract

We introduce a novel approach to training generative adversarial networks (GANs, Goodfellow et al., 2014), where we train a generator to match a target distribution that converges to the data distribution at the limit of a perfect discriminator. This objective can be interpreted as training a generator to produce samples that lie on the decision boundary of a current discriminator in training at each update, and we call a GAN trained using this algorithm a boundary-seeking GAN (BGAN). This approach can be used to train a generator with discrete output when the generator outputs a parametric conditional distribution. We demonstrate the effectiveness of the proposed algorithm with discrete image data. In contrary to the proposed algorithm, we observe that the recently proposed Gumbel-Softmax technique for re-parametrizing the discrete variables does not work for training a GAN with discrete data. Finally, we notice that the proposed boundary-seeking algorithm works even with continuous variables, and demonstrate its effectiveness with two widely used image data sets, SVHN and CelebA.

1. Introduction

Generative models provide a means of representing the underlying structure in data.While the objective is to faithfully generate data with representative statistics, the strength of generative models lies partially in representing the data as a hypothetical compressed version of its underlying structure. Generative adversarial networks (GAN, Goodfellow et al., 2014) are a unique generative learning framework that use two separate models with opposing, competing, or *adversarial* objectives.

[*]Equal contribution [1]University of Montreal, Montreal, Quebec, Canada [2]Montreal Institute for Learning Algorithms [3]IVADO [4]University Of Waterloo [5]New York University [6]CIFAR. Correspondence to: R Devon Hjelm <erroneus@gmail.com>.

In contrast to those models that generate the data from a parameteric distribution, such as directed graphical models (e.g., Helmholtz machines (Dayan et al., 1995) and variational autoencoders (VAEs, Kingma & Welling, 2013)) or undirected graphical models (e.g., restricted Boltzmann machines (RBMs, Hinton, 2002) and deep Boltzmann machines (DBMs, Salakhutdinov & Hinton, 2009)), GANs do not rely on maximum likelihood estimation (MLE, Dempster et al., 1977) for learning. Rather, training a GAN, which consists of two competing network–discriminator and generator–, only requires back-propagating a learning signal originating from a learned objective function corresponding to the loss of a discriminator trained in an adversarial way.

This framework is powerful, as it trains a generator without relying on an explicit formulation of the probability function which requires marginalizing out latent variables. Also, it avoids the issue with the MLE objective which makes an MLE-trained model conservative in where they put probability mass. The latter issue is known to be a problem behind MLE-trained models generating samples that lack real-world qualities, often evidenced by the lack of sharpness in natural images (Goodfellow, 2016). Unlike MLE-trained models, GANs have been shown to generate often-diverse and realistic samples even when trained on high-dimensional large-scale data (Radford et al., 2015) in the case of continuous variables.

GANs however has a serious limitation on the type of variables they can model, because it requires the composition of the generator and discriminator to be fully differentiable. With discrete variables this is not true. For instance, consider using a step function at the end of a generator in order to generate a discrete value. In this case, back-propagation alone cannot provide the training signal for the generator, because the derivative of a step function is 0 almost everywhere (and so would be the back-propagated signal.) The general issue of credit assignment for computational graphs which involve discrete operations (e.g. discrete stochastic neurons) is difficult, and only several approximate solutions have been proposed in the past (Bengio et al., 2013; Gu et al., 2015; Gumbel & Lieblein, 1954; Jang et al., 2016; Maddison et al., 2016).

This situation is analogous to that in training Helmholtz machines with discrete hidden variables (i.e., sigmoid belief networks (SBNs, Saul et al., 1996)), where advanced learning algorithms are necessary to train an approximate inference network (Mnih & Gregor, 2014; Bornschein & Bengio, 2014). When those hidden units are continuous, usual backpropagation with a re-parametrization technique works (Kingma & Welling, 2013). On the other hand, when they are discrete, only recently an approximate re-parametrization technique, called "Gumbel-Softmax technique", was introduced (Gumbel & Lieblein, 1954; Jang et al., 2016; Maddison et al., 2016). The effectiveness of this approach was only demonstrated with simple directed models having a single layer of discrete latent variables, and there is no consensus nor evidence whether it works with GANs.

In this paper, we introduce a novel learning algorithm for GANs. This algorithm estimate the gradient of the discriminator's output with respect to the generator, as the weighted sum of the gradients of the log-probabilities of the samples generated from the generator. The weights are given by the discriminator's performance on the samples, such that those samples, that look similar to true, training examples, are weighted more (because they would be scored highly by the discriminator) and vice versa. This lets us interpret the discriminator as computing a target distribution for the discrete stochastic units, allowing us to train a generator to converge toward the hypothetical true distribution of the data as the discriminator is optimized. This distribution was discovered independently from us by Che et al. (2017) in the context of language modeling.

This objective can further be interpreted as an alternative loss on the discriminator output, which allows us to derive a novel learning algorithm for GANs even with continuous variables. Under this interpretation, the learning objective of a generator is to minimize the difference between the discriminator's log-probabilities for the sample being positive and negative. This objective can be thought of as training the generator to *place generated samples at the decision boundary of the discriminator*, and hence we call a GAN trained with the proposed algorithm a boundary-seeking generative adversarial network (BGAN).

We demonstrate the effectiveness of the proposed BGAN in two settings; discrete and continuous variables. We use MNIST and quantized CelebA for the discrete setting, and SVHN and original, continuous CelebA for the continuous setting. In the case of discrete variables, we also try the Gumbel-Softmax technique, however, without much success, unlike the proposed BGAN which we found extremely easy to train. In the case of continuous variables, we observe that the proposed algorithm works qualitatively as well as the conventional GAN, suggesting the generality of the proposed learning algorithm for both discrete and continuous GANs.

2. Methods

2.1. Generative Adversarial Networks

Generative adversarial networks (GANs, Goodfellow et al., 2014) are a two-model generative framework where a generator model is pitted against a discriminatory adversary. The generator is composed of a prior distribution, $p(\mathbf{z})$, and a generator function, $G(\mathbf{z})$, which maps from the space of $\mathbf{z}$ to the space of $\mathbf{x}$. The objective of the discriminator is to correctly distinguish between data and samples from the generator model by maximizing:

$$\mathbb{E}\left[\log D(\mathbf{x})\right]_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})} + \mathbb{E}\left[\log(1 - D(G(\mathbf{z}))\right]_{\mathbf{z}\sim p(\mathbf{z})}, \tag{1}$$

where $D(.)$ is a discriminator function with output that spans $[0, 1]$. The generator is trained to "fool" the generator, that is minimize $\mathbb{E}\left[\log(1 - D(G(\mathbf{z}))\right]_{\mathbf{z}\sim p(\mathbf{z})}$ or the easier to minimize proxy $\mathbb{E}\left[-\log(D(G(\mathbf{z}))\right]_{\mathbf{z}\sim p(\mathbf{z})}$. Ignoring the proxy case, the two models play a minimax game with value function:

$$\min_G \max_D V(G, D) = \mathbb{E}[\log D(\mathbf{x})]_{\mathbf{x}\sim p(\mathbf{x})} + \mathbb{E}[\log(1 - D(G(\mathbf{z})))]_{\mathbf{z}\sim p(\mathbf{z})}. \tag{2}$$

Previous deep generative models like the variational auto-encoder (VAE) (Kingma & Welling, 2013) involve a neural network which takes the latent sampled values $\mathbf{z}$ and outputs parameters of a distribution (e.g. Bernoulli, Gaussian) from which the samples are drawn. Instead, the GAN generator is defined by the composition of sampling the latent values $\mathbf{z}$ and applying the deterministic function G. The generative objective involves one continuous function with parameters from both models: $D(G(.;\psi);\phi)$ but optimized over ϕ only. This allows for training a generative model with a relatively simple fitness function and backpropagation, rather than a complex marginalized likelihood function, and which does not suffer from some of the learning difficulties of maximum likelihood estimators (MLE), e.g., giving rise to blurred samples.

In practice, the generator, rather than being trained to minimize the above value function, can be trained to maximize the proxy $\log(D(G(\mathbf{z})))$ or $\log(D(G(\mathbf{z}))) - \log(1 - D(G(\mathbf{z})))$, which can alleviate some learning issues related to the discriminator loss saturating early in learning and improve stability. With basic GANs, both generator and discriminator are feedforward networks, $D(.;\boldsymbol{\phi})$ and $G(.;\boldsymbol{\psi})$, while in deep convolutional GANs (DCGAN, Radford et al., 2015), the discriminator and generator are convolutional neural networks with strided and fractionally-strided convolutional layers, respectively.

In order for a generating function, $G(.;\boldsymbol{\psi})$, with real-valued parameters, $\boldsymbol{\psi}$, to generate discrete-valued outputs, $\mathbf{x}$, $G(.;\boldsymbol{\psi})$ cannot be fully continuous. A natural choice for the generator would be a continuous function, $\mathbf{y} = f(\mathbf{z})$ (e.g., a feed forward network), followed by a step function at 0.5, so that:

$$x_i = \phi(y_i) = \begin{cases} 1, & \text{if } y_i \geq 0.5 \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $\mathbf{x} = \{x_i\}$ and $\mathbf{y} = \{y_i\}$. However, as the gradients that would otherwise be used to train the generator do not naturally back-propagate through discontinuous functions, it is not possible to train the generator from the discriminator error signal. Approximations for the back-propagated signal exist (Bengio et al., 2013; Gu et al., 2015; Gumbel & Lieblein, 1954; Jang et al., 2016), and we discuss comparisons in more detail in the Results section. We propose a novel approach to dealing with this issue, based on estimating a proposal target distribution for the generator output.

2.2. Target distribution for the generator

First consider the optimum discriminator function (Goodfellow et al., 2014), $D(\mathbf{x})^\star$, derived from analytically optimizing the discriminator objective given a fixed generator $G(.)$ and assuming no functional constraints on the discriminator, i.e., in a non-parametric setting:

$$D_G^\star(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}, \tag{4}$$

where $p_{\text{data}}(\mathbf{x})$ is the data distribution, and $p_g(\mathbf{x})$ is the distribution as generated by $p(\mathbf{z})$ and $G(\mathbf{z})$. Given this optimal discriminator, the density of the data can then be written as:

$$p_{\text{data}}(\mathbf{x}) = p_g(\mathbf{x}) \frac{D_G^\star(\mathbf{x})}{1 - D_G^\star(\mathbf{x})}. \tag{5}$$

Which indicates that, in the limit of a perfect discriminator, the true distribution can be perfectly estimated from said discriminator and any fixed generator, even if that generator is not perfectly trained. *A sample from that corrected estimator can thus be obtained by reweighing samples from the generator according to the ratio* $\frac{D_G^\star(\mathbf{x})}{1-D_G^\star(\mathbf{x})}$.

However, it is unlikely that such a perfect discriminator is learnable or even exists. One can then posit the following estimator for the true distribution, given an imperfect discriminator, $D(\mathbf{x})$:

$$\tilde{p}(\mathbf{x}) = \frac{1}{Z} p_g(\mathbf{x}) \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}, \tag{6}$$

where the normalization constant,

$$Z = \sum_{\mathbf{x}} g_g(\mathbf{x}) \frac{D(\mathbf{x})}{1 - D(\mathbf{x})},$$

guarantees that $\tilde{p}(\mathbf{x})$ is a proper probability distribution. This distribution was discovered independently from us by Che et al. (2017). Note that the optimum for the generator occurs at $D(\mathbf{x}) = D^\star(\mathbf{x}) = 1/2$, so that $Z = 1$ and $\tilde{p}(\mathbf{x}) = p_g(\mathbf{x}) = p(\mathbf{x})$. $\tilde{p}(\mathbf{x})$ is a potentially biased estimator for the true density. However, *the bias only depends on the quality of* $D(\mathbf{x})$: the closer $D(\mathbf{x})$ is to $D^\star(\mathbf{x})$, the lower the bias. This is an important consideration because it is likely that training the discriminator (a standard probabilistic binary classifier) is much easier than training the generator (whose objective function is a moving target as the discriminator adapts to the generator and vice-versa). The optimum for the generator exists when generated samples have equal probability of being classified as data or as samples. This occurs at the decision boundary of the discriminator, so that we call this approach boundary-seeking generative adversarial networks (BGAN).

2.3. BGAN with Discrete variables

The above target distribution reveals a straightforward learning algorithm when $p_g(\mathbf{x})$ is known. For discrete variables, parameterizing the generating distribution directly reveals a method for training the generator without relying on back-propagation through the generator's discrete output.

As with previous work on evaluating GANs (Wu et al., 2016), we parameterize $p_g(\mathbf{x})$ as a the marginalization of a joint density, $p_g(\mathbf{x}) = \sum_{\mathbf{z}} g(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, rephrasing the generator function, $G(\mathbf{z})$, as a conditional distribution, $g(\mathbf{x}|\mathbf{z})$. To minimize the distance between $\tilde{p}(\mathbf{x})$ and $p_g(\mathbf{x})$, we can minimize the KL divergence, so that the

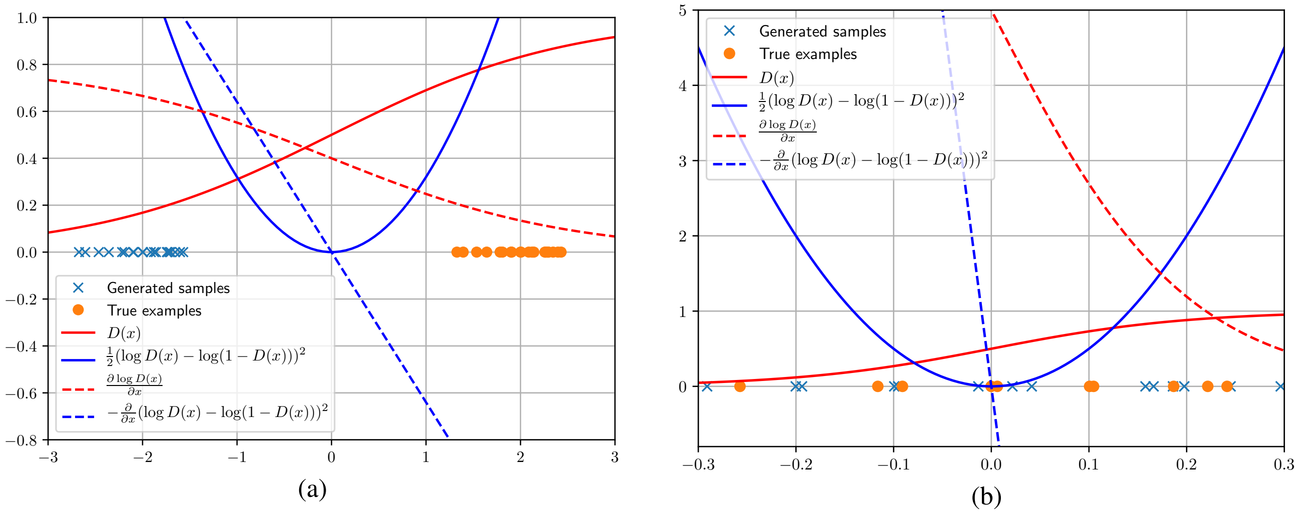


Figure 1. Qualitative comparison between the conventional GAN and the proposed BGAN in 1-D examples. The discriminator D is fixed to a logistic regression classifier with a coefficient 0.8. Note that the maximum of the generator objective from the conventional GAN is at the positive infinity (red curve), while the minimum from the proposed BGAN is at the decision boundary (blue curve). Unlike the proposed BGAN, the learning gradient of the conventional GAN (red dashed curve) pushes the generated samples beyond the real samples.

gradients w.r.t. the generator parameters, ψ, are:

$$\begin{aligned}\nabla_\psi D_{KL}(\tilde{p}(\mathbf{x})||p_g(\mathbf{x})) &= \nabla_\psi \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log \frac{\tilde{p}(\mathbf{x})}{p_g(\mathbf{x})} \\ &\approx -\sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \nabla_\psi \log p_g(\mathbf{x}) \\ &= -\sum_{\mathbf{x}} \frac{1}{Z} p_g(\mathbf{x}) \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \nabla_\psi \log p_g(\mathbf{x}) \\ &= -\sum_{\mathbf{x}} \sum_{\mathbf{z}} \frac{1}{Z} g(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \nabla_\psi \log p_g(\mathbf{x}),\end{aligned} \tag{7}$$

where $\tilde{p}(\mathbf{x})$ is held fixed. The gradients would require some sort of Monte-Carlo (MC) estimator across the input and output noise and will have high variance, notably due to the partition function, Z. The intuition here is to note that, as the conditional density, $g(\mathbf{x}|\mathbf{z})$ is unimodal, it can be used to define a unimodal distribution analogous to that of Equation 6. Following this intuition and by inspection of Eq. 7 for a given sample $\mathbf{z}$ from $p(\mathbf{z})$, we propose updating the conditional, $g(\mathbf{x}|\mathbf{z})$, to fit a *mode* of the data as defined by:

$$\tilde{p}_{\mathbf{z}}(\mathbf{x}) = \frac{1}{Z_{\mathbf{z}}} g(\mathbf{x}|\mathbf{z}) \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}, \tag{8}$$

where we have used $\log p_g(\mathbf{x}) = \log g(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z})$ (and the second term is a constant in the parameters), $\tilde{p}_{\mathbf{z}}$ is an estimate of the data in the neighborhood of the mode defined by $\mathbf{z}$, and $Z_{\mathbf{z}} = \sum_{\mathbf{x}} g(\mathbf{x}|\mathbf{z}) \frac{D(\mathbf{x})}{1-D(\mathbf{x})}$ is the marginal that ensures $\tilde{p}_{\mathbf{z}}$ is a proper probability distribution. The gradients then become:

$$\begin{aligned}\nabla_\psi D_{KL}(\tilde{p}_{\mathbf{z}}(\mathbf{x})||g(\mathbf{z}|\mathbf{x})) &\approx -\sum_{\mathbf{x}} \tilde{p}_{\mathbf{z}}(\mathbf{x}) \nabla_\psi \log g(\mathbf{x}|\mathbf{z}) \\ &\approx -\sum_m \tilde{w}^{(m)} \nabla_\psi \log g(\mathbf{x}^{(m)}|\mathbf{z}),\end{aligned}$$

where

$$\tilde{w}^{(m)} = \frac{w^{(m)}}{\sum_{m'} w^{(m')}}$$

and

$$w^{(m)} = \frac{D(\mathbf{x}^{(m)})}{1 - D(\mathbf{x}^{(m)})}$$

are the normalized and unnormalized importance weights respectively (note how $Z_{\mathbf{z}}$ cancels in the normalization) and $\mathbf{x}^{(m)}$ are samples from the generator for the given $\mathbf{z}$. Note that the normalization can help generate a relative learning signal, which will help in the case that the samples are all fairly bad from the point of view of $D(\mathbf{x})$. The gradients can be computed over a mini-batch of samples from $p(\mathbf{z})$, so that the updates to the generator parameters, ψ, become:

$$\Delta\psi \propto \frac{1}{N} \sum_n \sum_m \tilde{w}^{(m)} \nabla_\psi \log g(\mathbf{x}^{(m)}|\mathbf{z}^{(n)}). \tag{9}$$

This reveals a straightforward procedure for training discrete GANs with a parametric conditional generator without relying on back-propagation. Compared to normal GANs, this method requires M times more space but computing the M (instead of 1) scalar values of $D(\mathbf{x}^{(m)})$ can be parallelized, as samples from $g(\mathbf{x}|\mathbf{z})$ can be drawn independently.

2.4. Continuous variables

For continuous variables, we could introduce a conditional distribution as we did in the discrete case, defining importance weights, and computing a gradient directly at the generator output using the KL divergence. However, this would abandon some of the strengths central to GANs relative to other generative models with continuous data (Goodfellow, 2016).

While $p_g(\mathbf{x})$ is unavailable to us, a simple analysis of Equation 6 shows that the distance (by a number of metrics) between $\tilde{p}(\mathbf{x})$ and $p_g(\mathbf{x})$ is minimized when $\frac{1}{Z}\frac{D(\mathbf{x})}{1-D(\mathbf{x})} = 1$. But, as noted, as the ratio $\frac{D(\mathbf{x})}{1-D(\mathbf{x})}$ converges to one, so does the partition function. This reveals an alternative learning objective to that of normal GANs for the generator with continuous data. There are a number of loss functions that would be suitable, and we have experimented with a simple squared-error loss:

$$\mathcal{L}_g(\mathbf{x}) = \frac{1}{2}\left(\log D(\mathbf{x}) - \log(1 - D(\mathbf{x}))\right)^2. \qquad (10)$$

The optimum of this loss corresponds exactly with the optimum of the KL divergence, $\mathcal{D}_{KL}(\tilde{p}(\mathbf{x})||p_g(\mathbf{x}))$. The above loss function can be used as an alternative to the usual GAN generator objective for continuous data, where we effectively minimize the objective function of the discriminator, and this worked well across our experiments described below.

Visual comparison: GAN vs. BGAN In the case of continuous variables, we can qualitatively analyze the difference between the conventional GAN and the proposed boundary-seeking GAN. We consider an one-dimensional variable, and draw 20 samples as each of real and generated sets of samples from two Gaussian distributions. We consider two cases; (a) early stage of learning, and (b) late stage of learning. We use -2 and 2 in the first case, and -0.1 and 0.1 in the second case, for the centers of those two Gaussians. The variances were set to 0.3 for both distributions. Instead of learning, we simply set our discriminator $D(x)$ to

$$D(x) = \frac{1}{1 + \exp(-cx)},$$

where c is set to 0.8 and 10 for the cases (a) and (b), respectively. These values were selected to illustrate sub-optimal discriminators, which is almost always true when training a GAN on a real data.

In Fig. 1, we plot both real and generated samples on the x-axis ($y = 0$). The solid red curve corresponds to the discriminator $D(x)$ above, and its log-gradient ($\partial \log D(x)/\partial x$) is drawn with a dashed red curve. It is clear that maximizing $\log D(x)$, as conventionally done with GAN, pushes the generator beyond the real samples (orange circles). On the other hand, the proposed criterion from Eq. (10) has its minimum at the decision boundary of the discriminator. Minimizing this criterion has the effect of pushing the generated samples, or correspondingly the generator, toward the real samples, but never beyond the region occupied by them. The issue is much more apparent in Fig. 1 (b), where the real and generated samples are extremely close to each other. The proposed BGAN encourages the generator to stay close to the center of real samples, while the conventional objective pushes the generated samples beyond the real samples. We conjecture that this property makes learning the proposed BGAN more stable.

3. Related Work

GAN for discrete variables Our approach introduces a new learning algorithm that allows for training generative adversarial networks (GANs) on discrete data, a first in the literature. Discrete data with GANs is an active area of research, particularly with language model data involving recurrent neural network (RNN) generators (Yu et al., 2016; Li et al., 2017). Our discrete solution is highly analogous to reweighted wake-sleep (RWS, Bornschein & Bengio, 2014), a highly successful approach for training Helmholtz machines with discrete variables (but not in a GAN framework). This proposed approach is further strengthened by the fact that it can be used for both discrete and continuous variables. From our framework, we were able to derive a new objective for training a generator in the case of continuous variables.

The Gumbel-Softmax technique is another approach to train Helmholz machines with discrete variables (Jang et al., 2016; Maddison et al., 2016), which effectively allows for training them as variational autoencoders (VAEs, Kingma & Welling, 2013). However, this technique has only been shown to work in very limited settings, and has yet to been shown to work well with GANs with discrete data. We study this question more thoroughly in the following section, showing negative results for the Gumbal-max technique on the simple task of generating MNIST digits. The quantized CelebA is available for download from `http://anonymous.url`.

Learning algorithms for GAN Since its introduction by Goodfellow et al. (2014), there have been a number of novel objectives for stabilizing GAN training. Arjovsky et al. (2017), within a few weeks of this submission, proposed a modification to the learning procedure of GAN based on the earth-mover's distance. Tolstikhin et al.

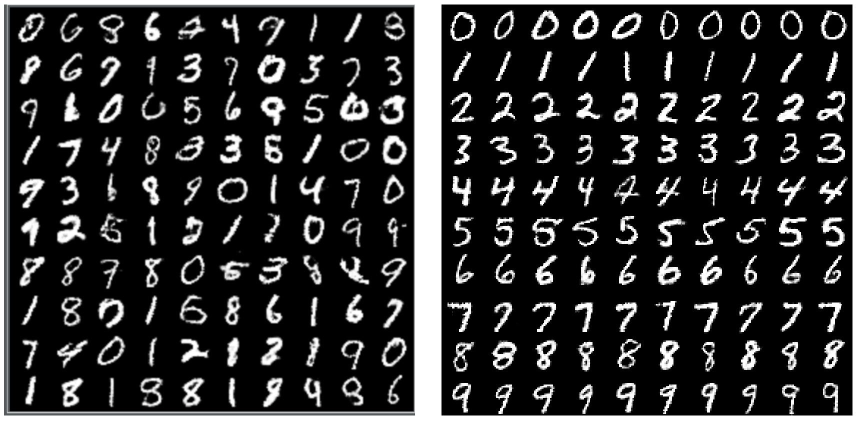


Figure 2. Left: Random samples from the generator trained as a boundary-seeking GAN (BGAN) with discrete MNIST data. Shown are the Bernoulli centers of the generator conditional distribution. Samples show high variability, with realistic generated hand-drawn digits. Right: Hand-picked samples of digits produced from the generator trained as a boundary-seeking GAN (BGAN). These were manually selected in order to illustrate the diversity of generated samples.

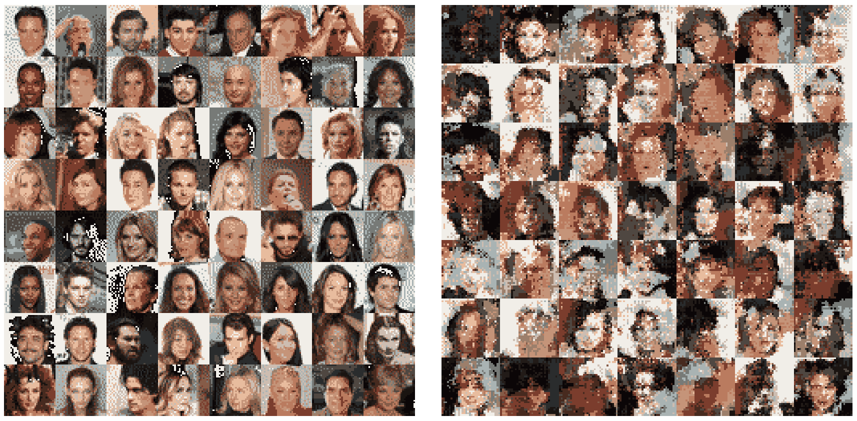


Figure 3. Left: Ground-truth 16-color (4-bit) quantized CelebA images downsampled to 32 $\times$ 32. Right: Samples produced from the generator trained as a boundary-seeking GAN on the quantized CelebA for 30 epochs. Samples are not perfectly realistic, but have good resemblance to the original examples. We also observe strong diversity.

(2017) proposed to iteratively re-weight training examples in order to train a series of GAN's, and showed that this procedure significantly stabilizes learning. Our work contributes to this pile of recent works, however, with a distinct goal of building a unified learning framework for both discrete and continuous variables.

4. Discrete variables

4.1. Datasets

We test our approach to training discrete generative adversarial networks (GANs) using two benchmarks: the common discretized MNIST dataset (Salakhutdinov & Murray, 2008) and a new quantized version of the CelebA dataset (see Liu et al., 2015, for the original CelebA dataset). For CelebA quantization, we first downsampled the images from 64×64 to 32×32. We then generated a 16-color palette using Pillow, a fork of the Python Imaging Project (https://python-pillow.org). This palette was then used to quantize the RGB values of the CelebA samples to a 4-bit representation (16 colors).

4.2. Setup

Our models used deep convolutional GANs (DCGAN, Radford et al., 2015). The generator is fed a vector of 100 i.i.d. random variables drawn from a uniform distribution, $[0, 1]$. For MNIST, the generator was composed of dense layers with 1024 and $128 \times 7 \times 7$ units followed by 2 fractionally-strided convolutional layers with 64 filters and 1 filter, respectively. For quantized CelebA, the generator was composed of a dense layer with $512 \times 4 \times 4$ units followed by 3 fractionally-strided convolutional layers with 256, 128, and 4 filters respectively. The fractional stride was chosen to ensure that the filter size doubled at each layer, so that the output was 4 channels of size 32×32. All layers other than the output layer of the generator were rectified linear units (ReLU), while all layers other than the output layer of the discriminator were leaky rectifiers. The discriminator used 3 convolutional layers, the structure of which was the transpose of that in the generator, followed by a dense layer with a single-unit output and a sigmoid nonlinearity.

We applied batch normalization to the generator, as is consistent in the literature. However, across our experiments with discrete datasets, we found that batch normalization in the discriminator made for unstable learning and worse results. We therefore omitted the batch normalization from the discriminator.

The model was trained using the Adam method for stochastic optimization (Kingma & Ba, 2014) with exponential decay rates of $\beta_1 = 0.95$ and $\beta_2 = 0.5$. MNIST

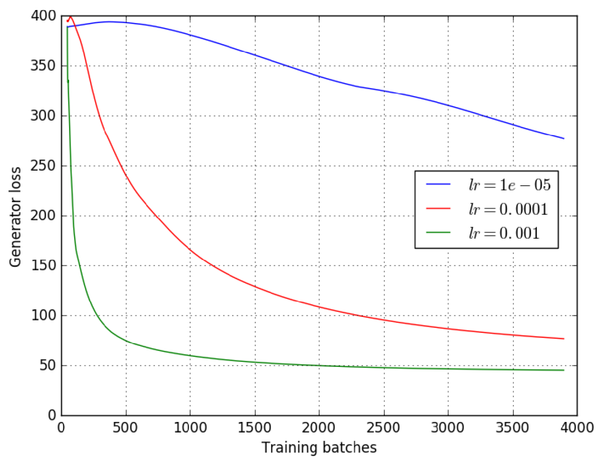


Figure 4. The evolution of the generator cost with varying learning rates when training a boundary-seeking GAN (BGAN). We observe a smooth convergence of the generator cost for all the learning rates, demonstrating the stability of learning with the proposed BGAN.

was trained for 200 epochs, while quantized CelebA was trained for 30 epochs. Unlike what is commonly done in the literature for training GANs, the generator and discriminator were updated the same number of times, without updating the discriminator multiple times for every generator update. We trained the model with learning rates of $1e{-}4$, $1e{-}5$, and $1e{-}6$ to test convergence properties. The number of samples per generated examples, M, was set to 20.

4.3. Generation result

Our results showed that training BGAN on discrete MNIST data is stable and produces realistic and highly variable generated handwritten digits. Figure 2 shows example digits from the model trained with a learning rate of $1e{-}4$, sampled randomly from the generator (left) and hand-picked (right) to demonstrate individual digit quality, respectively. The model was also relatively robust against learning rate, showing stable learning for the generator across learning rates tested (Figure 4).

For quantized CelebA, the generator trained as a BGAN did not produce perfectly realistic images, though they highly resemble the original dataset. In addition, the faces produced are highly variable, and show the type of diversity we expect from a good generator.

4.4. Comparison to Gumbel-Softmax Technique

The ”Gumbel technique“ (Gumbel & Lieblein, 1954; Jang et al., 2016) has been shown to be successful in some tasks where back-propogation through discrete random variables is not possible. Using the same model parameterization as our MNIST experiments above, we compare our approach to Gumbel-Softmax (Jang et al., 2016) and straight-through Gumbel-Softmax (Jang et al., 2016) on the binarized MNIST dataset. The temperature was annealed using the schedule $\tau = \max(0.5, \exp(-rt))$ as a function of the global training step t, where τ is updated every N steps. The values for the grid search of r and N are listed in Table 1.

Table 1. Hyperparameters used in our sweep of discrete GANs trained using the Gumbel-Softmax and straight-through Gumbel-Softmax

Optimizer	[Adam, RMSProp, SGD]
Learning Rate	[$1e{-}2$, $1e{-}3$, $1e{-}4$, $1e{-}5$]
Anneal Rate (r)	[0.01, 0.001, 0.0001]
Anneal Interval (N)	[200, 300, 500]

We observed that the generator cost of Gumbel-Softmax and straight-through Gumbel diverged across all hyperparameters (Figure 5). No set of hyperparameters produced a generator that was able to produce satisfactory representative samples of the original dataset.

5. Continuous Variables

5.1. Datasets

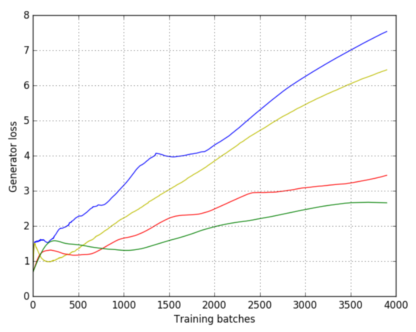
We tested boundary-seeking objective on two standard datasets for GANs: the streetview house number dataset (SVHN) and CelebA (Liu et al., 2015).

5.1.1. Setup

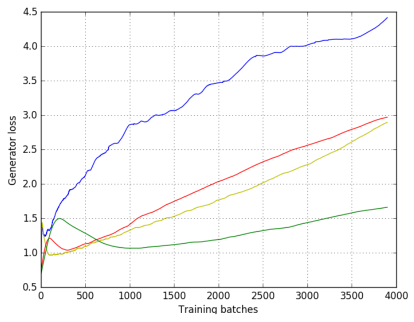
As with our experiments with discrete data, we used deep convolutional GANs (DCGAN, Radford et al., 2015). For the SVHN dataset, we used the same model parameterization as with the quantized CelebA dataset, except the output of the generator was matched to RGB data, with no nonlinearity on the output. For CelebA, we used this same parameterization, except with one additional convolutional layer for both the generator and discriminator. All models were trained with the same Adam hyperparameters with a learning rate of $1e{-}5$. The SVHN model was trained for 100 epochs, while the CelebA model was trained for 20 epochs. We trained the same models with a normal GAN objective for comparison.

5.2. Generation result

For both datasets, the generator trained on the BGAN objective was able to generate diverse samples which were for the most part representative of the original datasets (Figures 6 (a) and 6 (b)). However, qualitatively, these models were no better nor worse than those produced



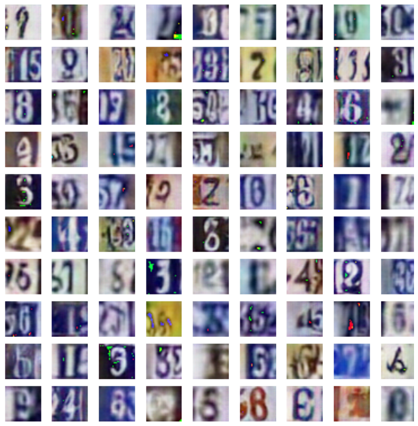
(a) Gumbel-Softmax



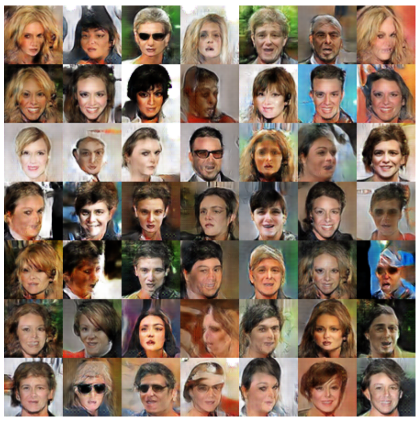
(b) Gumbel-Softmax (Straight-Through)

Figure 5. The diverging behaviour of the generator training when using Gumbel-Softmax technique, both (a) with and (b) without straight-through estimation. The curves are a running average and are sampled from our hyperparameter search:

		b	**y**	**r**	**g**
(a)		adam	rmsprop	adam	rmsprop
	lr	10^{-4}	10^{-4}	10^{-5}	10^{-5}
	r	10^{-3}	10^{-4}	10^{-3}	10^{-3}
	N	500	200	200	300
(b)		adam	rmsprop	adam	rmsprop
	lr	10^{-4}	10^{-4}	10^{-5}	10^{-5}
	r	10^{-3}	10^{-5}	10^{-3}	10^{-3}
	N	300	500	200	300



(a) SVHN



(b) CelebA (Continuous)

Figure 6. Samples generated from the continuous BGANs trained on real-valued datasets; (a) SVHN and (b) CelebA (without quantization). Similarly to the discrete cases, we observe high diversity among generated samples with both of the datasets.

from a generator trained on a normal GAN objective.

6. Conclusion

Reinterpreting the generator objective to match the proposal target distribution reveals a novel learning algorithm for training a generative adversarial network (GANs, Goodfellow et al., 2014). At the core of this algorithm is the boundary seeking objective for a generator that stabilizes learning greatly. This proposed approach of boundary-seeking provides us with a unified framework under which learning algorithms for both discrete and continuous variables are derived. Empirically, we showed the effectiveness of training a GAN with the proposed learning algorithm, to which we refer as a boundary-seeking GAN (BGAN), on both discrete and continuous variables. In due course, we observed that the proposed approach is much more stable compared to the recently proposed re-parametrization technique for discrete variables, called the Gumbel-Softmax technique.

The proposed learning algorithm can be extended for the case of continuous variables by incorporating the idea of importance sampling, which was a core recipe of the version of the proposed algorithm for discrete variables. This can be done by using the normalized weights as a multinomial distribution over a batch of generated samples. This amounts to filtering out poor samples generated from a generator according to a discriminator, and can be used both during training and inference. We leave this for the future.

Acknowledgements

RDH thanks IVADO, MILA, MRN, and UdeM for their support. APJ thanks UWaterloo, Waterloo AI lab and MILA for their support. KC thanks Google (Reward Awards 2015, 2016), NVIDIA (NVAIL) and Facebook for their support. YB thanks CIFAR, NSERC, IBM, Google, Facebook and Microsoft for their support. Finally, we wish to thank the developers of Theano (Al-Rfou et al., 2016), Lasagne `http://lasagne.readthedocs.io`, and Fuel (Van Merriënboer et al., 2015) for their valuable code-base.

References

Al-Rfou, Rami, Alain, Guillaume, Almahairi, Amjad, Angermueller, Christof, Bahdanau, Dzmitry, Ballas, Nicolas, Bastien, Frédéric, Bayer, Justin, Belikov, Anatoly, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bornschein, Jörg and Bengio, Yoshua. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*, 2014.

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

Dayan, Peter, Hinton, Geoffrey E, Neal, Radford M, and Zemel, Richard S. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Dempster, Arthur P, Laird, Nan M, and Rubin, Donald B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

Goodfellow, Ian. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

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

Gu, Shixiang, Levine, Sergey, Sutskever, Ilya, and Mnih, Andriy. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

Gumbel, Emil Julius and Lieblein, Julius. Statistical theory of extreme values and some practical applications: a series of lectures. *US Govt. Print. Office*, 1954.

Hinton, Geoffrey E. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kingma, Diederik and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, DP and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Liu, Ziwei, Luo, Ping, Wang, Xiaogang, and Tang, Xiaoou. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3730–3738, 2015.

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

Mnih, Andriy and Gregor, Karol. Neural variational inference and learning in belief networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1791–1799, 2014.

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Salakhutdinov, Ruslan and Hinton, Geoffrey E. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pp. 448–455, 2009.

Salakhutdinov, Ruslan and Murray, Iain. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pp. 872–879. ACM, 2008.

Saul, Lawrence K, Jaakkola, Tommi, and Jordan, Michael I. Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4 (1):61–76, 1996.

Tolstikhin, Ilya, Gelly, Sylvain, Bousquet, Olivier, Simon-Gabriel, Carl-Johann, and Schölkopf, Bernhard. Adagan: Boosting generative models. *arXiv preprint arXiv:1701.02386*, 2017.

Van Merriënboer, Bart, Bahdanau, Dzmitry, Dumoulin, Vincent, Serdyuk, Dmitriy, Warde-Farley, David, Chorowski, Jan, and Bengio, Yoshua. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, 2015.

Wu, Yuhuai, Burda, Yuri, Salakhutdinov, Ruslan, and Grosse, Roger. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*, 2016.

Yu, Lantao, Zhang, Weinan, Wang, Jun, and Yu, Yong. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.