A Framework of Composite Functional Gradient Methods for Generative Adversarial Models

Rie Johnson and Tong Zhang

Abstract—Generative adversarial networks (GAN) are trained through a minimax game between a generator and a discriminator to generate data that mimics observations. While being widely used, GAN training is known to be empirically unstable. This paper presents a new theory for generative adversarial methods that does not rely on the traditional minimax formulation. Our theory shows that with a strong discriminator, a good generator can be obtained by composite functional gradient learning, so that several distance measures (including the KL divergence and the JS divergence) between the probability distributions of real data and generated data are simultaneously improved after each functional gradient step until converging to zero. This new point of view leads to stable procedures for training generative models. It also gives a new theoretical insight into the original GAN. Empirical results on image generation show the effectiveness of our new method.

Index Terms—Generative adversarial models, functional gradient learning, neural networks, image generation.

1 Introduction

Given examples of real data $x_1^*, \ldots, x_n^* \in \mathbb{R}^k$ from an unknown distribution p_* on $\mathbb{R}^k$ and a random variable Z with a known distribution p_z (e.g., a Gaussian), we are interested in learning transformation G of variable Z so that the distribution of the transformed variable $G(Z)$ becomes close to the distribution of real data. This is the setting considered in generative adversarial networks (GAN) [10], and G is often referred to as a *generator*.

While being widely used, GAN training is known to be difficult due to its instability. This fact has led to numerous studies, e.g., Wasserstein GAN (WGAN) and its extensions [2], [11], [30] to pursue a different minimax objective, regularization to tackle the issue of mode collapse and instability [5], [38], f-GAN [34], unrolled GAN [28], AdaGAN [44], MMD GAN [23], and so forth and references therein.

An important concept introduced by GAN is the idea of *adversarial learner*, denoted here by d, which tries to discriminate real data from generated data. GAN training can be described as the following minimax game between a *discriminator* d and a generator G:

$$\min_G \max_d \left[\mathbb{E}_{x^* \sim p_*} \ln d(x^*) + \mathbb{E}_{z \sim p_z} \ln(1 - d(G(z)))\right]. \quad (1)$$

It was shown in [10] that assuming the optimality of d, (1) is equivalent to minimizing the Jensen-Shanon (JS) divergence between the distribution of real data and that of generated data.

Parameterizing G and d, the GAN training procedure (Algorithm 4 below) seeks to find the solution to the minimax formulation (1) by incrementally using a stochastic gradient method, where a gradient step is taken with respect to the model parameters of d and G. As suggested by [10], however, minimization of $\ln(1 - d(G(z)))$ with respect to G above is often replaced by maximization of $\ln(d(G(z)))$ with respect to G, called *log$d$ trick*, in practice. GAN with the log$d$ trick, though often more effective, can *not directly* be explained by the theory based on the minimax formulation (1).

This paper provides a new theory for generative adversarial methods which does not rely on the traditional minimax formulation. We show that a good generator can be learned where 'goodness' is measured by the divergence between the distributions of real data and generated data, by using *functional gradient learning* greedily, similar to gradient boosting [8]. However, unlike standard gradient boosting, which uses additive models, this paper considers functional compositions of the following form

$$G_t(Z) = G_{t-1}(Z) + \eta_t g_t(G_{t-1}(Z)), \quad (t = 1, \ldots, T) \quad (2)$$

to obtain $G(Z) = G_T(Z)$. Here η_t is a small step size, and each g_t is a function to be estimated from data. An initial generator $G_0(Z) \in \mathbb{R}^k$ is assumed to be given. We learn from data g_t greedily from $t = 1$ to $t = T$ so that improvement in terms of the divergence between the two distributions is guaranteed.

The first part of the theory considers the limit where the step size η_t approaches 0 and thus considers a generator that continuously evolves in time. We show that the algorithm suggested by our analysis *simultaneously* minimizes multiple distance measures such as the KL divergence and the JS divergence, and more generally, f-divergences with certain properties. The second part relaxes the condition on the step size and thus considers a generator that evolves in discrete steps.

Our theory leads to a new algorithm for learning generative adversarial models that is stable and effective. It also provides a new theoretical insight into the original GAN (either with or without the log$d$ trick). The experiments show the effectiveness of our new method on image generation in comparison with GAN variants.

A shorter version of the paper was presented at the 35-th International Conference on Machine Learning (ICML 2018) [19].

- *Rie Johnson is with RJ Research Consulting, Tarrytown, NY, USA.*
- *Tong Zhang is with HKUST, Hong Kong.*

1.1 Preliminaries

Notation: Throughout the paper, we use x to denote data in $\mathbb{R}^k$. The probability density function of real data is denoted by p_*. We use $\|\cdot\|$ to denote the vector 2-norm and the matrix spectral norm. Given a scalar function $h(x)$, we use $\nabla h(x)$ to denote the gradient with respect to x. Note that we have $\nabla h(x) \in \mathbb{R}^k$ as $x \in \mathbb{R}^k$. Given a vector function $g(x)$, we use $\nabla g(x)$ to denote its Jacobi matrix. Convergences are all pointwise.

Logistic regression: Our analysis will use the following known facts on logistic regression. Let p_* and p be the probability densities of real data and generated data, respectively, and define $\mathcal{D}(x)$ by

$$\mathcal{D}(x) := \ln \frac{p_*(x)}{p(x)} \ . \tag{3}$$

Then $\mathcal{D}$ is the analytical solution to a logistic regression problem for discriminating real data and generated data; i.e.,
$\mathcal{D}=\arg\min_{D'} \left[\mathbf{E}_{x\sim p_*} \ln(1+e^{-D'(x)})+\mathbf{E}_{x\sim p} \ln(1+e^{D'(x)})\right]$.
In practice, one can only approximate the theoretical optimum solution above by choosing from some class of functions $\mathcal{C}$ using finite amounts of sample S_* and S to solve:

$$\arg\min_{D'\in\mathcal{C}} \left[\sum_{x\in S_*} \frac{\ln(1+e^{-D'(x)})}{|S_*|}+\sum_{x\in S} \frac{\ln(1+e^{D'(x)})}{|S|}\right] \ . \tag{4}$$

Statistical consistency of such approximation has been formally studied (e.g., [49]), but intuitively, approximation improves with larger sample and more universal $\mathcal{C}$.

2 Continuous Theory

Our goal is to transform a random variable $Z \in \mathbb{R}^k$ with a known distribution by (2):

$$G_t(Z) = G_{t-1}(Z) + \eta_t g_t(G_{t-1}(Z)), \quad (t = 1, \ldots, T) \ ,$$

so that the probability density of the transformed variable $G_T(Z)$ is close to p_*, the density of real data. The sequence of transformation in (2) takes discrete steps from time $t-1$ to time t, but in this section, let us instead take a small time step δ and set $\eta_t = \delta$ so that we have

$$G_{t+\delta}(Z) = G_t(Z) + \delta g_t(G_t(Z)) \ . \tag{5}$$

By letting $\delta \to 0$, we have a generator that evolves *continuously* in time t that satisfies an ordinary differential equation

$$\frac{d(G_t(Z))}{dt} = g_t(G_t(Z)) \ . \tag{6}$$

In this section, we study this continuously evolving generator. In practice, we envision a learning process that starts with a given initial generator G_0 and proceeds with discretization by alternating (5) and $t \leftarrow t + \delta$ with a small δ. Thus, having step-size $\delta \to 0$ is an idealization that simplifies the analysis and therefore helps to understand the problem. We will relax this condition in the next section.

2.1 Analysis

The goal is to learn $g_t : \mathbb{R}^k \to \mathbb{R}^k$ from data so that the probability density of $G_t(Z)$, which continuously evolves by (6), becomes close to the density p_* of real data as t continuously increases. To measure the 'closeness', we use a distance measure in the form of:

$$L(p) = \int \ell(p_*(x), p(x))dx, \tag{7}$$

where $\ell : R^2 \to R$ is a pre-defined function so that L satisfies $L(p) = 0$ if and only if $p = p_*$ and $L(p) \geq 0$ for any probability density function p.

From the following theorem, we will derive the choice of $g_t(\cdot)$ that guarantees that transformation (5) with $\delta \to 0$ can always reduce/improve $L(\cdot)$.

Theorem 2.1. Using the definitions above, let p_t be the probability density of random variable $G_t(Z)$. Let $\ell'_2(\rho_*, \rho) = \partial\ell(\rho_*, \rho)/\partial\rho$. Then we have

$$\frac{dL(p_t)}{dt} = \int p_t(x)\nabla_x \ell'_2(p_*(x), p_t(x)) \cdot g_t(x)dx. \tag{8}$$

The proof is given in Appendix A.

The theorem implies that for $\frac{dL(p_t)}{dt}$ to be negative so that the distance L decreases/improves, we should choose $g_t(x)$ to be:

$$g_t(x) = -s_t(x)\phi_0(\nabla_x \ell'_2(p_*(x), p_t(x))), \tag{9}$$

where $s_t(x) > 0$ is an arbitrary scaling factor. $\phi_0(u)$ is a vector function such that $\phi(u) = u \cdot \phi_0(u) \geq 0$ and that $\phi(u) = 0$ if and only if $u = 0$, e.g., ($\phi_0(u) = u$, $\phi(u) = \|u\|_2^2$) or ($\phi_0(u) = \text{sign}(u)$, $\phi(u) = \|u\|_1$). With this choice of $g_t(x)$, we obtain

$$\frac{dL(p_t)}{dt} = -\int s_t(x)p_t(x)\phi(\nabla_x \ell'_2(p_*(x), p_t(x)))\ dx \ \leq 0 \ ,$$

that is, the distance L is guaranteed to improve unless the equality holds. Moreover, this implies that we have $\lim_{t\to\infty} \int s_t(x)p_t(x)\phi(\nabla_x \ell'_2(p_*(x), p_t(x)))\ dx = 0$. (Otherwise, $L(p_t)$ would keep going down and become negative as t increases, but $L(p_t) \geq 0$ by definition.) With a continuity condition (see Appendix B.5), this further implies

$$\lim_{t\to\infty} p_t(x)\nabla_x \ell'_2(p_*(x), p_t(x)) = 0. \tag{10}$$

We show below the cases where (10) can further lead to $\lim_{t\to\infty} p_t(x) = p_*(x)$, i.e., the distribution of generated data converges to that of real data provided that $g_t(x)$ is chosen as in (9).

2.2 f-divergences

Let us consider a case where the distance measure $L(\cdot)$ is an f-divergence. With a convex function $f : \mathbb{R}^+ \to \mathbb{R}$ such that $f(1) = 0$ and that f is strictly convex at 1, $L(p_t)$ defined by

$$L(p_t) = \int p_*(x) f\left(r_t(x)\right) dx \quad \text{where } r_t(x) = \frac{p_t(x)}{p_*(x)} \tag{11}$$

is called f-divergence[1]. Here we focus on a special case where f is twice differentiable and strongly convex so that the second order derivative of f, denoted here by f'', is

1. The conventions are: $0f(0/0) = 0$, $f(0) = \lim_{\gamma\to 0} f(\gamma)$, $0f(a/0) = \lim_{\rho\to 0} \rho f(a/\rho) = a \lim_{\beta\to\infty}(f(\beta)/\beta)$; see e.g., [6].

	$f(\gamma)$	$f'(\gamma)$	$f''(\gamma)$	$\gamma f''(\gamma)$
KL	$-\ln(\gamma)$	$-1/\gamma$	$1/\gamma^2$	$1/\gamma$
rev KL	$\gamma\ln(\gamma)$	$\ln\gamma+1$	$\frac{1}{\gamma}$	1
2JS	$\ln\frac{2}{1+\gamma}+\gamma\ln\frac{2\gamma}{1+\gamma}$	$\ln\frac{2\gamma}{1+\gamma}$	$\frac{1}{\gamma(1+\gamma)}$	$\frac{1}{1+\gamma}$
H^2	$(\sqrt{\gamma}-1)^2$	$1-\frac{1}{\sqrt{\gamma}}$	$\frac{1}{2\gamma\sqrt{\gamma}}$	$\frac{1}{2\sqrt{\gamma}}$

TABLE 1: Examples of f-divergences: KL divergence, reverse KL divergence, JS divergence×2, and squared Hellinger distance.

always positive, which includes commonly used measures in Table 1. (11) implies $\ell(\rho_*,\rho)=\rho_* f(\rho/\rho_*)$, and therefore

$$\nabla_x \ell_2'(p_*(x),p_t(x)) = f''(r_t(x))\nabla r_t(x)\ . \tag{12}$$

Using this and letting $\phi_0(u)=u$, (9) and (10) can be rewritten respectively as

$$g_t(x) = -s_t(x)f''(r_t(x))\nabla r_t(x)\ , \tag{13}$$
$$\lim_{t\to\infty} p_t(x)f''(r_t(x))\nabla r_t(x) = 0\ . \tag{14}$$

As $f''(r_t(x))>0$ by assumption, (14) implies that if p_t has full support on $\mathbb{R}^k$ as $t\to\infty$, then the pdf ratio $r_t(x)$ goes to a constant for all x, which implies that $p_t\to p_*$, as $t\to\infty$. Furthermore, (14) can be rewritten as

$$\lim_{t\to\infty} p_*(x)r_t(x)f''(r_t(x))\nabla r_t(x) = 0\ .$$

For some divergences, this can be further rewritten as follows (see Table 1 for derivation):

KL divergence: $\lim_{t\to\infty} p_*(x)\nabla_x \ln(r_t(x)) = 0$

reverse KL divergence: $\lim_{t\to\infty} p_*(x)\nabla_x r_t(x) = 0$

JS divergence: $\lim_{t\to\infty} p_*(x)\frac{1}{1+r_t(x)}\nabla r_t(x) = 0$

squared Hellinger distance: $\lim_{t\to\infty} p_*(x)\nabla_x\sqrt{r_t(x)} = 0$

Thus, with these divergences, if p_* has full support on $\mathbb{R}^k$, then the pdf ratio $r_t(x)$ goes to a constant for all x, which implies that $p_t\to p_*$ as $t\to\infty$.

The convergence does not hold if p_* (or p_t as $t\to\infty$) does not have full support. Moreover, for x such that $p_*(x)=0$ and $p_t(x)>0$, $r(x)$ goes to infinity, and $r(x)=0$ when $p_t(x)=0$ and $p_*(x)>0$, which pushes $f''(\gamma)$ to infinity in some cases (see Table 1). Therefore, if $p_*(x)$ or $p_t(x)$ was zero in a substantial part of $\mathbb{R}^k$, then f-divergences might not be suitable for the purpose. In that case, it would be rather sensible to either *abandon* f-divergences or work with *approximations* – by approximating such distributions with more manageable distributions that have full support. Essentially, the former argues for WGAN [2], and we take the latter approach. The practicality of the assumptions will be further discussed later in Section 4 after the whole picture is laid out.

2.2.1 Algorithms

To derive an algorithm from the results above, note that the suggested $g_t(x)$ in (13) cannot be obtained from data in practice as it depends on unknown densities p_* and p_t, but that it can be approximated using the discriminator. Let $D(x)$ be the solution to the empirical logistic regression problem (4) for discriminating real data from generated data at time t; for simplicity, we omit the subscript t from all variables in this section. Assuming that $D(x)\approx\mathcal{D}(x)=\ln\frac{p_*(x)}{p(x)}$ (3), let us define $\tilde{r}(x)$ so that

$$\tilde{r}(x)=\exp(-D(x))\approx\frac{p(x)}{p_*(x)}=r(x)$$

and replace $r(x)$ in (13) with $\tilde{r}(x)$. Then, as $\nabla\tilde{r}(x)=-\tilde{r}(x)\nabla D(x)$, we obtain

$$g(x)=s(x)v(x)\nabla D(x),$$

where $v(x)=\tilde{r}(x)f''(\tilde{r}(x))$. Since $v(x)>0$ (because $f''(\cdot)>0$ by assumption), $v(x)$ can be absorbed into the data-dependent arbitrary scaling factor $s(x)$. Thus, we obtain an algorithm that repeats the following:

- Update discriminator D to minimize logistic loss (4).
- Update generator G by $G(z)\leftarrow G(z)+\delta g(G(z))$ with $g(x)=s(x)\nabla D(x)$.

This is essentially identical to the algorithm that we will derive from our discrete analysis in the next section (Algorithm 1), and so we will look into it in more detail there. Nevertheless, it is worth noting that the obtained algorithm can be regarded as *simultaneously optimizing several distance measures* (all the f-divergences with strongly convex f such as the KL divergence, JS divergence, and the Hellinger distance). That is, even if we simply fix $s(x)$ to a constant (e.g., $s(x)=1$) instead of customizing it for each measure, the resulting algorithm optimizes several measures.

[34] extended GAN, which was shown to be associated with the JS divergence [10], to f-GAN for various f-divergences. One difference here is that our analysis indicates that a single algorithm (Algorithm 1) using a single optimization objective (i.e., minimization of the logistic loss) simultaneously optimizes multiple divergences. By contrast, the f-GAN study proposed different optimization objectives (leading to different computations) for minimizing different divergences.

Least squares variant: The pdf ratio can also be estimated by other optimization objectives such as least squares. Use of the least squares estimate also leads to the generator update procedure above, thus leading to the least squares variant that replaces the discriminator update using logistic regression with the update using the least squares objective.

3 Discrete Theory

Now we turn to our discrete theory on the generator $G_T(Z)$ obtained by taking discrete steps (2):

$$G_t(Z)=G_{t-1}(Z)+\eta_t g_t(G_{t-1}(Z)),\quad (t=1,\dots,T)\ .$$

Our analysis in this section will include the effect of the step size η_t and the deviation of the discriminator from the optimum. Before we start, let us first consider how the results of our continuous analysis apply towards this end.

Corollary 3.1. Let $L(p)$ be the f-divergence $L(p)=\int p_*(x)f(r(x))dx$ where $r(x)=\frac{p(x)}{p_*(x)}$ with strongly convex f as in Section 2.2. Let p and p' be the densities

Algorithm 1 CFG: Composite Functional Gradient Learning of GAN

Input: real data $x_1^*, \ldots, x_n^*$, initial generator $G_0(z)$ with generated data $\{G_0(z_1), \ldots, G_0(z_m)\}$. Meta-parameter: T.

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $D_t(x) \leftarrow \arg\min_D \left[\frac{1}{n}\sum_{i=1}^n \ln(1+\exp(-D(x_i^*))) + \frac{1}{m}\sum_{i=1}^m \ln(1+\exp(D(G_{t-1}(z_i))))\right]$
3: $g_t(x) \leftarrow s_t(x)\nabla D_t(x)$ ($s_t(x)$ is for scaling, e.g., most simply $s_t(x) = 1$)
4: $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for some $\eta_t > 0$.
5: **end for**
6: **return** generator $G_T(z)$

of random variables X and X', respectively, such that $X' = X + \eta g(X)$. Let $\mathcal{D}(x) = \ln \frac{p_*(x)}{p(x)}$. Then we have:

$$L(p') = L(p) - \eta \int p_*(x) v(x) \nabla \mathcal{D}(x) \cdot g(x) + j(\eta)$$

where $v(x) = (r(x))^2 f''(r(x))$ and $j(\eta) = o(\eta)$ as $\eta \to 0$.

Proof

$$\frac{dL(p)}{dt} = \int p(x) f''(r(x)) \nabla r(x) \cdot g(x)\, dx \tag{15}$$

$$= -\int p_*(x) v(x) \nabla \mathcal{D}(x) \cdot g(x)\, dx \tag{16}$$

(15) is from Theorem 2.1 and (12), and (16) uses $\nabla r(x) = -r(x)\nabla \mathcal{D}(x)$ and $p(x) = r(x)p_*(x)$. The desired result is obtained by using (16) in the Taylor series of $L(p')$. ■

We have $v(x) \geq 0$ as f is strongly convex, and so if $j(\eta)$ is small, ignoring the equality, this result would suggest that the reduction of the f-divergence is guaranteed with $g(x) = s(x)\mathcal{D}(x)$ with any $s(x) > 0$. Although this is essentially true under appropriate conditions, making a formal statement has some complication. Since we no longer assume $\eta \to 0$, we need to bound $j(\eta)$ as η increases. The regularity conditions for this purpose would be rather complex in the general f-divergence case. For this reason, we choose to prove a rigorous statement only for the KL divergence, which simplifies presentation. More details regarding this choice are given in Appendix B.4.

We start with stating the definitions and assumptions.

3.1 Definitions and assumptions

3.1.1 Definitions

As before, let p_* and p be the probability densities of real data and generated data, respectively. We continue to use $\mathcal{D}(x) = \ln \frac{p_*(x)}{p(x)}$ from Section 1.1 to indicate the theoretical solution to the logistic regression problem, and let discriminator D be the solution to the empirical logistic regression problem (4). We let L be the KL divergence:

$$L(p) = \int p_*(x) \ln \frac{p_*(x)}{p(x)} dx.$$

3.1.2 Assumptions

Assumption 3.1 (Boundedness of the pdf ratio). $|\mathcal{D}(x)|$ is bounded so that there exists a positive number $B < \infty$ such that $|\mathcal{D}(x)| \leq B$.

As $\mathcal{D}(x) = \ln \frac{p_*(x)}{p(x)}$, this assumption implies that both $p_*(x)$ and $p(x)$ are nonzero everywhere. (However, also note that we have in Appendix B an analysis with slightly more relaxed assumptions that allow p without full support.)

Assumption 3.2 (ϵ-approximation condition on the discriminator). Discriminator D satisfies the following ϵ-approximation condition for some ϵ such that $0 \leq \epsilon < \infty$:

$$\int q_*(x) \left| D(x) - \mathcal{D}(x) \right| dx \leq \epsilon$$

$$q_*(x) = p_*(x) \max(1, \|\nabla \ln p_*(x)\|)\ .$$

The *optimal* discriminator has been often assumed and we slightly relax it by quantifying the deviation from the optimal $\mathcal{D}(x)$. Note that 'smallness' of ϵ (corresponding to a *strong* discriminator) is not required for proving our theorem below, but it is required for deriving the convergence.

Nonzero smooth light-tailed p_*: We assume that p_*, the density of real data, is nonzero and smooth with light tails; we use a constant $h_0 > 0$ that depends on the shape of p_*. Common exponential distributions such as Gaussians and Gaussian mixtures all satisfy the assumption. The precise statements are given in Appendix B.2.2.

Remark: To put it intuitively, we assume that any data point in $\mathbb{R}^k$ has some possibility (which changes smoothly) of being real as well as some possibility of being generated, and that $D(x)$ indicates which is more likely.

Our assumptions are traditional and so make a sharp contrast with the low dimensional manifold assumption of [1], which led to WGAN [2]. We will discuss the practicality of our assumptions later in Section 4 in the context of developing practical algorithms for image generation.

3.2 Analysis

The goal is to approximate the true density p_* on $\mathbb{R}^k$ through $G_t(Z) = G_{t-1}(Z) + \eta_t g_t(G_{t-1}(Z)), (t = 1, \ldots, T)$. Our analysis here focuses on one step at time t, namely, random variable transformation of $G_{t-1}(Z)$ to $G_t(Z)$. To simplify notation, we assume that we are given a random variable X with a probability density p on $\mathbb{R}^k$. We are interested in finding a function $g : \mathbb{R}^k \to \mathbb{R}^k$, so that the transformed variable $X' = X + \eta g(X)$ for small $\eta > 0$ has a density closer to p_*, while closeness is measured by the KL divergence $L(\cdot)$.

The consequence of the following theorem shows that with an appropriately chosen $g(\cdot)$, the transformation $X \to X + \eta g(X)$ can always reduce the KL divergence $L(\cdot)$, and so transformation $X + \eta g(X)$ is an improvement from X.

Theorem 3.1. Under the assumptions in Section 3.1.2, let $g : \mathbb{R}^k \to \mathbb{R}^k$ be a continuously differentiable transformation such that $\|g(\cdot)\| \leq a$ and $\|\nabla g(\cdot)\| \leq b$. Let p and p' be the probability densities of random variables X and X', respectively, such that $X' = X + \eta g(X)$ where $0 < \eta <$

$\min(1/b, h_0/a)$. Then there exists a positive constant c such that:

$$L(p') \leq L(p) - \eta \int p_*(x)\, \nabla D(x) \cdot g(x)\, dx + c\eta^2 + c\eta\epsilon.$$

The proof is given in Appendix B.

If ϵ is large, i.e., if the discriminator deviates from the optimal discriminator by a lot, then $c\eta\epsilon$ will dominate the right-hand side of the inequality and so whether or not $L(p') < L(p)$ is unknown, which means that the generator may degrade. This is not surprising. ϵ can become large, for example, when the sample is small or when the function class from which the discriminator is chosen is poor.

To examine the dependency on η, let us assume that ϵ is as small as η so that $c\eta\epsilon$ does not dominate. Then, the theorem implies that we should choose $g(x)$ to be

$$g(x) = s(x)\nabla D(x)$$

where, as before, $s(x) > 0$ is an arbitrary scaling factor. With this choice of g, and letting $\epsilon = \eta$, we have

$$L(p') \leq L(p) - \eta \int p_*(x)s(x)\, \|\nabla D(x)\|_2^2\, dx + 2c\eta^2. \quad (17)$$

This means that L will be reduced for a sufficiently small η unless the following functional gradient vanishes

$$\int p_*(x)s(x)\, \|\nabla D(x)\|_2^2\, dx.$$

The vanishing condition implies that $D(x)$ is a constant when p_* has full support on $\mathbb{R}^k$. In this case, the discriminator is unable to distinguish the real data from the generated data. Thus, it is implied that letting $g(x) = s(x)\nabla D(x)$ makes the probability density of generated data closer to that of real data until the discriminator becomes unable to distinguish the real data and generated data.

We note that taking $g(x) = s(x)\nabla D(x)$ is analogous to taking a gradient descent step of $L(p)$ in the function space, so that a step is taken to modify the function instead of the model parameters. Therefore, Theorem 3.1 presents a functional gradient view of variable transformation that *can always improve the quality of the generator* — when the quality is measured by the KL divergence between the real data and the generated data.

If we repeat the process described above, Algorithm 1 is obtained. We call it *composite functional gradient learning of GAN (CFG-GAN)*, or simply *CFG*. CFG forms g_t using the functional gradient, as suggested by Theorem 3.1. Also note that the continuous theory with general f-divergences leads to the CFG algorithm too, as shown earlier.

By cascading (17) from $t = 1$ to $t = T$, we obtain:

$$L(p_T) \leq L(p_0) - \eta \sum_{t=1}^{T} \int p_*(x)s_t(x)\|\nabla D_t(x)\|^2 dx + 2Tc\eta^2$$

where p_T is the probability density of generated data after updating the generator T times, and p_0 is the density of $G_0(Z)$. With $\eta = 1/\sqrt{T}$, this leads to

$$\frac{1}{T}\sum_{t=1}^{T} \int p_*(x)s_t(x)\|\nabla D_t(x)\|^2 dx \quad (18)$$
$$\leq (L(p_0) - L(p_T))/(T\eta) + 2c\eta$$
$$= (L(p_0) - L(p_T))/\sqrt{T} + 2c/\sqrt{T} = O(1/\sqrt{T}) \quad (19)$$

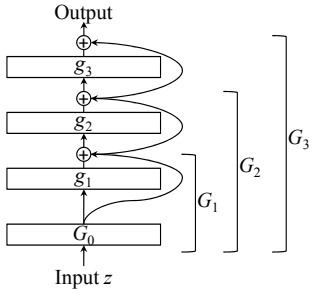


Fig. 1: Generator network automatically created by CFG or ICFG with $T = 3$. '$\oplus$' indicates addition.

This implies $\lim_{t\to\infty} \int p_*(x)s_t(x)\|\nabla D_t(x)\|^2 dx = 0$. (Otherwise, (18) would be no smaller than some positive constant, but (19) goes to 0 as T goes to infinity.) With a continuity condition (see Appendix B.5), this further implies that as t increases, $\nabla D_t(x) \to 0$ and so $D_t(x)$ approaches a constant if p_* has full support on $\mathbb{R}^k$. That is, in the limit where $T \to \infty$ and $\eta = \epsilon = 1/\sqrt{T}$, the discriminator $D_T(x)$ is unable to distinguish the real data from the generated data, and the algorithm converges.

4 Algorithms

Starting from the CFG algorithm above, we empirically explored algorithms using image generation as an example task. This section describes variants of CFG that were found to be efficient and effective and discusses their relation to the original GAN.

Notation: The algorithms introduced below assume that parametric model definitions (e.g., neural network architectures) are given for use as a discriminator and others. We write θ_D for the model parameters of D (or θ_G for G).

4.1 Empirical behavior of CFG on image generation

In this section, we first describe empirical issues of CFG when applied to image generation. We then provide a theoretical interpretation of the issues and a solution to them.

CFG (Algorithm 1) optimizes a discriminator to convergence with a fixed generator in every iteration. This causes two empirical issues when applied to image generation. First, its computation is prohibitively expensive. Second, it is apparently harmful to excessively update the discriminator with a fixed generator; let us broadly call it *overtraining*. One example of overtraining we observed is as follows. We start with G_0 that generates high-quality images as a result of being trained elsewhere. As we keep updating discriminator D with the generator fixed to G_0, the discriminator starts assigning larger and larger values to real data and smaller values to generated data. But also it starts assigning large values (even larger than to real data) to garbled images that look like *neither real data nor generated data*. This pushes the generator in their directions in generator update as $\nabla D(x)$ points to them, and degrades the generator.

Our theory above assumes that essentially, any data point in $\mathbb{R}^k$ has some possibility of being real as well as some possibility of being generated, and that $D(x)(\approx \mathcal{D}(x))$ indicates which is more likely. If it is reasonable to expect

that the pdf ratio $\mathcal{D}(x) = \ln \frac{p_*(x)}{p(x)} \approx 0$ for the images that look like neither real nor generated, then having a large $D(x)$ for such images means that the discriminator D deviates a lot from the optimal discriminator $\mathcal{D}$, i.e., ϵ in the ϵ-approximation condition is large. Thus, according to our theory, the degradation of the generator in the case above results from large ϵ.

Fortunately, we found that *early stopping* prevents overtraining. These problematic images belong to the low-density region where both $p_*(x)$ and $p(x)$ are small. From the viewpoint of classification learning, this is D's failure of generalizing to unseen (and rarely occurring) data. If we regard it as a peculiar form of overfitting, it makes sense that early stopping, a common technique for preventing overfitting, can counteract it. Early stopping also solves the issue of expensive computation.

The harm of discriminator overtraining has been noted also in the training of GANs. In the analysis of GAN training by [1], which led to WGAN [2], it was shown that essentially, the optimal (as well as near-optimal) discriminator $d_*(x) \in [0, 1]$ is harmful[2] if real data and generated data are contained in respective low dimensional manifolds (and so their distributions do not admit a density). With this assumption, the optimal d_* becomes a *perfect classifier* that achieves $d_*(x) = 1$ on the support of the real data distribution and $d_*(x) = 0$ on that of generated data. To see the relation between [1] and our analysis, note that as we use a logistic model $d(x) = \frac{1}{1+\exp(-D(x))}$, we have: $d(x) = 1$ iff $D(x) = \infty$, $d(x) = 0$ iff $D(x) = -\infty$. This means that the optimal and yet harmful discriminator d_* of [1] is the kind of discriminator we also wish to avoid in modeling data, as expressed in our assumption $|D(x)| < \infty$ (implied by $|\mathcal{D}(x)| < \infty$ and $D(x) \approx \mathcal{D}(x)$). We found that such an extreme discriminator can also be avoided by early stopping.

Our assumptions do not hold if the supports of data distributions are indeed contained in respective low-dim manifolds. In that case, our approach should be viewed as *approximating* such hard and spiky distributions without a density by soft and smooth distributions with nonzero densities, which are much *easier to work with*. A discriminator can be encouraged to behave as assumed by, e.g., adding a small Guassian noise to every observed data point [1]. Our empirical finding is, however, that noise addition is not needed for our algorithms; stable training with good performances can be achieved when we combine, as in the algorithms presented below, early stopping of discriminator training (like GANs) with functional gradient learning in generator update (unlike GANs). The latter ensures improvement of the generator, and we will later revisit this point.

4.2 ICFG: Incremental CFG

We therefore considered an incremental variant of CFG, *incremental CFG of GAN (ICFG)*, shown in Algorithm 2. ICFG incrementally updates a discriminator little by little interleaved with the generator updates, so that the generator can keep providing new and more challenging examples to prevent the discriminator from going into an undesirable state. That is, similar to GANs, ICFG alternates between U updates of the discriminator using U minibatches (where U is a meta-parameter) and one update of the generator.

Algorithm 2 ICFG: Incremental CFG

Input: a set of training examples S_*, prior p_z, initial generator G_0, discriminator D. Meta-parameters: T, mini-batch size b, discriminator update frequency U.

for $t = 1, 2, \ldots, T$ **do**
  for U steps **do**
    Sample $x_1^*, \ldots, x_b^*$ from S_*.
    Sample $z_1, \ldots, z_b$ according to p_z.
    Update D by descending the stochastic gradient: $\nabla_{\theta_D} \frac{1}{b} \sum_{i=1}^{b} \left[\ln(1+e^{-D(x_i^*)}) + \ln(1+e^{D(G_{t-1}(z_i))})\right]$
  end for
  $g_t(x) \leftarrow s_t(x)\nabla D(x)$ (e.g., $s_t(x) = 1$)
  $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for $\eta_t > 0$.
end for
return generator G_T (and the updated D if so desired).

The switch to incremental update was for improving ϵ (and efficiency); in that sense, our theory could still apply here. However, very small U (e.g., $U = 1, 5, 10$) works well as shown later. In this case, it is unlikely that ϵ (the deviation of D from the optimum) is always small, especially, at the beginning of training, and so an additional analysis is required to explain the situation. Informally, imagine smooth *surrogate distributions*, which are close to flat at the beginning and gradually approach to the true distributions as more and more data points are observed as training proceeds. We conjecture that when such surrogate distributions are considered in place of the true distributions, ϵ could be small, and so improvement of the generator could be explained.

ICFG shares a nice property with CFG that there is no need to design a complex generator model. The generator model is automatically and implicitly derived from the discriminator, and it dynamically *grows* as training proceeds. Figure 1 illustrates a generator created by ICFG with $T = 3$ (i.e., 3 iterations). As is clear from the illustration, the generator forms a *residual net* [12], where each building block g_t is automatically derived from the discriminator at time t.

A shortcoming of ICFG, however, is that the implicit generator network can become very large. At time t, the size of the generator network is in $O(t)$, and therefore, the cost for computing $G_t(z)$ starting from scratch is in $O(t)$. This means that the computational cost for performing T iterations of training could be in $O(\sum_{t=1}^{T} t) = O(T^2)$. Therefore, on a task that requires many iterations (i.e., a large T), training would become quite expensive. We found that image generation requires a relatively large T, e.g., $T > 1000$, and that T in this order is problematic, causing slow training, a large model that takes up a lot of space, and slow image generation.

4.3 xICFG: Approximate incremental CFG

As a solution to the issue of large generators, we propose *Approximate ICFG* (*xICFG*, Algorithm 3). xICFG periodically compresses the generator obtained by ICFG, by training an *approximator* of a fixed size that approximates the behavior of the generator obtained by ICFG. That is, given a definition

2. We write "d_*" for "D_*" of the *perfect discrimination theorems* in [1].

Algorithm 3 xICFG: Approximate ICFG

Input: a set of training examples S_*, prior p_z, approximator $\widetilde{G}$ at its initial state, discriminator D.
Meta-parameters: (T, b, U) for ICFG, and N.

1: **loop**
2: G,$D \leftarrow$ output of ICFG using $S_*, p_z, \widetilde{G}, D$ as input.
3: **if** exit criteria are met **then return** generator G **fi**
4: Sample $z_1, z_2, \ldots, z_N$ according to p_z.
5: Update $\widetilde{G}$ to minimize $\sum_i^N \frac{1}{2}\|\widetilde{G}(z_i) - G(z_i)\|^2$
6: **end loop**

of an approximator $\widetilde{G}$ and its initial state, xICFG repeatedly alternates the following.

- Using the approximator $\widetilde{G}$ as the initial generator, perform T iterations of ICFG to obtain generator G.
- Update the approximator $\widetilde{G}$ to achieve $\widetilde{G}(z) \approx G(z)$.

The generator size is again in $O(T)$, but unlike ICFG, which requires T to be large (e.g., $T > 1000$) for complex tasks such as image generation, T for xICFG can be small (e.g., $T = 15$), and so xICFG is efficient. Optionally, one can perform input pooling for reducing computation, as was done in our short paper [19], but in this work we focus on xICFG without input pooling, which is simpler.

A small N (the number of data points used for approximator update) and a small T would reduce the runtime of one iteration of xICFG, but they would increase the number of required iterations, as they reduce the amount of the improvement achieved by one iteration of xICFG, and so a trade-off should be found empirically. In particular, as approximation tends to cause some degradation, it is important to set T and N to sufficiently large values so that the amount of the generator improvement exceeds the amount of degradation caused by approximation. In our experiments, however, tuning of meta-parameters turned out to be relatively easy; essentially one set of meta-parameters achieved stable training in all the tested settings across datasets and network architectures, as shown later.

Our theory does not apply to xICFG as a whole, but it applies to ICFG performed in each iteration of xICFG to the extent discussed above. Compression of the generator (G grown by ICFG) by approximating its behavior with a smaller network (approximator $\widetilde{G}$) is related to *distillation* [14]. xICFG's performance partly depends on the representation power of $\widetilde{G}$, which can become a bottleneck, and we will later report empirical results obtained by both high-capacity $\widetilde{G}$ and low-capacity $\widetilde{G}$.

4.4 Relation to GAN

It is known that training of GANs can be hard due to its instability. In this section, we show that GAN training (Algorithm 4) can be regarded as a coarse approximation of ICFG, and in particular, it is closely related to a special case of xICFG that sets the meta-parameters to extreme values. This viewpoint leads to some insight into GAN's instability.

We start with the fact that GANs with the logistic model (and so $d(x) = \frac{1}{1+\exp(-D(x))}$) and ICFG share the discriminator update procedure as both minimize the logistic loss. This fact becomes more obvious when we plug

Algorithm 4 GAN training [10]

Input: S_*, p_z, discriminator d, G. Meta-parameters b, U.

1: **repeat**
2: **for** U steps **do**
3: Sample $x_1^*, \ldots, x_b^*$ from S_*.
4: Sample $z_1, \ldots, z_b$ according to p_z.
5: Update d by ascending the stochastic gradient:
$\nabla_{\theta_d} \frac{1}{b} \sum_{i=1}^b [\ln d(x_i^*) + \ln(1 - d(G(z_i)))]$
6: **end for**
7: Sample $z_1, \ldots, z_b$ according to p_z.
8: Update G by descending the stochastic gradient:
$\nabla_{\theta_G} \frac{1}{b} \sum_{i=1}^b \ln(1 - d(G(z_i)))$
9: **until** exit criteria are met
10: **return** generator G

$d(x) = \frac{1}{1+\exp(-D(x))}$ into the GAN discriminator update step, Line 5 of Algorithm 4.

Next, we show that *the generator update of GAN is equivalent to coarsely approximating a generator produced by ICFG*; in more detail, it is equivalent to taking one step of gradient descent in order to approximate the generator produced by ICFG with $T = 1$. To see this, first note that GAN's generator update (Line 8 of Algorithm 4) requires the gradient $\nabla_{\theta_G} \ln(1 - d(G(z)))$. Using $d(x) = \frac{1}{1+\exp(-D(x))}$ again, and writing $[v]_i$ for the i-th component of vector v, the k-th component of this gradient can be written in terms of D as:

$$\begin{aligned}[\nabla_{\theta_G} \ln(1 - d(G(z)))]_k &= \left[\nabla_{\theta_G} \ln \tfrac{\exp(-D(G(z)))}{1+\exp(-D(G(z)))}\right]_k \\ &= -s_0(G(z)) \textstyle\sum_j [\nabla D(G(z))]_j \frac{\partial [G(z)]_j}{\partial [\theta_G]_k} \end{aligned} \quad (20)$$

where $s_0(x) = \frac{1}{1+\exp(-D(x))}$, a scalar resulting from differentiating $f(y) = -\ln \frac{\exp(-y)}{1+\exp(-y)}$ at $y = D(x)$. Now suppose that we apply ICFG with $T = 1$ to a generator G to obtain a new generator G':

$$G'(z) = G(z) + \eta g(G(z)) = G(z) + \eta s(G(z)) \nabla D(G(z)) \,,$$

and then we update G to approximate G' so that

$$\frac{1}{2} \|G'(z) - G(z)\|^2$$

is minimized as in Line 5 of xICFG. To take one step of gradient descent for this approximation, we need the gradient of the square error above with respect to θ_G. It is easy to verify that the k-th component of this gradient is:

$$\begin{aligned}\left[\nabla_{\theta_G} \tfrac{1}{2} \|G'(z) - G(z)\|^2\right]_k &= -\textstyle\sum_j [G'(z) - G(z)]_j \frac{\partial [G(z)]_j}{\partial [\theta_G]_k} \\ &= -\eta s(G(z)) \textstyle\sum_j [\nabla D(G(z))]_j \frac{\partial [G(z)]_j}{\partial [\theta_G]_k} \end{aligned}$$

By setting the scaling factor $s(x) = s_0(x)/\eta$, this is exactly the same as (20), required for GAN's generator update. (Recall that our theory and algorithms accommodate an arbitrary data-dependent scaling factor $s(x) > 0$.)

Thus, algorithmically GAN is closely related to a special case of xICFG that does the following:

- Let ICFG update the generator *just once* (i.e., $T = 1$).
- To update the approximator, take *only one* gradient descent step with *only one* mini-batch (i.e., small N)

instead of optimizing to the convergence with many examples. Therefore, the degree of approximation could be poor.

The same argument applies also to the log$d$-trick variant of GAN. The generator update with the log$d$ trick requires the same gradient as (20) except that s_0 becomes $s_1(x) = \frac{1}{1+\exp(D(x))}$; note that the change from $s_0(x) = \frac{1}{1+\exp(-D(x))}$ is the sign in front of D. Thus, GAN with the log$d$ trick is also closely related to the special case of xICFG above.

To compare GANs with and without the log$d$ trick, consider the situation where generated data x is very far from the real data and therefore $D(x) \ll 0$. In this situation, we have $s_0(x) \approx 0$ without the log$d$ trick, which would make the gradient (for updating θ_G) vanish, as noted in [10], even though the generator is poor and so requires updating. In contrast, we have $s_1(x) \approx 1$ with the log$d$ trick in this poor generator situation, and when the generated data x is close to the real data and therefore $D(x) \approx 0$, we have $s_1(x) \approx \frac{1}{2}$. Thus, with the log$d$ trick, the scaling factor of the gradient is likely to fall into $[\frac{1}{2}, 1)$, which is more sensible as well as more similar to our choice ($s(x) = 1$) for the xICFG experiments, compared with GANs without the trick.

xICFG vs. GANs: In spite of their connection, GANs are unstable, and xICFG with appropriate meta-parameters is stable (shown later). Thus, it is inferred that GAN's instability derives from what is unique to GANs, the two bullets above – *an extremely small* T and *coarse approximation*. Either can cause degradation of the generator, and when this happens, a GAN generator could fail to keep providing challenging examples to the discriminator, which would disrupt the balance of the progress of the discriminator and that of the generator, leading to instability.

ICFG vs. GANs: We have contrasted GANs and xICFG. Now we compare generator update of GANs and that of ICFG to consider the algorithmic merits of our functional gradient approach. The short-term goal of generator update can be regarded as the increase of the discriminator output on generated data, i.e., to have $D(G_{t+1}(z)) > D(G_t(z))$ for any $z \sim p_z$. ICFG updates the generator by $G_{t+1}(z) = G_t(z) + \eta \nabla D(G_t(z))$, and so with small η, $D(G(z))$ is guaranteed to increase for any z. This is because by definition $\nabla D(G_t(z))$ is the direction that increases the discriminator output for z, and it is *precisely* obtained *on the fly* for *every* z at the time of generation.

By contrast, GAN training *stochastically* and *approximately* updates θ_G using a *small sample* (one mini-batch SGD step that backpropagates ∇D), and so GAN's update can be noisy, which can lead to instability through generator degradation. Noise in each mini-batch SGD step would not be an issue, for example, in standard classifier training because the noise could be collectively cancelled out after many steps are taken. The GAN setting is different in that each generator update is immediately followed by discriminator update where the generator is used to produce input, which makes it prone to cascading failure.

5 Experiments

We tested xICFG on the image generation task.

5.1 Experimental setup

5.1.1 Baseline methods

For comparison, we tested the original GANs without the log$d$ trick ('GAN0') and with the log$d$ trick ('GAN1'), motivated by their relation to xICFG as analyzed above. As a representative of state-of-the-art methods, we tested WGAN with the gradient penalty (WGANgp) [11]. WGANgp has been shown to achieve stable training on a number of network architectures and datasets and rival or outperform a number of previous methods such as the original WGAN with weight clipping, Least Squares GAN [24], Boundary Equilibrium GAN [3], denoising feature matching [45], and Fisher GAN [30]. We also experimented with three of the more recent GAN training methods, which will be described later with their results.

5.1.2 Evaluation metrics

Making reliable likelihood estimates with generative adversarial models is known to be challenging [42], and we instead evaluated the visual quality of generated images by adopting the *inception score* [40] using datasets that come with labels for classification. The intuition behind this score is that high-quality images should lead to high confidence in classification. It is defined as $\exp(\mathbb{E}_x \mathrm{KL}(p(y|x)||p(y)))$ where $p(y|x)$ is the label distribution conditioned on generated data x and $p(y)$ is the label distribution over the generated data. Following previous work (e.g., [46], [5]), the probabilities were estimated by a classifier trained with the labels provided with the datasets (instead of the ImageNet-trained *inception* model used in [40]) so that the image classes of interest were well represented in the classifier. We, however, call this score the 'inception score', following custom. We note that the inception score is limited, e.g., it would not detect mode collapse or missing modes. Apart from that, we found that it generally corresponds well to human perception.

In addition, we used *Fréchet inception distance* (FID) of [13]. FID measures the distance between the distribution of $f(x^*)$ for real data x^* and the distribution of $f(x)$ for generated data x, where function f is set to convert an image to the internal representation of a classifier network; to obtain function f we used the same classifiers as used for the inception score evaluation. Let P_1 and P_2 be the two distributions of comparison, and let μ_1, μ_2 be their means and let Σ_1, Σ_2 be their covariance matrices. Then, the distance $d(P_1, P_2)$ we measure is defined by $d^2(P_1, P_2) = |\mu_1 - \mu_2|^2 + \mathrm{tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{1/2})$. If P_1 and P_2 are multivariate normal distributions (which can be completely described by the mean and covariance), then $d(P_1, P_2)$ can be proved to be the Fréchet distance [7]. One advantage of this metric is that it would be high (poor) if mode collapse occurs, and a disadvantage is that its computation is relatively expensive.

In the results below, we call these two metrics the (inception) *score* and the (Fréchet) *distance*.

5.1.3 Data

We used MNIST, the Street View House Numbers dataset (SVHN) [32], and the large-scale scene understanding (LSUN) dataset[3]. These datasets are provided with class labels (digits

3. http://lsun.cs.princeton.edu/2015.html

'0' – '9' for MNIST and SVHN and 10 scene types for LSUN). A number of studies have used only one LSUN class ('bedroom') for image generation. However, since a single-class dataset would preclude evaluation using class labels described above, we instead generated a balanced two-class dataset using the same number of training images from the 'bedroom' class and the 'living room' class (LSUN BR+LR). Similarly, we generated a balanced dataset from 'tower' and 'bridge' (LSUN T+B). The number of the images used as the training examples was 60K (MNIST), 521K (SVHN, the 'extra' set minus 10K for validation), 2.6 million (LSUN BR+LR), and 1.4 million (LSUN T+B). The number of held-out examples (used for the Fréchet distance evaluation) were 10K (MNIST and SVHN), 32K (LSUN BR+LR), and 17K (LSUN T+B). The LSUN images were shrunk and cropped into 64×64 as in previous studies [37], [2]. The pixel values were scaled into $[-1, 1]$.

5.1.4 Network architectures

The tested methods require as input a network architecture of a discriminator and that of an approximator or a generator. Among the numerous network architectures we could experiment with, we focused on two types with two distinct merits – good results and simplicity.

The first type (convolutional; stronger) aims at complexity appropriate for the dataset so that good results can be obtained. On MNIST and SVHN, we used an extension of DCGAN [37], adding 1×1 convolution layers. The DCGAN discriminator consists of convolution layers with stride 2 and batch normalization, and the generator is essentially the reverse of the discriminator, using transposed convolution layers with stride 2 and batch normalization. For LSUN, whose images are larger (64×64) and whose number of training examples is also larger, we used a residual net (ResNet) [12] of four residual blocks, which is a simplification from the WGANgp code release, for both the discriminator and the approximator/generator. Details are given in Appendix C.2.1

These networks include batch normalization layers [18]. [11] states that WGANgp does not work well with a discriminator with batch normalization. Although it would be ideal to use exactly the same networks for all the methods, it would be rather unfair for the other methods if we always remove batch normalization. Therefore, we removed batch normalization from D (as suggested by [11]) for experimenting with WGANgp while we used the network definitions as they are for the rest; also, we tested cases without batch normalization anywhere for all the methods.

The second type (fully-connected $\widetilde{G}$ or G; weaker) uses a minimally simple approximator/generator, consisting of two 512-dim fully-connected layers with ReLU, followed by the output layer with $\tanh$, which has a merit of simplicity, requiring less design effort. We combined it with a convolutional discriminator, the DCGAN extension above.

5.1.5 xICFG implementation details

To speed up training, we limited the number of epochs of the approximator training in xICFG to 10 while reducing the learning rate by multiplying by 0.1 whenever the training loss stops going down. The scaling function $s(x)$ in ICFG (Line 7 of Algorithm 2) was set to $s(x) = 1$. To initialize the approximator $\widetilde{G}$ for xICFG, we first created a simple generator $G_{\mathrm{rand}}(z)$ consisting of a projection layer with random weights set by the Gaussian distribution with zero mean and standard deviation 0.01 to produce data points of the desired dimensionality, and then trained $\widetilde{G}$ to approximate the behavior of G_{rand}. The training time reported below includes the time spent for this initialization. Unlike our short paper [19], no input pooling was performed.

b	mini-batch size	64
U	discriminator update frequency	1
N	# of examples used for updating $\widetilde{G}$	$10b$
T	number of iterations in ICFG	25

TABLE 2: Meta-parameters for xICFG.

5.1.6 Other details

In all cases, the prior p_z was set to generate 100-dimensional Gaussian vectors with zero mean and standard deviation 1. All the experiments were done using a single NVIDIA Tesla P100.

The meta-parameter values for xICFG were fixed to those in Table 2 unless otherwise specified. For GANs, we used the same mini-batch size as xICFG, and we set the discriminator update frequency U to 1 as other values often led to poorer results. The SGD update was done with *rmsprop* [43] for xICFG and GANs. The learning rate for rmsprop was fixed for xICFG for both the discriminator update and the approximator training, but we tried several values for GANs as it turned out to be critical. Similarly, for xICFG, we found it important to set the step size η for the generator update in ICFG to an appropriate value. The SGD update for WGANgp was done with Adam [21] with meta-parameters set to the values suggested by [11] except that we tried several values for the learning rate. Thus, the amount of tuning effort was about the same for all. Tuning was done based on the performances on the validation set of 10K input vectors (i.e., 10K 100-dim Gaussian vectors), and we report the results on the test set of 10K input vectors, disjoint from the validation set.

5.2 Results

5.2.1 On the quality of generated images

Inception score results: First, we report the inception score results. The scores of the real data (in the held-out sets) are 9.91 (MNIST), 9.13 (SVHN), 1.84 (LSUN BR+LR), and 1.90 (LSUN T+B), respectively, which roughly set the upper bounds that can be achieved by generated images. Fig. 2 and 3 show the score of generated images (in relation to training time) with the convolutional networks with and without batch normalization, respectively. As discussed in Section 4.3 a smaller T has practical advantages of a smaller generator resulting in faster generation and smaller footprints while a larger T stabilizes xICFG training by ensuring that training makes progress by overcoming the degradation caused by approximation. With convolutional networks, we tested T=15 in addition to T=25 (the value shown in Table 2 which worked well for all) and found that setting T=15 also achieves stable training. The results in Fig. 2–3 were obtained by setting T=15. xICFG generally performs well compared

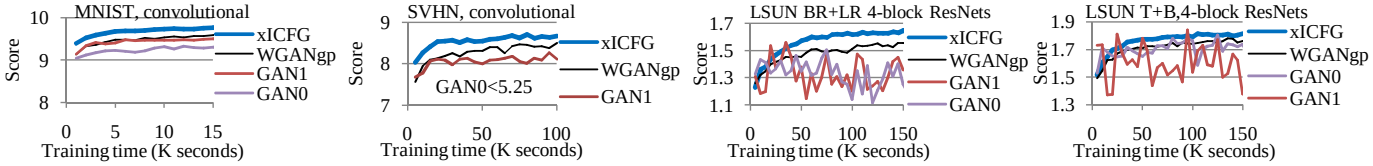


Fig. 2: Image quality measured by the inception score in relation to training time. Convolutional networks. The legends are roughly sorted from the best to the worst.

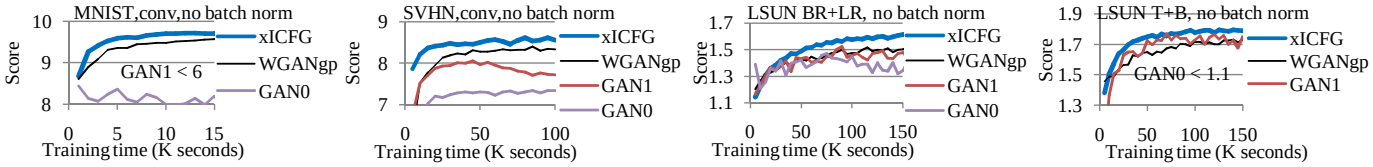


Fig. 3: Image quality measured by the inception score in relation to training time. Convolutional networks without batch normalization anywhere. The legends are roughly sorted from the best to the worst.

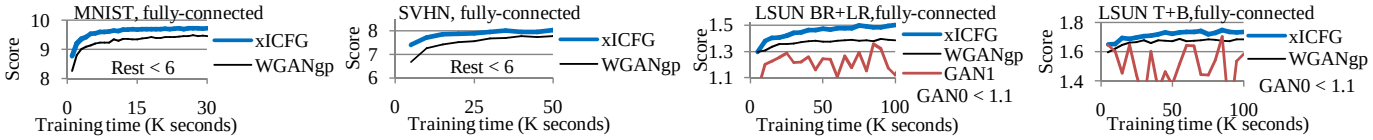


Fig. 4: Image quality measured by the inception score in relation to training time. Fully-connected approximators/generators.

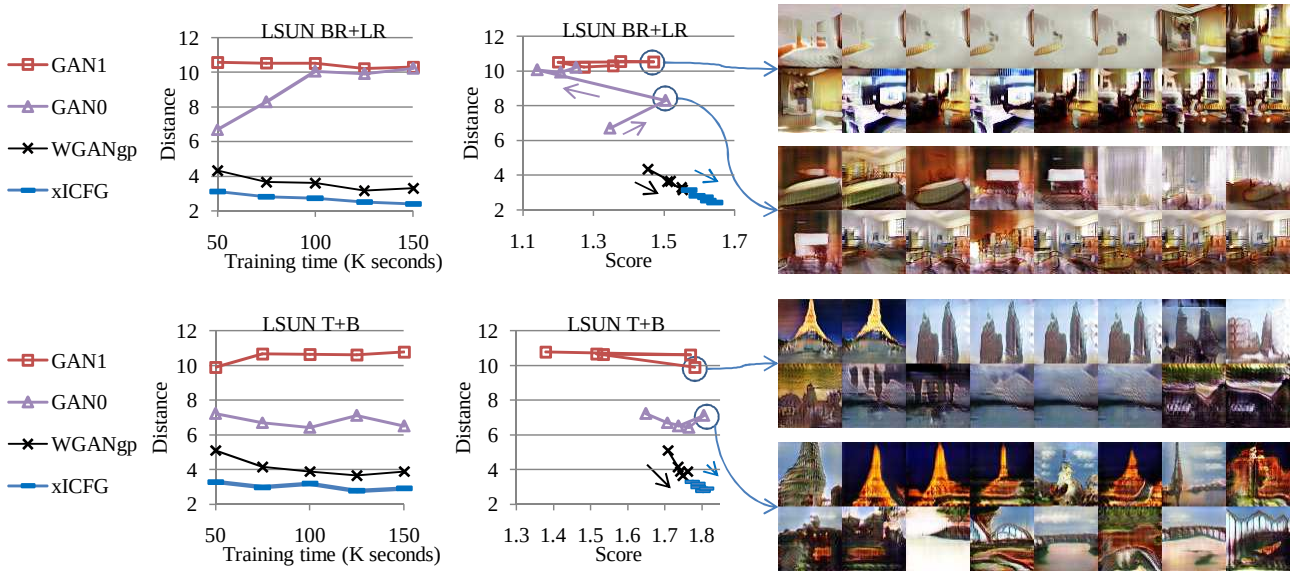
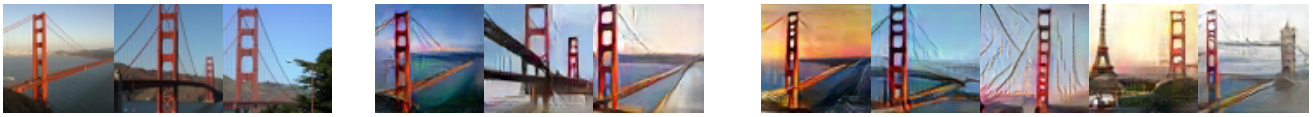
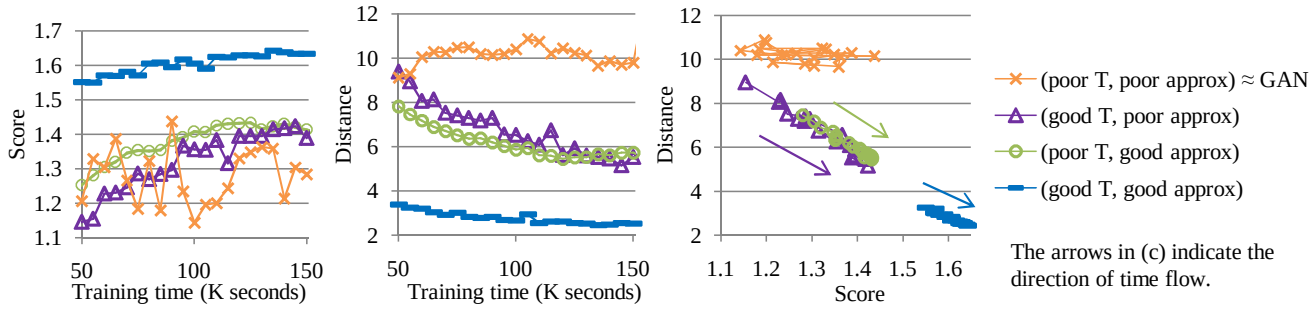


Fig. 5: Fréchet distance in relation to training time (left) and the inception score (right) on the runs in Fig 2. The arrows in the right graphs show the direction of time flow. On both LSUN BR+LR (up) and T+B (down), GAN0 and GAN1 suffer from mode collapse or lack of diversity; their inception score fluctuates (Fig 2) and their Fréchet distance stays relatively high. xICFG (and WGANgp) shows no sign of mode collapse (Fig 13–14) and performs well in both metrics.



(a) Real images. (b) "Realistic" images (xICFG) (c) "Creative" images (xICFG)

Fig. 6: "Realistic" and "creative" images generated by xICFG. (a) Real Golden Gate Bridge images in the training set. (b) Images generated by xICFG that look like Golden Gate Bridge though not perfect. (c) Images generated by xICFG that look like modifications of Golden Gate Bridge.



(a) Inception score vs. time (b) Fréchet distance vs. time (c) Inception score vs. Fréchet distance

Fig. 7: xICFG. LSUN BR+LR. 4-block ResNets. The setting almost equivalent to GANs performs poorly, similar to GANs. The performance improves as we improve the setting of T and the approximator update.

with others. WGANgp also achieves stable training, but its score generally falls a little short of xICFG. On LSUN datasets, GAN1 occasionally exceeds xICFG, but inspection of generated images reveals that it suffers from severe mode collapse. The inception score results with the simple but weak fully-connected approximator/generator are shown in Fig. 4. xICFG achieved stable training. Among the baseline methods, only WGANgp succeeded in this setting, but its score fell behind xICFG. These results show that xICFG is effective and efficient.

		MNIST	SVHN	BR+LR	T+B
Convolutional (Fig. 2)	xICFG	*3.35*	*5.29*	*2.42*	*2.92*
	WGANgp	*3.41*	*5.34*	3.31	3.90
	GAN0	4.55	10.60	10.21	6.50
	GAN1	4.56	5.82	10.31	10.77
Fully-conn. (Fig. 4)	xICFG	*3.39*	*5.93*	*4.65*	*5.23*
	WGANgp	4.67	6.21	5.80	6.55
	GAN0	58.59	18.25	14.86	32.49
	GAN1	38.77	22.40	11.27	12.84

TABLE 3: Fréchet distance results. Evaluated at the maximum training time of the respective graphs.

Fréchet distance results: Table 3 shows the Fréchet distance results evaluated at the end of training runs shown in Fig. 2 and 4. xICFG generally performs well, followed by WGANgp except that with convolutional networks on MNIST and SVHN, WGANgp and xICFG produce similar values. We found that the Fréchet distance mostly correlates with the inception score, i.e., generally, when the score is good (large), the distance is also good (small). One prominent exception is when mode collapse occurs, and Fig. 5 shows examples of this on the LSUN datasets. That is, GAN0 and GAN1 both suffer from mode collapse or lack of diversity on these datasets (Fig. 5 right); while their inception score fluctuates, the Fréchet distance stays large, correctly indicating that the distribution of generated data differs from that of real data. By contrast, xICFG and WGANgp show no sign of severe mode collapse as shown later in Fig. 13–14, and their Fréchet distance improves (goes down) as their inception score improves (goes up), outperforming GAN0 and GAN1 in both metrics (Fig. 5).

Visual inspection of generated images: Examples of generated images are shown in Fig. 11–15. The MNIST and SVHN images in Fig. 11–12 were randomly chosen and sorted by the predicted classes. To use limited space effectively, we take a more focused approach for larger LSUN images and show the 'best' and 'worst' images, in terms of the confidence of a classifier. That is, in Fig. 13–15 we show images that were assigned (by the classifier used for evaluation) the highest probability of being, e.g., a "bedroom" or a "living room" on LSUN BR+LR and also show the images with the highest entropy values. They are the best and the worst contributors to the inception score, respectively, as well.

Note that small samples may not represent the population well due to variability. Even so, it is clear that the images in Fig. 11f, 12c, 12f, 13d, 14d, 15c, and 15f suffer from low quality and/or mode collapse or lack of diversity. These images were generated by either GAN0 or GAN1.

xICFG and the best-performing baseline (WGANgp) consistently produce visibly better images than the original GANs in all the settings. Overall, we feel that visual impressions of the images generated by xICFG are sometimes better than and at least as good as those of WGANgp, which is one of the state-of-the-art methods. More image examples are shown in Appendix C.1.

No memorization: When generated images look somewhat realistic, one may wonder if the generator is memorizing training images instead of capturing the essence of images. In Fig. 6, we show examples that indicate that xICFG does something more 'creative' than memorization. The LSUN T+B includes a number of pictures of Golden Gate Bridge. As seen in Fig. 6a, in reality, Golden Gate Bridge's tower component (the reddish vertical object) has four grids above the horizontal part. xICFG generates images that look like Golden Gate Bridge (Fig. 6b) though they are not perfect (the towers look good, but the wires are wobbly and the ocean is missing from some). It also generates images that look like modifications of Golden Gate Bridge (Fig. 6c), making the tower component longer with more grids, placing it with objects that are not there in reality, and so forth.

5.2.2 *Transition from GAN to 'good' xICFG*

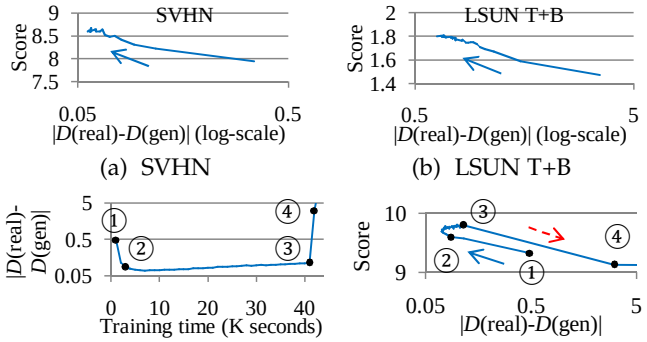
We have shown in Section 4.4 that the original GAN is closely related to the special (and extreme) case of xICFG that sets T to the minimum (T=1) and makes the minimal effort for updating the approximator, going over only one mini-batch just once. We also presented an insight that since xICFG with appropriate meta-parameters is stable, GAN's instability could be due to these differences – extremely small T and poor approximation.

To follow up on this, we experimented with xICFG with meta-parameter settings that transition from the poor setting corresponding to GAN towards the good setting used above. The results using the convolutional networks (used in Fig. 2) on LSUN BR+LR are shown in Fig. 7. The legends in this figure represent the following:

- "poor T": T=1
- "good T": T=15
- "poor approx": Poor approximation iterating only once over only one mini batch ($N = b$) for updating the approximator.
- "good approx": Good approximation iterating 10 times over 10 mini batches ($N = 10b$) for updating the approximator, as was done in Fig. 2–5.

As predicted, "(poor T, poor approx)≈ GAN" performs just like GANs – the inception score fluctuates a lot and the Fréchet distance stays large; manual inspection of images indicates severe mode collapse. When we improve either the value of T ("(good T, poor approx)") or approximation ("(poor T, good approx)"), training becomes more stable and the performance in both metrics improves as training proceeds; however, the performance improvement stops without catching up with "(good T, good approx)". That is, in order to obtain the best results, we need to have both right – sufficiently large T *and* good approximation. The results are consistent with our theoretical insight into GANs presented earlier and support our CFG approach.

5.2.3 On the discriminator output values



(a) SVHN (b) LSUN T+B

(c) Eventual collapse of training on MNIST.

Fig. 8: Relations between image quality and $|D(\text{real}) - D(\text{gen})|$ (=Δ_D). The arrows indicate the direction of time flow. A correlation is observed both when training is succeeding (blue solid arrows) and failing (red dotted arrows).

Successful training should make it harder and harder for the discriminator to distinguish real images and generated images, which would manifest as the discriminator output values for real images and generated images becoming closer and closer. We quantify this notion by the difference between discriminator output values for real images and generated images averaged over time intervals of a fixed length, obtained as a by-product of the forward propagation for updating the discriminator. We call it '$|D(\text{real}) - D(\text{gen})|$' or Δ_D in short.

Fig. 8 shows that, as expected, Δ_D generally correlates with the progress of training. When training is going well (indicated by blue solid arrows), Δ_D decreases and the inception score improves as training proceeds. When it is failing, Δ_D goes up rapidly and the inception score degrades rapidly; see the change from the state #3 to #4 in Fig 8c. Here, due to excessive training, the discriminator is overfitting little by little to relatively small MNIST training data (#2 to #3), increasing ϵ of the ϵ-approximation. (Note that this is overfitting in the standard sense as it was confirmed that the average of $D(x)$ on the real data in the held-out set became smaller and smaller than that of the training set.) That slows down and eventually stops the progress of the generator, resulting in the rapid increase of Δ_D and rapid degradation of the generator (#3 to #4). In practice, training should be stopped before the rapid growth of Δ_D – *early stopping*. Thus, the decrease/increase of Δ_D values, which can be obtained at almost no cost during training, can be used as an indicator of the status of xICFG training.

Additionally, it might be worth mentioning that the Δ_D value is related to the sum of the KL divergence and the reverse KL divergence between the two distributions, which were shown to be optimized by the CFG algorithm (Section 2). Details are given in Appendix C.2.3.

5.2.4 On the values of U and N

We have shown that T can be reduced for practical advantages in some cases, and here we consider the effects of changing U (discriminator update frequency) and N (the number of examples used for updating the approximator) from their default values, U=1 and N=10b where b is the mini-batch size.

Essentially, a larger U may make the discriminator closer to the optimum, but a very large U would increase the risk of discriminator overtraining. A larger N would make better approximation, but it may slow down training. If one takes the viewpoint that a discriminator should capture the essence of real data, and it should become better and better in doing so each time it is updated, then one can argue that a larger (but not too large) U would increase the amount of discriminator improvement per each call of ICFG, and therefore, also increase the amount of xICFG generator improvement per xICFG iteration. This viewpoint suggests that to benefit from a larger U, we should also consider increasing N so that the approximator $\tilde{G}$ can keep up with the generator instead of becoming a bottleneck.

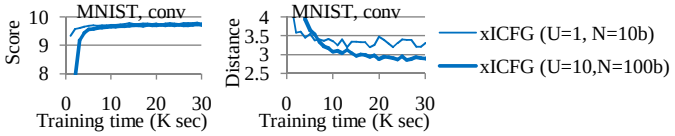


Fig. 9: Larger values for U and N on MNIST. Convolutional.

We tested U=5 and U=10 with convolutional networks used in Fig. 2 and found that stable training can be achieved also with these values while in some cases N also needed to be increased as expected. On MNIST the increase of U (and N) improved the Fréchet distance while producing similar inception scores (Fig. 9). On the other datasets, a larger U and/or N either exhibited similar performances (SVHN) or slowed down training (LSUN), and it was not clear whether doing so was advantageous.

We note that MNIST differs from the other tested datasets in that training data is much smaller (2–10% of the others)

and that the images are simpler (grayscale digits). These differences make MNIST somewhat uniquely prone to discriminator overfitting; indeed, if trained excessively long with the default meta-parameters, Δ_D starts going up (Fig 8c). On MNIST, a larger U and N makes training more stable (no increase of Δ_D after 30K seconds) and improves the Fréchet distance.

Thus, our finding is that the default values are a good starting point, and it depends on the datasets and possibly network architectures whether more careful meta-parameter tuning pays off.

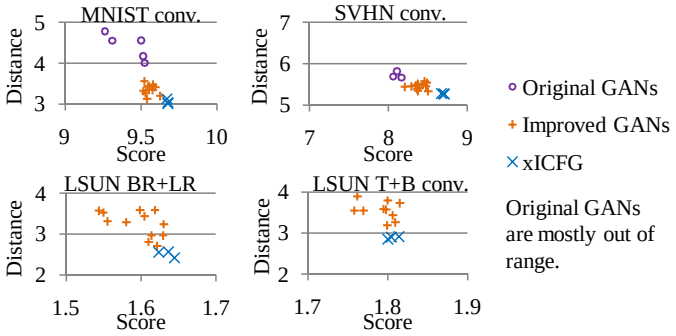
5.2.5 Comparison with recent GAN variants

Finally, we compare xICFG with more recent GAN training methods that appeared around the first publication of this work: spectral normalization (hereafter GAN-sn) [29] and zero-centered gradient penalty regularization (GAN-gp) [26]. These methods stabilize GAN training by normalization/regularization of the discriminator. GAN-sn constrains the weight matrix of each layer to have the unit 2-norm. GAN-gp penalizes larger gradients by a regularization term $\gamma \mathbf{E}_{x\sim q}\|\nabla D(x)\|^2$ where q is the distribution of either real data or generated data (corresponding to R_1- or R_2-regularizer of [26]), and we tested both. Following the tuning protocol above, we made substantial efforts to obtain their best performances; details are given in Appendix C.2.4.

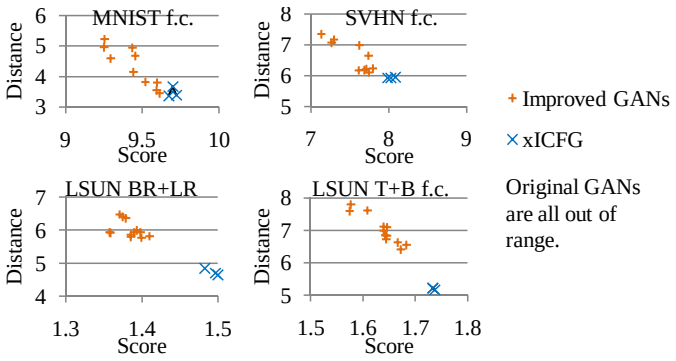
In this series of experiments, we performed 3 runs per method using 3 different random seeds in order to account for the variability caused by random factors such as weight initialization. For comparison, we also performed 3 runs of WGANgp and the original GANs.

Fig. 10 shows the scores (x-axis) and the Fréchet distances (y-axis) on (a) the convolutional networks and (b) the fully-connected approximators/generators, used in Fig. 2 and 4, respectively. Each method has three points in each graph, except that the original GAN results are shown only when they are as good as the rest and so they can fit in the same graph. Since higher scores and smaller distances are better, the points closer to the bottom right of each graph are better. The baseline methods are organized into two groups: 'Original GANs' (GAN0 and GAN1) and 'Improved GANs' (GAN-sn, GAN-gp of two types, and WGANgp);

Improved GANs indeed show clear and consistent improvements over the original GANs. In many cases, the performances of the original GANs are poor to the extent that they are out of range of the graphs. xICFG performs consistently well, irrespective of the random seeds, and it generally exceeds or rivals the improved GANs. It should also be noted again, however, that the generator model of xICFG is larger than the baseline methods; thus, footprints are larger and generation is slower, which may need consideration in practical applications.



(a) Using convolutional networks as in Fig.2.



(b) Using fully-connected approximators/generators as in Fig.4.

Fig. 10: Scores (x-axis) and distances (y-axis). 3 runs (with 3 random seeds) per method. 'Original GANs': GAN0 and GAN1. 'Improved GANs': GAN-sn, GAN-gp of two types, and WGANgp.

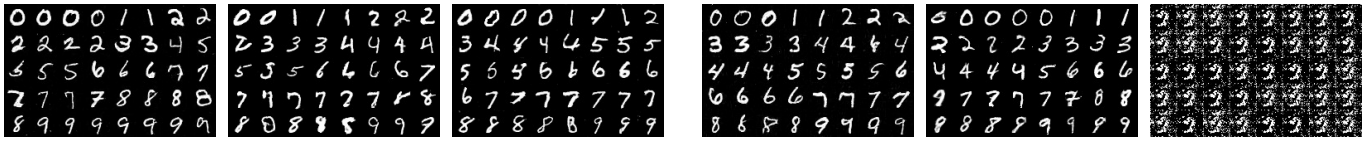
6 Related Work

Analyzing and improving GANs: Improvements of GANs are often based on analyses of GAN training. As a result, they typically stay within the minimax optimization framework of the original GAN. In contrast, our framework is that of greedy learning of functional compositions, which puts our work in a relatively unique position. There are a large number of studies related to GANs, and here we focus on a small number of those perceived as most related.

One approach to improving GANs involves change of the training objective, e.g., WGAN [2], [1], Least squares GAN [24], and f-GAN [34]. We have discussed f-GAN in Section 2.2.1 and the low-dim manifold view of [1] in Section 4.1. [11] regularizes the discriminator of WGAN with the *gradient penalty* term (WGANgp), adopted by many studies, and we compared it with xICFG in our experiments.

Another type of approach involves regularization or normalization of GANs. Spectral normalization constrains the weight matrix of each layer to have the unit 2-norm, proposed by [29] and adopted for class-conditional GANs [4], [47]. Regularization of the discriminator by penalizing large $\|\nabla D(x)\|$ was proposed in [38], [26]. *Jacobian clamping* [35] keeps the singular values of the input-output Jacobian of the generator in a pre-defined range. [31] regularizes the generator by penalizing large gradients with respect to model parameters of the discriminator. [27] penalizes large gradients with respect to model parameters both on the generator and the discriminator. We reported performance comparison with spectral normalization [29] and the gradient penalty regularization methods of [26].

While the theoretical motivations of these methods vary (e.g., approximating the effect of adding noise [38], [26], a local stability analysis near an equilibrium point [31], [26], a Jacobian analysis of the gradient vector field [27]), it appears that generally, these methods stabilize GAN training by preventing sudden large changes which could immediately push the minimax game into a vicious cycle. [4] and [27] have

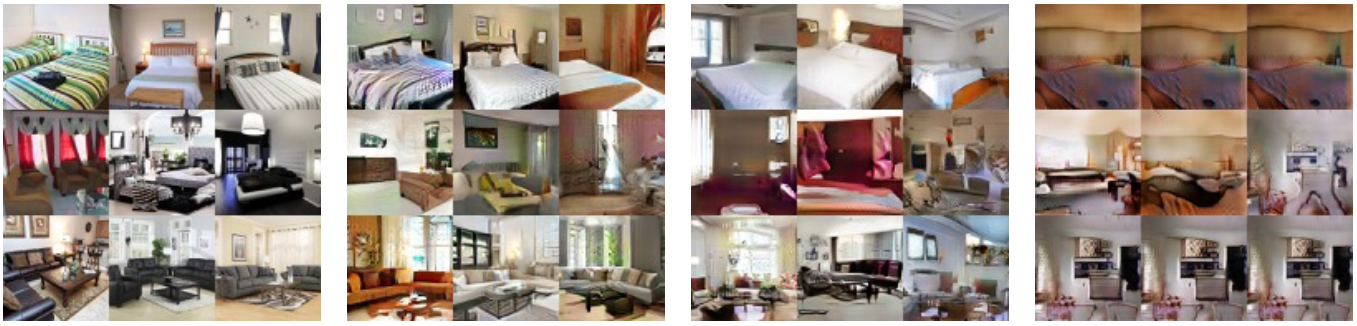


(a) xICFG (b) Best baseline (c) Worst baseline (d) xICFG (e) Best baseline (f) Worst baseline

Fig. 11: MNIST. Random samples sorted by the predicted classes. (a-c) Convolutional networks. (d-f) Fully-connected $\widetilde{G}$ or G.

(a) xICFG (b) Best baseline (c) Worst baseline (d) xICFG (e) Best baseline (f) Worst baseline

Fig. 12: SVHN. Random samples sorted by the predicted classes. (a-c) Convolutional networks. (d-f) Fully-connected $\widetilde{G}$ or G.



(a) Real images (b) xICFG (c) Best baseline (d) Second best baseline

Fig. 13: LSUN BR+LR (64×64). 4-block ResNets. These images are most likely bedrooms (1st row), most uncertain (2nd row), most likely living rooms (3rd row), among a random sample of 1000, according to a classifier.

(a) Real images (b) xICFG (c) Best baseline (d) Second best baseline

Fig. 14: LSUN T+B (64×64). 4-block ResNets. These images are most likely towers (1st row), most uncertain (2nd row), most likely bridges (3rd row), among a random sample of 1000, according to a classifier.

(a) xICFG (BR+LR) (b) Best baseline (c) 2nd best baseline (d) xICFG (T+B) (e) Best baseline (f) 2nd best baseline

Fig. 15: Weak $\widetilde{G}/G$ (fully-connected) on LSUN BR+LR (a-c) and T+B (d-f). 64×64. The images were chosen as in Fig. 13 and 14.

noted cases where regularization stabilizes GAN training but leads to a poor solution. Our xICFG is free of this potential problem since it does not involve regularization. On the other hand, many of these regularization/normalization techniques (in particular, those which work on the discriminator) can be easily integrated into xICFG, and doing so may be useful in some situations.

Unrolled GANs [28] optimize the generator with respect to an unrolled optimization of the discriminator in order to make generator update closer to the state of using the optimal discriminator without actually updating the discriminator. There is a high-level similarity between unrolled GANs and xICFG as both involve multiple states of the discriminator.

[20] generated high-resolution images by incrementally adding layers for higher resolutions, using WGANgp as a base learner. We view xICFG as a component that, similar to WGANgp, can be used to build higher levels of architecture such as that of [20] and stacked GANs [15], [48].

[22] described a cascading process related to CFG, motivated by Langevin dynamics sampling, called *introspective neural networks*. Based on the theory of Langevin, their generation process requires repeated noise addition, and so our generation is simpler.

Functional gradient learning: Our approach can be regarded as functional gradient learning. Similar to gradient boosting, where more and more decision trees are added to the model as training proceeds, ICFG adds more and more components (layers if the discriminator is a neural network) to the generator, following the gradients in a function space.

Natural gradient descent [17], [16] has been applied to neural networks for deep learning, e.g., [41], [25], [36], [39]. It can be regarded as moving in a function space so that the change in the objective is maximized per a fixed amount of move in the function space, where moves are measured by the KL divergence. However, the function space explored by natural gradient descent corresponds to the one obtained by changing parameters of a fixed neural network structure. By contrast, CFG and its variants dynamically *grows* a generator, as illustrated in Fig.1.

In parallel to our work, [33] proposed *gradient layers* for fine-tuning WGAN, which is similar to our ICFG. As noted above, ICFG suffers from the issue of large generators *if* used for image generation *from scratch*, and so does insertion of gradient layers. This issue was the motivation for xICFG, which periodically compresses the generator by approximation.

7 Conclusion

In the generative adversarial learning setting, we considered a generator that can be obtained using composite functional gradient learning. Our theoretical results led to the new stable algorithm xICFG. The experimental results showed that xICFG generated equally good or better images than GAN and WGAN variants in a stable manner.

References

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2017.

[3] David Berthelot, Thomas Schumm, and Luke Metz. BEGAN: Boundary equilibrium generative adversarial networks. *arXiv:1703.10717*, 2017.

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

[5] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[6] Imre Csiszár and Paul C. Shields. Information theory and statistics: a tutorial. *Communications and Information Theory*, 1(4):417–528, 2004.

[7] D. C. Dowson and B. V. Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12:450–455, 1982.

[8] Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 2001.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, and Bernhard Nessler. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Proceedings of Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

[15] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. Stacked generative adversarial networks. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] Shun ichi Amari. Neural learning in structured parameter spaces – natural Riemannian gradient. In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, 1996.

[17] Shun ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning (ICML)*, 2015.

[19] Rie Johnson and Tong Zhang. Composite functional gradient learning of generative adersarial models. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

[20] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

[22] Justin Lazarow, Long Jin, and Zhuowen Tu. Introspective neural networks for generative modeling. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017.

[23] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[24] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. *arXiv:1611.04076*, 2017.

[25] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of International Conference on Machine Learning (ICML)*, 2015.
[26] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.
[27] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of GANs. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.
[28] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.
[29] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.
[30] Youssef Mroueh and Tom Sercu. Fisher GAN. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.
[31] Vaishnavh Nagarajan and J. Zico Kolter. Gradient descent gan optimization is locally stable. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.
[32] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *Proceedings of NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
[33] Atsushi Nitanda and Taiji Suzuki. Gradient layer: Enhancing the convergence of adversarial training for generative models. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.
[34] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.
[35] Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B. Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to gan performance? In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.
[36] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2014.
[37] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015.
[38] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.
[39] Nicolas L. Roux, Pierre antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, 2007.
[40] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.
[41] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of International Conference on Machine Learning (ICML)*, 2015.
[42] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.
[43] Tijman Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.
[44] Ilya Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. AdaGAN: Boosting generative models. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.
[45] David Warde-Farley and Yoshua Bengio. Improving generative adversarial networks with denoising feature matching. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.
[46] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. LR-GAN: Layered recursive generative adversarial networks for image generation. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.
[47] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv:1805.08318*, 2018.
[48] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017.
[49] Tong Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32(1):56–85, 2004.

Rie Johnson Rie Johnson received the Ph.D. degree in computer science from Cornell University, Ithaca, NY, USA, in 2001. She was a research scientist with the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, until 2007. Her current research interests include machine learning and its applications.

Tong Zhang Tong Zhang is a professor of Computer Science and Mathematics at the Hong Kong University of Science and Technology. His research interests are machine learning, big data and their applications. He obtained a BA in Mathematics and Computer Science from Cornell University, and a PhD in Computer Science from Stanford University. Before joining HKUST, Tong Zhang was a professor at Rutgers University, and worked previously at IBM, Yahoo as research scientists, Baidu as the director of Big Data Lab, and Tencent as the director of AI Lab. Tong Zhang was an ASA fellow and IMS fellow, and has served as the chair or area-chair in major machine learning conferences such as NIPS, ICML, and COLT, and has served as associate editors in top machine learning journals such as PAMI, JMLR, and Machine Learning Journal.

APPENDIX A
SUPPLEMENTS TO SECTION 2 (CONTINUOUS THEORY)

Proof of Theorem 2.1 (continuous CFG)

We use $\|\cdot\|$ to denote the vector 2-norm and the matrix spectral norm (the largest singular value of a matrix). Given a differentiable scalar function $h(x):\mathbb{R}^k\to\mathbb{R}$, we use $\nabla h(x)$ to denote its gradient, which becomes a k-dimensional vector function. Given a differentiable function $g(x):\mathbb{R}^k\to\mathbb{R}^k$, we use $\nabla g(x)$ to denote its Jacobi matrix and we use $\nabla\cdot g(x)$ to denote the divergence of $g(x)$, defined as

$$\nabla\cdot g(x):=\sum_{j=1}^{k}\frac{\partial[g(x)]_j}{\partial[x]_j},$$

where we use $[x]_j$ to denote the j-th component of x. We know that

$$\int\nabla\cdot w(x)dx=0 \tag{21}$$

for all vector function $w(x)$ such that $w(\infty)=0$.

To prove Theorem 2.1, we also need the following lemma, which specifies the dynamics of p_t.

Lemma A.1. Using the definitions and notation in Theorem 2.1, we have

$$\frac{dp_t(x)}{dt}=-\nabla p_t(x)\cdot g_t(x)-p_t(x)\nabla\cdot g_t(x) \tag{22}$$

for all data $x\in\mathbb{R}^k$.

Proof

To simplify notation, let random variable $X=G_t(Z)\in\mathbb{R}^k$, and let p be the probability density of X. Let $X'=G_{t+\delta}(Z)$ for a small δ so that X' is a random variable transformed from X by $X'=X+\delta g(X)+o(\delta)$ and let p' be the probability density of X'. For an arbitrary $x'\in\mathbb{R}^k$, let $x'=x+\delta g(x)+o(\delta)$. Then we have

$$\begin{aligned}
p'(x')=&p(x)\left|\det(dx'/dx)\right|^{-1}\\
=&p(x)\left|\det(I+\delta dg(x)/dx+o(\delta))\right|^{-1}\\
=&p(x)(1+\delta\nabla\cdot g(x)+o(\delta))^{-1}\\
=&p(x)(1-\delta\nabla\cdot g(x)+o(\delta)) &(23)\\
=&p(x)-\delta p(x')\nabla\cdot g(x')+o(\delta) &(24)\\
=&p(x')-\delta g(x')\cdot\nabla p(x')-\delta p(x')\nabla\cdot g(x')+o(\delta). &(25)
\end{aligned}$$

The first three equalities used the multivariate change of variables formula for probability densities for the change from X to X' and the definition of determinant with terms explicitly expanded up to $O(\delta)$. (23) used the Taylor expansion of $(1+z)^{-1}=1-z+o(z)$ with $z=\delta\nabla\cdot g(x)$. (24) follows from the fact that $p(x')=p(x)+o(1)$, $\nabla\cdot g(x')=\nabla\cdot g(x)+o(1)$. (25) is due to $p(x)=p(x')-(x'-x)\cdot\nabla p(x')+o(\delta)$. Since $x'\in\mathbb{R}^k$ is arbitrary, this implies that:

$$p'(x)=p(x)-\delta g(x)\cdot\nabla p(x)-\delta p(x)\nabla\cdot g(x)+o(\delta),$$

for all $x\in\mathbb{R}^k$, which leads to the desired result by taking $\delta\to0$, setting $g=g_t$, and noting that $p=p_t$ as both are the density of $G_t(Z)$. ■

Proof of Theorem 2.1

Using the chain rule and Lemma A.1, we have

$$\begin{aligned}
\frac{d}{dt}\ell(p_*(x),p_t(x))=&\ell_2'(p_*(x),p_t(x))\frac{d}{dt}p_t(x)\\
=&\ell_2'(p_*(x),p_t(x))(-\nabla p_t(x)\cdot g_t(x)-p_t(x)\nabla\cdot g_t(x))\\
=&p_t(x)\nabla_x\ell_2'(p_*(x),p_t(x))\cdot g_t(x)-\nabla_x\cdot[\ell_2'(p_*(x),p_t(x))p_t(x)g_t(x)]\,. &(26)
\end{aligned}$$

The third equality follows from

$$\begin{aligned}
\nabla_x\cdot[\ell_2'(p_*(x),p_t(x))p_t(x)g_t(x)]=\;&\ell_2'(p_*(x),p_t(x))p_t(x)[\nabla\cdot g_t(x)]\\
&+\ell_2'(p_*(x),p_t(x))[\nabla p_t(x)]\cdot g_t(x)\\
&+[\nabla_x\ell_2'(p_*(x),p_t(x))]\cdot[p_t(x)g_t(x)],
\end{aligned}$$

which is obtained by applying the product rule to $\nabla_x\cdot[\ell_2'(p_*(x),p_t(x))p_t(x)g_t(x)]$. Now, by integrating (26) over x, and by using the fact that $\int\nabla\cdot f(x)dx=0$ with $f(x)=\ell_2'(p_*(x),p_t(x))p_t(x)g_t(x)$, we obtain the desired bound.

APPENDIX B
SUPPLEMENTS TO SECTION 3 (DISCRETE THEORY)

B.1 Proof of Theorem 3.1

Theorem 3.1 in the main paper can be obtained by setting $\alpha = 1$ in Theorem B.1 below.

B.2 Weighted version of discrete theory: Theorem B.1

Theorem 3.1 is a special case of Theorem B.1 below, where we consider a weighted logistic regression problem and a weighted KL divergence. We first state the definitions and assumptions of the weighted version.

B.2.1 Definitions

Let $\alpha \in (0.5, 1]$ be a tuning parameter, and let $\bar{\alpha} = 1 - \alpha$. Theorem 3.1 is obtaned by setting $\alpha = 1$ in Theorem B.1.

Define a function $\mathcal{D}_\alpha$ as:

$$\mathcal{D}_\alpha(x) := \ln \frac{\alpha p_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} p_*(x) + \alpha p(x)} \tag{27}$$

Then, $\mathcal{D}_\alpha$ is the analytical solution to the following weighted logistic regression problem:

$$\mathcal{D}_\alpha = \arg\min_{D'} \left[\alpha \left(\mathbf{E}_{x\sim p_*} \ln(1 + e^{-D'(x)}) + \mathbf{E}_{x\sim p} \ln(1 + e^{D'(x)}) \right) + \bar{\alpha} \left(\mathbf{E}_{x\sim p} \ln(1 + e^{-D'(x)}) + \mathbf{E}_{x\sim p_*} \ln(1 + e^{D'(x)}) \right) \right].$$

In practice, $\mathcal{D}_\alpha$ should be approximated by solving:

$$\arg\min_{D'} \left[\alpha \left(\sum_{x\in S_*} \frac{\ln(1 + e^{-D'(x)})}{|S_*|} + \sum_{x\in S} \frac{\ln(1 + e^{D'(x)})}{|S|} \right) + \bar{\alpha} \left(\sum_{x\in S} \frac{\ln(1 + e^{-D'(x)})}{|S|} + \sum_{x\in S_*} \frac{\ln(1 + e^{D'(x)})}{|S_*|} \right) \right], \tag{28}$$

where S_* and S are sets of data points sampled according to p_* and p, respectively. Let D_α be the solution to this empirical problem.

Define the weighted KL divergence by:

$$L_\alpha(p) := \int (\alpha p_*(x) + \bar{\alpha} p(x)) \ln \frac{\alpha p_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} p_*(x) + \alpha p(x)} dx \,. \tag{29}$$

B.2.2 Assumptions

Assumptions B.1 and B.2 below are the weighted versions of Assumptions 3.1 and 3.2, respectively.

Assumption B.1 (Boundedness of the weighted pdf ratio). $|\mathcal{D}_\alpha(x)|$ is bounded so that there exists a positive number $B < \infty$ such that $|\mathcal{D}_\alpha(x)| \le B$.

Assumption B.2 (ϵ-approximation condition on the discriminator). D_α, the empirical solution to (28), satisfies the following ϵ-approximation condition for some ϵ such that $0 \le \epsilon < \infty$

$$\int q_*(x) \left(\alpha \left| D_\alpha(x) - \mathcal{D}_\alpha(x) \right| + \bar{\alpha} \left| e^{D_\alpha(x)} - e^{\mathcal{D}_\alpha(x)} \right| \right) dx \le \epsilon \quad \text{where} \quad q_*(x) = p_*(x) \max(1, \|\nabla \ln p_*(x)\|) \,.$$

Although it has exponential terms, they are multiplied with $\bar{\alpha}$, which should be small.

Assumption B.3 (Nonzero smooth light-tailed p_*). There are constants $c_0, h_0 > 0$ such that when $h \in (0, h_0)$, we have

$$\begin{aligned}
&\int \sup_{\|g\|\le h} |p_*(x) + \nabla p_*(x)^\top g - p_*(x+g)| dx \le c_0 h^2, \\
&\int \frac{\sup_{\|g\|\le h} |p_*(x+g) - p_*(x)|^2}{p_*(x)} dx \le c_0 h^2, \\
&\int \|\nabla p_*(x)\| dx \le c_0.
\end{aligned}$$

Assumption B.3 holds, e.g., for Gaussian distributions and mixtures of Gaussians as well as other smooth distributions with light tails. If we assume that $\nabla^2 \ln p_*(x)$ is continuous, then it can be simplified to the following conditions, which are easier to verify.

$$\begin{aligned}
&\int \sup_{\|g\|\le h} \|\nabla^2 \ln p_*(x+g)\| \; p_*(x) dx < \infty, \\
&\int \sup_{\|g\|\le h} \left[e^{2h\|\nabla \ln p_*(x+g)\|} \|\nabla \ln p_*(x+g)\|^2 \right] \; p_*(x) dx < \infty, \\
&\int \|\nabla \ln p_*(x)\| \; p_*(x) dx < \infty.
\end{aligned}$$

Technically, Assumption B.3 can be removed by approximating p_* with a density function of this property (e.g., a Gaussian mixture, which can approximate any density) and then conducting the analysis with this approximating density in place of p_*. But we instead just assume that p_* has this property as it is simpler.

Remark: For Assumption B.1 to hold, if $\alpha = 1$ as in the main paper, p_* and p need to be nonzero everywhere. If we set $\alpha < 1$, we can take $B = \ln(\alpha/\bar{\alpha}) < \infty$ and the nonzero constraints on p and p_* can be relaxed to not being zero at the same time; in this case, since nonzero p_* is assumed in Assumption B.3, we assume p_* to be nonzero instead of p.

To state the assumptions above, we have implicitly assumed that the distributions of interest are absolutely continuous and so permit a density. In modeling data, an advantage of distributions with smooth nonzero densities is that they are easier to deal with than, for example, a distribution with a support contained in a low dimensional manifold [1], [2]. If needed, there are several things that can be done in practice for encouraging the discriminator to behave as assumed:

- Set $\alpha < 1$ (e.g., $\alpha = 0.999$) and update the discriminator for the weighted logistic regression problem above.
- Add a small Gaussian noise to every observed data point, as also suggested by [1].
- Regularize the discriminator with the gradient penalty as suggested by [38], [26] to approximate the effect of adding noise.

However, interestingly, our experiments on image generation indicate that we do not have to do any of these things to achieve stable training to obtain good performances rivaling the state of the art methods. This is a good thing as we do not have to adjust α or the noise amount or the degree of regularization or incur the overhead of regularization.

B.2.3 Weighted version of discrete theorem

Theorem B.1. Under the definitions and assumptions above, let $g : \mathbb{R}^k \to \mathbb{R}^k$ be a continuously differentiable transformation such that $\|g(\cdot)\| \leq a$ and $\|\nabla g(\cdot)\| \leq b$. Let p and p' be the probability densities of a random variable X and X', respectively, such that $X' = X + \eta g(X)$ where $0 < \eta < \min(1/b, h_0/a)$. Then there exists a positive constant c such that:

$$L_\alpha(p') \leq L_\alpha(p) - \eta \int p_*(x)(\alpha - \bar{\alpha}\exp(D_\alpha(x)))\, g(x)^\top \nabla D_\alpha(x)\, dx + c\eta^2 + c\eta\epsilon.$$

B.3 Proof of Theorem B.1

Note that we continue to use the notation given at the beginning of Appendix A. We also need the following lemmas.

Lemma B.1. Assume that $g(x) : \mathbb{R}^k \to \mathbb{R}^k$ is a continuously differentiable transformation. Assume that $\|g(x)\| \leq a$ and $\|\nabla g(x)\| \leq b$, then when $\eta b < 1$, the inverse transformation $x = h^{-1}(x')$ of $x' = h(x) = x + \eta g(x)$ is unique.
Moreover, consider transformation of random variables by $h^{-1}(\cdot)$. Define $\tilde{p}_*$ to be the associated probability density function after this transformation when the pdf before the transformation is p_*. Then for any $x \in \mathbb{R}^k$, we have:

$$\tilde{p}_*(x) = p_*(h(x))|\det(\nabla h(x))|. \tag{30}$$

Similarly, we have

$$p(x) = p'(h(x))|\det(\nabla h(x))|, \tag{31}$$

where p and p' are defined in Theorem B.1.

Proof Given x', define map $g'(x)$ as $g'(x) = x' - \eta g(x)$, then the assumption implies that $g'(x)$ is a contraction when $\eta b < 1$: $\|g'(x) - g'(x')\| \leq \eta b\|x - x'\|$. Therefore $g'(x)$ has a unique fixed point x, which leads to the inverse transformation $h^{-1}(x') = x$.
(30) and (31) follow from the standard density formula under transformation of variables. ■

Lemma B.2. Under the assumptions of Lemma B.1, there exists a constant $c > 0$ such that

$$|\det(\nabla h(x)) - (1 + \eta\nabla \cdot g(x))| \leq c\eta^2. \tag{32}$$

Proof
We note that

$$\nabla h(x) = I + \eta\nabla g(x).$$

Therefore

$$\det(\nabla h(x)) = 1 + \eta\nabla \cdot g(x) + \sum_{j\geq 2} \eta^j m_j(g(x)),$$

where $m_j(g)$ is a function of ∇g. Since ∇g is bounded, we obtain the desired formula. ■

Lemma B.3. Under the assumptions of Lemma B.1, and assume that Assumption B.3 holds, then there exists a constant $c > 0$ such that

$$\int |\tilde{p}_*(x) - (p_*(x) + \eta p_*(x)\nabla \cdot g(x) + \eta\nabla p_*(x)^\top g(x))|dx \leq c\eta^2. \tag{33}$$

and

$$\int \frac{(\tilde{p}_*(x) - p_*(x))^2}{p_*(x)} dx \le c\eta^2. \tag{34}$$

Proof Using the algebraic inequality

$$\begin{aligned}
&\big|p_*(h(x))|\det(\nabla h(x))| - (p_*(x) + \eta p_*(x)\nabla\cdot g(x) + \eta\nabla p_*(x)^\top g(x))\big| \\
\le&\big|p_*(h(x)) - (p_*(x) + \eta\nabla p_*(x)^\top g(x))\big|\,\big|\det(\nabla h(x))\big| \\
&+ \big|(p_*(x) + \eta\nabla p_*(x)^\top g(x))\big|\,\big|(1+\eta\nabla\cdot g(x)) - |\det(\nabla h(x))|\big| \\
&+ \eta^2\big|\nabla\cdot g(x)\ \nabla p_*(x)^\top g(x))\big|,
\end{aligned}$$

and using $\tilde{p}_*(x) = p_*(h(x))|\det(\nabla h(x))|$ from (30), we obtain

$$\begin{aligned}
&\int \big|\tilde{p}_*(x) - (p_*(x) + \eta p_*(x)\nabla\cdot g(x) + \eta\nabla p_*(x)^\top g(x))\big| dx \\
\le&\underbrace{\int \big|p_*(h(x)) - (p_*(x) + \eta\nabla p_*(x)^\top g(x))\big|\,|\det(\nabla h(x))| dx}_{A_0} \\
&+ \underbrace{\int \big|(p_*(x) + \eta\nabla p_*(x)^\top g(x))\big|\,\big|(1+\eta\nabla\cdot g(x)) - |\det(\nabla h(x))|\big|\,dx}_{B_0} \\
&+ \eta^2\underbrace{\int |\nabla\cdot g(x)\ \nabla p_*(x)^\top g(x))| dx}_{C_0} \\
\le& c\eta^2
\end{aligned}$$

for some constant $c > 0$, which proves (33). The last inequality uses the following facts.

$$A_0 = \int \big|p_*(h(x)) - (p_*(x) + \eta\nabla p_*(x)^\top g(x))\big|\, O(1) dx = O(\eta^2),$$

where the first equality follows from the boundedness of g and ∇g, and the second equality follows from the first inequality of Assumption B.3.

$$B_0 = \int \big|(p_*(x) + \eta\nabla p_*(x)^\top g(x))\big|\, O(\eta^2)\, dx = O(\eta^2),$$

where the first equality follows from (32), and the second equality follows from the third equality of Assumption B.3.

$$C_0 = \int \|\nabla p_*(x)\| O(1) dx = O(1),$$

where the first equality follows from the boundedness of g and ∇g, and the second equality follows from the third equality of Assumption B.3.

Moreover, using (30), we obtain

$$|\tilde{p}_*(x) - p_*(x)| \le |p_*(h(x)) - p_*(x)|\,|\det(\nabla h(x))| + p_*(x)||\det(\nabla h(x))| - 1|.$$

Therefore

$$\begin{aligned}
&\int \frac{(\tilde{p}_*(x) - p_*(x))^2}{p_*(x)} dx \\
\le& 2\int \frac{(p_*(h(x)) - p_*(x))^2|\det(\nabla h(x))|^2 + p_*(x)^2(|\det(\nabla h(x))| - 1)^2}{p_*(x)} dx \le c\eta^2
\end{aligned}$$

for some $c > 0$, which proves (34). The second inequality follows from the second inequality of Assumption B.3, and the boundedness of $|\det(\nabla h(x))|$, and the fact that $||\det(\nabla h(x))| - 1| = O(\eta)$ from (32). ■

Proof of Theorem B.1

In the following integration, with a change of variable from x to x' using $x' = h(x)$ as in Lemma B.1, we obtain

$$\begin{aligned}
&\int (\alpha p_*(x') + \bar{\alpha} p'(x')) \ln \left(\frac{\alpha p_*(x') + \bar{\alpha} p'(x')}{\bar{\alpha} p_*(x') + \alpha p'(x')} \right) dx' \\
=&\int (\alpha p_*(h(x)) + \bar{\alpha} p'(h(x))) \ln \left(\frac{\alpha p_*(h(x)) + \bar{\alpha} p'(h(x))}{\bar{\alpha} p_*(h(x)) + \alpha p'(h(x))} \right) |\det(\nabla h(x))| dx \\
=&\int (\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)) \ln \frac{\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} \tilde{p}_*(x) + \alpha p(x)} dx,
\end{aligned}$$

where the first equality is basic calculus, and the second equality uses (30) and (31).

It follows that

$$\begin{aligned}
L_\alpha(p') =& \int (\alpha p_*(x') + \bar{\alpha} p'(x')) \ln \frac{\alpha p_*(x') + \bar{\alpha} p'(x')}{\bar{\alpha} p_*(x') + \alpha p'(x')} dx' \\
=& \int (\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)) \ln \frac{\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} \tilde{p}_*(x) + \alpha p(x)} dx \\
=& A_1 + B_1 + C_1,
\end{aligned}$$

where A_1, B_1, and C_1 are defined as follows.

$$\begin{aligned}
A_1 =& \int (\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)) \ln \frac{\alpha p_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} p_*(x) + \alpha p(x)} dx \\
=& \int (\alpha p_*(x) + \bar{\alpha} p(x)) \ln \frac{\alpha p_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} p_*(x) + \alpha p(x)} dx \\
&+ \eta\alpha \int (p_*(x) \nabla \cdot g(x) + \nabla p_*(x)^\top g(x)) \ln \frac{\alpha p_*(x) + \bar{\alpha} p(x)}{\bar{\alpha} p_*(x) + \alpha p(x)} dx + O(\eta^2) \\
=& L_\alpha(p) + \alpha\eta \int \nabla \cdot (p_*(x) g(x)) \, \mathcal{D}_\alpha(x) dx + O(\eta^2) \\
=& L_\alpha(p) + \alpha\eta \int \nabla \cdot (p_*(x) g(x)) \, D_\alpha(x) dx + O(\eta\epsilon + \eta^2) \\
=& L_\alpha(p) - \alpha\eta \int p_*(x) g(x)^\top \nabla D_\alpha(x) dx + O(\eta\epsilon + \eta^2),
\end{aligned}$$

where the second equality uses (33) and the assumption that $B < \infty$ (Assumption B.1) The fourth equality uses the ϵ-approximation condition (Assumption B.2). The last equality uses integration by parts and (21).

$$\begin{aligned}
B_1 =& \int (\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)) \ln \frac{\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)}{\alpha p_*(x) + \bar{\alpha} p(x)} dx \\
=& \int (\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)) \ln \left(1 + \alpha \frac{\tilde{p}_*(x) - p_*(x)}{\alpha p_*(x) + \bar{\alpha} p(x)} \right) dx \\
\leq& \alpha \int (\alpha \tilde{p}_*(x) + \bar{\alpha} p(x)) \frac{\tilde{p}_*(x) - p_*(x)}{\alpha p_*(x) + \bar{\alpha} p(x)} dx \\
=& \alpha \int (\alpha \tilde{p}_*(x) - \alpha p_*(x)) \frac{\tilde{p}_*(x) - p_*(x)}{\alpha p_*(x) + \bar{\alpha} p(x)} + \underbrace{\alpha \int (\alpha p_*(x) + \bar{\alpha} p(x)) \frac{\tilde{p}_*(x) - p_*(x)}{\alpha p_*(x) + \bar{\alpha} p(x)} \, dx}_{0} \\
=& \alpha^2 \int \frac{(\tilde{p}_*(x) - p_*(x))^2}{\alpha p_*(x) + \bar{\alpha} p(x)} dx = O(\eta^2),
\end{aligned}$$

where the inequality uses $\ln(1 + \delta) \leq \delta$. The second to the last inequality uses the fact that $\int \tilde{p}_*(x) dx = \int p_*(x) dx = 1$. The last equality uses (34).

$$
\begin{aligned}
C_1 =& \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \ln \frac{\bar{\alpha}p_*(x) + \alpha p(x)}{\bar{\alpha}\tilde{p}_*(x) + \alpha p(x)} dx \\
=& \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \ln \left(1 + \bar{\alpha}\frac{p_*(x) - \tilde{p}_*(x)}{\bar{\alpha}\tilde{p}_*(x) + \alpha p(x)}\right) dx \\
\leq& \bar{\alpha} \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \frac{p_*(x) - \tilde{p}_*(x)}{\bar{\alpha}\tilde{p}_*(x) + \alpha p(x)} dx \\
=& \bar{\alpha} \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \frac{p_*(x) - \tilde{p}_*(x)}{(\bar{\alpha}\tilde{p}_*(x) + \alpha p(x))(\bar{\alpha}p_*(x) + \alpha p(x))} (\bar{\alpha}p_*(x) + \alpha p(x)) dx \\
=& \bar{\alpha} \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \frac{p_*(x) - \tilde{p}_*(x)}{(\bar{\alpha}\tilde{p}_*(x) + \alpha p(x))(\bar{\alpha}p_*(x) + \alpha p(x))} \left((\bar{\alpha}\tilde{p}_*(x) + \alpha p(x)) + \bar{\alpha}(p_*(x) - \tilde{p}_*(x))\right) dx \\
=& \bar{\alpha} \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \frac{p_*(x) - \tilde{p}_*(x)}{\bar{\alpha}p_*(x) + \alpha p(x)} dx + \bar{\alpha}^2 \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \frac{(p_*(x) - \tilde{p}_*(x))^2}{(\bar{\alpha}\tilde{p}_*(x) + \alpha p(x))(\bar{\alpha}p_*(x) + \alpha p(x))} dx \\
\underset{(a)}{=}& \bar{\alpha} \int (\alpha\tilde{p}_*(x) + \bar{\alpha}p(x)) \frac{p_*(x) - \tilde{p}_*(x)}{\bar{\alpha}p_*(x) + \alpha p(x)} dx + O(\eta^2) \\
\underset{(b)}{=}& -\bar{\alpha} \int (\alpha p_*(x) + \bar{\alpha}p(x)) \frac{\eta p_*(x)\nabla \cdot g(x) + \eta \nabla p_*(x)^\top g(x)}{\bar{\alpha}p_*(x) + \alpha p(x)} dx + O(\eta^2) \\
=& -\bar{\alpha}\eta \int (p_*(x)\nabla \cdot g(x) + \nabla p_*(x)^\top g(x)) \exp(\mathcal{D}_\alpha(x)) dx + O(\eta^2) \\
\underset{(c)}{=}& -\bar{\alpha}\eta \int (p_*(x)\nabla \cdot g(x) + \nabla p_*(x)^\top g(x)) \exp(D_\alpha(x)) dx + O(\eta\epsilon + \eta^2) \\
=& -\bar{\alpha}\eta \int (\nabla \cdot (p_*(x) g(x))) \exp(D_\alpha(x)) dx + O(\eta\epsilon + \eta^2) \\
=& \eta\bar{\alpha} \int p_*(x) g(x)^\top \nabla \exp(D_\alpha(x)) dx + O(\eta\epsilon + \eta^2),
\end{aligned}
$$

where the first inequality uses $\ln(1+\delta) \leq \delta$. The equality (a) uses (34). The equality (b) uses (33). The equality (c) uses the ϵ-approximation condition (Assumption B.2) The last equality uses integration by parts and (21).

By combining the estimates of A_1, B_1, and C_1, we obtain the desired bound.

B.4 Other f-divergences

In Theorem B.1 (and Theorem 3.1, a special case of Theorem B.1), the distance measure is specifically the KL divergence. It is possible to obtain similar results on other f-divergences; however, doing so could require rather complex regularity conditions. For the KL divergence, $f(\gamma) = -\ln(\gamma)$, and so $f(\gamma)$ is convex while $f(1/\gamma) = \ln(\gamma)$ is concave, and we can use the inequality $\ln\gamma \leq \gamma - 1$ to drop the second-order terms in the upper bounds of quantities B_1 and C_1 in the proof above. This simplifies the regularity conditions on the shape of p_* as given in Assumption B.3.

B.5 Convergences

Lemma B.4. As in Sections 2 and 3, let $\phi : \mathbb{R}^k \to \mathbb{R}^k$ be a vector function that satisfies $\phi(u) = u \cdot \phi_0(u) \geq 0$ with some $\phi_0 : \mathbb{R}^k \to \mathbb{R}^k$ and $\phi(u) = 0$ if and only if $u = 0$. Suppose that we have a function $q_t : \mathbb{R}^k \to \mathbb{R}$ such that $q_t(x) \geq 0$ and a vector function $h_t : \mathbb{R}^k \to \mathbb{R}^k$, and that $q_t(x)\phi(h_t(x))$ is continuous. If we have

$$
\lim_{t\to\infty} \int q_t(x)\phi(h_t(x)) dx = 0 \,, \tag{35}
$$

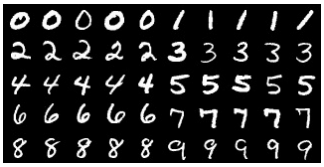
then we have $\lim_{t\to\infty} q_t(x)h_t(x) = 0$ pointwise everywhere.

Proof (35) and $q_t(x)\phi(h_t(x)) \geq 0$ implies $\lim_{t\to\infty} q_t(x)\phi(h_t(x)) = 0$ pointwise almost everywhere. The continuity condition further eliminates the possibility of $q_t(x)\phi(h_t(x))$ going to nonzero in sets of measure 0, and so we obtain $\lim_{t\to\infty} q_t(x)\phi(h_t(x)) = 0$ pointwise everywhere. Since $\phi(h_t(x)) = 0$ iff $h_t(x) = 0$, this implies $\lim_{t\to\infty} q_t(x)h_t(x) = 0$ pointwise everywhere, as desired. ■

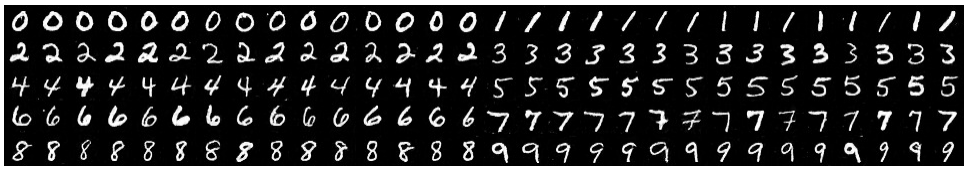
To obtain (10) $\lim_{t\to\infty} p_t(x)\nabla_x \ell_2'(p_*(x), p_t(x)) = 0$ in the continuous analysis, apply Lemma B.4 to $\lim_{t\to\infty} \int s_t(x)p_t(x)\phi(\nabla_x \ell_2'(p_*(x), p_t(x)))dx = 0$ with $q_t(x) = s_t(x)p_t(x)$ and $h_t(x) = \nabla_x \ell_2'(p_*(x), p_t(x))$.

To obtain $\lim_{t\to\infty} p_*(x)\nabla D_t(x) = 0$ at the end of Section 3 in the discrete analysis, apply Lemma B.4 to $\lim_{t\to\infty} \int p_*(x)s_t(x)\|\nabla D_t(x)\|_2^2 dx = 0$ with $q_t(x) = p_*(x)s_t(x)$, $h_t(x) = \nabla D_t(x)$, $\phi(u) = \|u\|_2^2$, and $\phi_0(u) = u$.

In both, further use the fact that the limit holds for an arbitrary function $s_t(x)$ such that $s_t(x) > 0$, which includes the one that satisfies $s_t(x) \geq c$ for some positive constant c (thus not converging to 0).

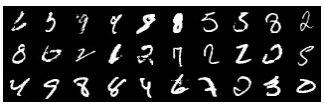


(a) Real images

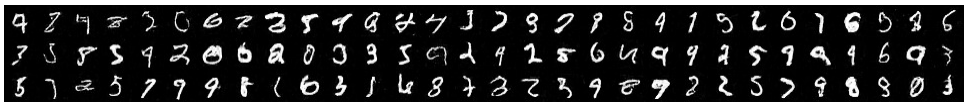


(b) Generated by xICFG (convolutional)

Fig. 16: 'Best' digits. MNIST. For each digit, showing images with the highest probabilities among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG.



(a) Real images



(b) Generated by xICFG (convolutional)

Fig. 17: 'Worst' digits. MNIST. Images with the highest entropy among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG. In some of the generated images in (b), it is hard to tell what the digits are, but that is also true in some of the real images in (a).

(a) Real images

(b) Generated by xICFG (convolutional)

Fig. 18: 'Best' digits. SVHN. For each digit, showing images with the highest probabilities among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG.

(a) Real images

(b) Generated by xICFG (convolutional)

Fig. 19: 'Worst' digits. SVHN. Images with the highest entropy among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG. In some of the generated images in (b), it is hard to tell what the digits are, but that is also true in some of the real images in (a).

APPENDIX C
SUPPLEMENTS TO SECTION 5 (EXPERIMENTS)

C.1 Image examples

As was done for the LSUN images in the main paper, we take a focused approach to show example images. We generate 1000 images by xICFG trained with the convolutional networks as in Fig. 2 and show the 'best' and 'worst' images among them. Similar to the inception score, the 'goodness' of images are measured by the confidence of a classifier, e.g., a image that a classifier assigns a high probability of being a "bedroom" is considered to be a good bedroom image. The 'worst' images are those with the highest entropy values. They are the best and worst contributors to the inception score, respectively, as well. In Figures 16–25, we compare real images and generated images side by side that were chosen by the same procedure from a random sample of 1000 real images or 1000 generated images (generated from one sequence of random inputs), respectively.

(a) Real images.

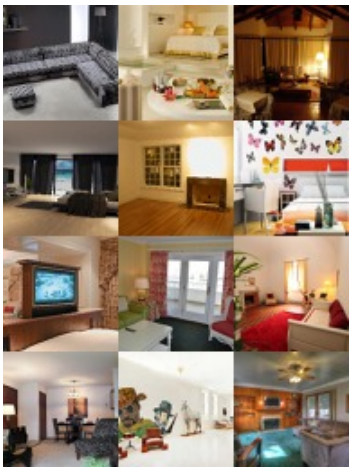
(b) Generated by xICFG (4-block ResNet)

Fig. 20: Bedrooms 'best' among 1000 (LSUN BR+LR). Predicted by a classifier to be "bedroom" with the highest probabilities among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG.

(a) Real images.

(b) Generated by xICFG (4-block ResNet)

Fig. 21: Living rooms 'best' among 1000 (LSUN BR+LR). Predicted by a classifier to be "living room" with the highest probabilities among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG.



(a) Real images.

(b) Generated by xICFG (4-block ResNet)

Fig. 22: Bedrooms/living rooms 'worst' among 1000 (LSUN BR+LR). Images with the highest entropy among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG. The generated images in (b) could be either of relatively low quality or depicting hard-to-classifiy rooms as the real images in (a) are.

(a) Real images.

(b) Generated by xICFG (4-block ResNet)

Fig. 23: Towers 'best' among 1000 (LSUN T+B). Predicted by a classifier to be "tower" with the highest probabilities among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG.

(a) Real images.

(b) Generated by xICFG (4-block ResNet)

Fig. 24: Bridges 'best' among 1000 (LSUN T+B). Predicted by a classifier to be "bridge" with the highest probabilities among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG.

(a) Real images.

(b) Generated by xICFG (4-block ResNet)

Fig. 25: Towers/bridges 'worst' among 1000 (LSUN T+B). Images with the highest entropy among 1000 images that were either (a) randomly chosen from real data or (b) generated by xICFG. The generated images in (b) could be either of relatively low quality or depicting hard-to-classify objects as the real images in (a) are.

C.2 Details of the experiments

C.2.1 Network architectures

MNIST and SVHN in Figure 2: The convolutional architectures used for MNIST and SVHN are an extension of DCGAN, inserting 1×1 convolution layers.

Approximator/Generator

1	Projection
2	ReLU
3	Transposed conv, 5×5, stride 2
4	BatchNorm
5	ReLU
6	Convolution, 1×1, stride 1
7	BatchNorm
8	ReLU
9	Transposed conv, 5×5, stride 2
10	$\tanh$

Repeat 2–7 twice.

Discriminator

1	Convolution, 5×5, stride 2
2	LeakyReLU
3	Convolution, 5×5, stride 2
4	BatchNorm
5	LeakyReLU
6	Convolution, 1×1, stride 1
7	BatchNorm
8	LeakyReLU
9	Flatten
10	Linear

Repeat 3–8 twice.

For the discriminator, start with 32 (MNIST) or 64 (SVHN) feature maps and double it at downsampling except for the first downsampling. For the approximator/generator, start with 128 (MNIST) or 256 (SVHN) and halve it at upsampling except for the last upsampling.

LSUN in Figure 2: The convolutional architecture used for LSUN is a simplification of a residual network found in the WGANgp code release, reducing the number of batch normalization layers for speedup, removing some irregularity, and so forth. Both the approximator/generator and discriminator are a residual network with four convolution blocks.

Approximator/Generator

1	Projection
2	ReLU
3	Upsampling (×2), nearest
4	Convolution, 3×3, stride 1
5	ReLU
6	Convolution, 3×3, stride 1
7	BatchNorm
8	ReLU
9	Convolution, 3×3, stride 1
10	$\tanh$

Repeat 2–7 four times.

Discriminator

1	ReLU (omitted in the 1st block)
2	Convolution, 3×3, stride 1
3	ReLU
4	Convolution, 3×3, stride 1
5	BatchNorm
6	Downsampling (/2), mean
7	ReLU
8	Flatten
9	Linear

Repeat 1–6 four times.

2–7 of the approximator/generator and 1–6 of the discriminator are convolution blocks with a shortcut connecting the beginning to the end. In the approximator/generator, the numbers of feature maps are 512 (produced by the projection layer) and 256, 256, 128, 128, 64, 64, 64, and 64 (produced by the convolution layers). In the discriminator, the numbers of feature maps produced by the convolution layers are 64, 64, 64, 128, 128, 256, 256, and 512.

C.2.2 Details of experimental settings for xICFG

The network weights were initialized by the Gaussian with mean 0 and standard deviation 0.01.

The rmsprop learning rate for xICFG (for updating the discriminator and the approximator) was fixed to 0.0001 across all the datasets when the approximator was fully-connected. Although 0.0001 worked well also for the convolutional approximator cases, these cases turned out to tolerate and benefit from a more aggressive learning rate resulting in faster training; hence, it was fixed to 0.00025 across all the datasets. Additionally, if we keep training long enough, the discriminator may eventually overfit as also noted on WGANgp in [11], or it may go to the overtrained state. It may be useful to reduce the learning rate (e.g., by multiplying 0.1) towards the end of training if the onset of discriminator overfit/overtraining needs to be delayed. Another option is to increase N and/or U as discussed in Section 5.2.4.

To choose η used for generator update $G_{t+1}(z) = G_t(z) + \eta \nabla D(G_t(z))$, we tried some of $\{0.1, 0.25, 0.5, 1, 2.5\}$ (not all as we tried only promising ones) for each configuration, following the meta-parameter selection protocol described in Section 5.1. Typically, multiple values were found to work well, and the table below shows the values used for the reported results.

	MNIST	SVHN	BR+LR	T+B
convolutional (Fig.2)	0.5	0.25	1	1
conv. no batch norm (Fig.3)	2.5	0.5	2.5	2.5
fully-connected (Fig.4)	0.1	0.25	0.5	0.5

In general, similar to the SGD learning rate, a larger η leads to faster training, but a too large value would break training. A too small value should be avoided since stable training requires a generator to make sufficient progress before each approximator update.

In our experiments, we fixed the scaling factor $s_t(x)$ to 1. It may be worth exploring methods of dynamically changing it with time t and/or data x for performance improvement or speeding up training.

C.2.3 Δ_D and the KL divergence

In Section 5.2.3, we show that the quality of an xICFG generator correlates to '$|D(\text{real})-D(\text{gen})|$' ($\Delta_D$ in short), which is the difference between the discriminator output values for real images and generated images averaged over time intervals of a fixed length, obtained as a by-product of the forward propagation for updating the discriminator. When D is trained for the logistic regression objective using real data and generated data as positive and negative examples, we have $D(x) \approx \ln(p_*(x)/p(x))$ where p_* and p are the probability density functions of the real data and generated data, respectively. Therefore, we have

$$\mathbb{E}_{x\sim p_*} D(x) - \mathbb{E}_{x\sim p} D(x) \approx \int p_*(x) \ln \frac{p_*(x)}{p(x)} dx + \int p(x) \ln \frac{p(x)}{p_*(x)} dx \ . \tag{36}$$

Δ_D is *related* to the left-hand side of (36), and the right-hand side of (36) is the sum of the KL divergence and the reverse KL divergence between the two disctirubtions, which we have shown in Section 2 that the CFG algorithm optimizes.

However, Δ_D is not a good estimate of this quantity, and in particular, it should not be used for evaluation. To make a good estimate of (36), D should be trained in a way independent of training; otherwise, the estimate would be influenced by the training settings such as meta-parameters. For the same reason, Δ_D is not comparable across training runs, but empirically, it is useful when compared within a run and it can be obtained at almost no cost during training.

C.2.4 Details of the experiments in Section 5.2.5

The experiments with spectral normalization (GAN-sn) [29] and the zero-centered gradient penalty (GAN-gp) [26] were done as follows. Following the original studies, the training objective was set to the one with the log$d$ trick. Batch normalization was removed from the discriminator also following the original work (we tested several cases and found that indeed better results were obtained by doing so). The regularization parameter for GAN-gp was set to 10 as in the original study. Optimization was done with rmsprop as was done for xICFG and the original GANs. We tuned the learning rate and U (the discriminator update frequency) choosing from $\{1,5\}$. The network weights were initialized as was done for the other methods (the Gaussian with mean 0 and standard deviation 0.01) for GAN-gp, and for GAN-sn, we chose between this default method and the method used in the GAN-sn code release by the authors of the original work, which is the uniform Glorot initialization [9] with an additional scaling factor per layer. We also tried spectral normalization in the generator as it was useful in [47], [4], but it did not help in our cases. We conjecture that this is because of the differences in the network architecture and the task setting (unconditional vs. class-conditional). Tuning was done according to the tuning protocol in Section 5.1.6. For xICFG with convolutional networks on MNIST, we set U=10 and N=100b as this setting was found slightly better in Section 5.2.4, and the default (U=1,N=10b) was used for the rest. For all the methods, to perform 3 runs using 3 different random seeds, we generally did tuning using one seed and used the chosen settings for all 3 runs, and only when the outcome deviated a lot from the expected values, did we tuning specifically for that run. The reported results are at the timing of the right end of Fig. 2 and 4 (the figures plotting the scores in relation to training time) except for a few runs where training collapsed before reaching that point; in such cases, the performances before the collapse are reported (early stopping based on the validation performances). It was noted in [4] that better performances were obtained by doing so rather than tuning the setting (e.g., tightening regularization) so that training would never collapse. We followed this tip.

C.2.5 Classifiers used for evaluation

For evaluation, classifiers in the following network architecture were trained on each dataset.

1	convolution, 5×5, stride 1
2	ReLU
3	Max pooling, 3×3, stride 2
4	convolution, 5×5, stride 1
5	BatchNorm
6	ReLU
7	Average pooling, 3×3, stride 2
8	Flatten
9	Linear

Repeat 4–7 three times (LSUN)
or twice (MNIST/SVHN).

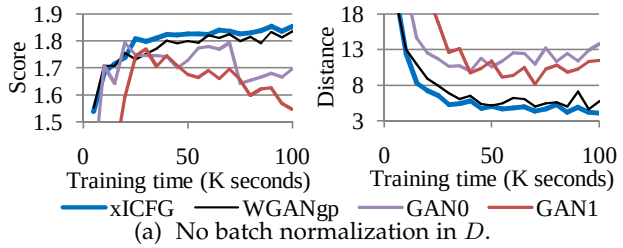
The number of feature maps produced by convolution layers was 64 at the beginning and doubled at every downsampling. The training sets (also used for training generative models) were disjoint from the held-out sets that were used for evaluating the Fréchet distance and estimating the inception score upper bounds shown in Section 5.2.1.

C.3 Additional experiments on CelebA

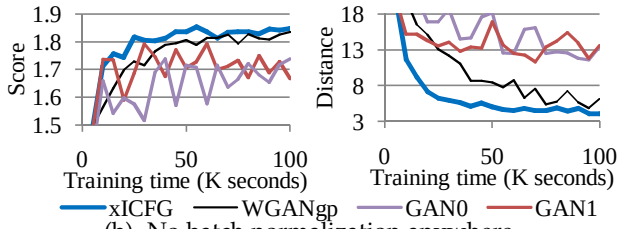
This section provides additional experimental results on the face image dataset CelebA. All the images were shrunk and center-cropped into 128×128, larger than those in the main paper. We held out 10K images and used the remaining 193K images for training. For evaluation, we trained a male/female classifier; the architecture was as in Appendix C.2.5, with 4–7 repeated four times and not doubling the number of feature maps for the last convolution layer.

Figure 26 shows the inception score and Fréchet distance results, obtained by using the DCGAN extension (as in C.2.1[4]) with (a) removing batch normalization from the discriminator (not only for WGANgp but also for all), and (b) removing batch normalization from both the discriminator and approximator/generator. With xICFG and WGANgp, both the inception score and Fréchet distance generally improve as training proceeds. By contrast, the performance of GAN0 and GAN1 is somewhat erratic. These results are consistent with the results on the other datasets in the main paper.

Using the same procedure as C.1, Figures 27–29 show examples of 'best' and 'worst' images from real data, images generated by xICFG and the best-performing baseline (WGANgp). The 'best' images are those which were assigned by the male/female classifier the highest probability that they are men/women among a random sample of 1000 images. The 'worst' images are those with the highest entropy among the same random sample of 1000 images used for choosing the 'best' images. The 'best' images generated by xICFG (Fig. 27b and 28b) are of high quality even though they are not perfect when compared with real images in detail. The real images with the highest entropy (Fig. 29a) appear to be outliers in terms of age (see the child and the older lady), the presence of objects around the face (see the sunglasses and the hat), the angle of the face, and so forth. They are diverse and make a good contrast to the 'best' real men and women (Fig. 27a and 28a), which look like typical celebrity face images. Accordingly, the highest-entropy images generated by xICFG (Fig. 29b) are diverse, and also, some of them are of low quality. Overall, we feel that visual impression of the images generated by xICFG is as good as, if not better than, that of WGANgp images.



(a) No batch normalization in D.



(b) No batch normalization anywhere.

Fig. 26: CelbA (128×128) Inception score and Fréchet distance results. DCGAN extension.

4. For the discriminator, start with 64 feature maps and double it at downsampling except for the first and last downsampling. For the approximator/generator, start with 512 feature maps and halve it at upsampling except for the first and last upsampling.

(a) Real data (b) xICFG (c) Best baseline (WGANgp)

Fig. 27: 'Best' men with the highest probability to be a man assigned by a classifier among 1000. Celeba (128×128).

(a) Real data (b) xICFG (c) Best baseline (WGANgp)

Fig. 28: 'Best' women with the highest probability to be a woman assigned by a classifier among 1000. Celeba (128×128).

(a) Real data (b) xICFG (c) Best baseline (WGANgp)

Fig. 29: 'Worst' with the highest entropy among a random sample of 1000 real/generated images. Celeba (128×128).