Composite Functional Gradient Learning of Generative Adversarial Models

Rie Johnson [*1] **Tong Zhang** [*2]

Abstract

This paper first presents a theory for generative adversarial methods that does not rely on the traditional minimax formulation. It shows that with a strong discriminator, a good generator can be learned so that the KL divergence between the distributions of real data and generated data improves after each functional gradient step until it converges to zero. Based on the theory, we propose a new stable generative adversarial method. A theoretical insight into the original GAN from this new viewpoint is also provided. The experiments on image generation show the effectiveness of our new method.

1. Introduction

We consider observed real data $x_1^*, \ldots, x_n^* \in \mathbb{R}^r$ from an unknown distribution p_* on $\mathbb{R}^r$. Moreover, assume that we are given a random variable Z with a known distribution such as a Gaussian. We are interested in learning a random variable transformation $G(Z)$ so that the generated data $G(Z)$ has a probability density function that is close to the real distribution p_*. This is the setting considered in *generative adversarial networks* (GAN) (Goodfellow et al., 2014), and the transformation $G(Z)$ is often referred to as a *generator*. While GAN has been widely used, it is also known that GAN is difficult to train due to its instability, which has led to numerous studies, e.g., Wasserstein GAN (WGAN) and its extensions to pursue a different minimax objective (Arjovsky et al., 2017; Gulrajani et al., 2017; Mroueh & Sercu, 2017), mode-regularized GAN to tackle the issue of mode collapse (Che et al., 2017), unrolled GAN (Metz et al., 2017), AdaGAN (Tolstikhin et al., 2017), MMD GAN (Li et al., 2017), 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. Mathematically, GAN solves the following minimax optimization problem:

$$\max_d \min_G \left[\sum_{x^* \in \text{real data}} \ln d(x^*) + \sum_{G(z) \in \text{fake data}} \ln(1 - d(G(z))) \right]. \quad (1)$$

Parameterizing d and G, (1) can be viewed as a saddle point problem in optimization, which can be solved using a stochastic gradient method, where one takes a gradient step with respect to the parameters in d and G (see Algorithm 4 below). However, the practical procedure, suggested by the original work (Goodfellow et al., 2014), replaces minimization of $\log(1 - d(G(z)))$ with respect to G in (1) with maximization of $\log(d(G(z)))$ with respect to G, called the *log$d$ trick*. Thus, 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 presents a new theory for generative adversarial methods which does not rely on the minimax formulation (1). Our theory shows that one can learn a good generator $G(Z)$ where 'goodness' is measured by the KL-divergence between the distributions of real data and generated data, by using *functional gradient learning* greedily, similar to gradient boosting (Friedman, 2001). However, unlike the standard gradient boosting, which uses additive models, we consider functional compositions in the following form

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

to obtain $G(Z) = G_T(Z)$. Here η_t is a small learning rate, and each g_t is a function to be estimated from data. An initial generator $G_0(Z) \in \mathbb{R}^r$ 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 KL-divergence) is guaranteed.

Our theory leads to a new stable generative adversarial method. It also provides a new theoretical insight into the original GAN both with and without the log$d$ trick. The experiments show the effectiveness of our new method on image generation in comparison with GAN variants.

Notation Throughout the paper, we use x to denote data in $\mathbb{R}^r$, and in particular, we use x^* to denote real data. The probability density function of real data is denoted by p_*. We use $\|\cdot\|$ to denote the vector 2-norm and $\nabla h(x)$ to denote the gradient w.r.t. x of a scalar function $h(x)$.

[*]Equal contribution [1]RJ Research Consulting, Tarrytown, NY, USA [2]Tencent AI Lab, Shenzhen, China. Correspondence to: Rie Johnson <riejohnson@gmail.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

2. Theory

To present our theory, we start with stating assumptions. We then analyze one step of random variable transformation in (2) (i.e., transforming $G_{t-1}(Z)$ to $G_t(Z)$) and examine an algorithm suggested by this analysis.

2.1. Assumptions and definitions

A strong discriminator Given a set S_* of real data and a set S of generated data, assume that we can obtain a strong *discriminator* D using logistic regression so that $D \approx \operatorname{argmin}_{D'} \left[\frac{1}{|S_*|} \sum_{x \in S_*} \ln(1+e^{-D'(x)}) + \frac{1}{|S|} \sum_{x \in S} \ln(1+e^{D'(x)}) \right]$. D tries to discriminate the real data from the generated data. Here we use the logistic model, and so d in (1) corresponds to $\frac{1}{1+\exp(-D(x))}$. Define a quantity $\mathcal{D}(x)$ by

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

where p_* and p are the probability density functions of real data and generated data, respectively, and assume that $|\mathcal{D}(x)|<\infty$. p_* and p are thus assumed to be nonzero everywhere. When the number of given examples is sufficiently large, the standard statistical consistency theory of logistic regression (see e.g., (Zhang, 2004)) implies that $D(x) \approx \mathcal{D}(x)$. Therefore, assume that the following ϵ-approximation condition is satisfied for a small $\epsilon > 0$:

$$\int q_*(x) \left(|D(x) - \mathcal{D}(x)| + \left| e^{D(x)} - e^{\mathcal{D}(x)} \right| \right) dx \leq \epsilon,$$
$$q_*(x) = p_*(x) \max(1, \|\nabla \ln p_*(x)\|) .$$

Note that the assumption of the *optimal* discriminator has been commonly used, and we slightly relax it to a *strong* discriminator by quantifying the deviation from the optimum by ϵ.

Smooth and light-tailed p_* Assume that p_*, the density of real data, is smooth with light tails; we use a constant $h_0 > 0$ that depends on the shape of p_*. Due to the space limit, the precise statements are deferred to the Appendix. Common exponential distributions such as Gaussian distributions and mixtures of Gaussians all satisfy the assumption, and an arbitrary distribution can always be approximated to an arbitrary precision by a mixture of Gaussians.

2.2. Analyzing one step of random variable transformation

The goal is to approximate the true density p_* on $\mathbb{R}^r$ through (2). Our analysis here focuses on one step of (2) 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 $\mathbb{R}^r$. We are interested in finding a function $g : \mathbb{R}^r \to \mathbb{R}^r$, so that the transformed variable $X' = X + \eta g(X)$ for small $\eta > 0$ has a distribution closer to p_*. We show that this can be achieved with a gradient-like step in the function space.

To measure the distance of a density p from the true density p_*, we will keep track of the KL-divergence

$$L(p) = \int p_*(x) \ln \frac{p_*(x)}{p(x)} dx \tag{3}$$

before and after the transformation. We know that $L(p) \geq 0$ for all p, and $L(p) = 0$ if and only if $p = p_*$.

The following theorem consequently shows that with an appropriately chosen $g(\cdot)$, the transformation $X \to X + \eta g(X)$ can always reduce the KL-divergence $L(\cdot)$. This means that transformation $X + \eta g(X)$ is an improvement from X. The proof is given in the Appendix.

Theorem 2.1 *Under the assumptions in Section 2.1, let $g : \mathbb{R}^r \to \mathbb{R}^r$ be a continuously differentiable transformation such that $\|g(\cdot)\| \leq a$ and $\|\nabla g(\cdot)\| \leq b$. Let p be the probability density of a random variable X, and let p' be the probability density of the random variable X' 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 for all $\epsilon > 0$:*

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

The consequences of the theorem become clear when we choose $g(x) = s(x)\nabla D(x)$ (where $s(x) > 0$ is an arbitrary scaling factor). By doing so and letting $\epsilon = \eta$, we have:

$$L(p') \leq L(p) - \eta \int p_*(x) s(x)\ \|\nabla D(x)\|_2^2\ dx + O(\eta^2). \tag{4}$$

This means that by letting $g(x) = s(x)\nabla D(x)$, the objective value $L(\cdot)$ will be reduced for a sufficiently small η unless $\int p_*(x)s(x)\ \|\nabla D(x)\|_2^2\ dx$ vanishes. The vanishing condition implies that $D(x)$ is approximately a constant when p_* has full support on $\mathbb{R}^r$. In this case, the discriminator is unable to differentiate 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 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, our theory leads to 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 true data and the generated data.

If we repeat the process described above, Algorithm 1 is ob-

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 $G_T(z)$

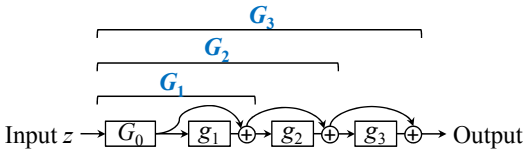


Figure 1. Generator network automatically derived by CFG or ICFG at $t = 3$. '$\oplus$' indicates addition.

tained. We call it *composite functional gradient learning of GAN (CFG-GAN)*, or simply *CFG*. CFG forms g_t by directly using the functional gradient $\nabla D_t(x)$, as suggested by our theory. If (4) holds, and if we choose $\eta_t{=}\eta{=}\epsilon{=}O(1/\sqrt{T})$, then by cascading (4) from $t{=}1$ to T we obtain the following bound: $\frac{1}{T}\sum_{t=1}^T \int p_*(x)s_t(x)\|\nabla D_t(x)\|^2 dx = \frac{1}{T}\left(O(T\eta) + \eta^{-1}\, L(p_0)\right) = O(T^{-1/2})$, where p_0 is the density of $G_0(z)$. This means that as t increases, $\nabla D_t(x) \to 0$ and thus $D_t(x)$ approaches a constant, assuming p_* has full support on $\mathbb{R}^r$. That is, in the limit, the discriminator is unable to differentiate the real data from the generated data.

We note that (Lazarow et al., 2017) describes a related but different cascading process motivated by Langevin dynamics sampling. The Langevin theory requires repeated noise addition in the generation process. Our generation is simpler as there is no need for noise addition.

3. Composite Functional Gradient Algorithms

Starting from the CFG algorithm above, we empirically explored algorithms on image generation. In this section we parameterize D and denote the model parameters by θ_D.

3.1. ICFG: Incremental CFG

The first change made to CFG is to make it incremental similar to GAN, resulting in *incremental CFG* (ICFG, Algorithm 2). CFG (Algorithm 1) optimizes the discriminator to convergence in every iteration. On image generation, this tends to cause the discriminator to overfit, which turned out to be much more harmful than underfitting as an overfit discriminator grossly violates the ϵ-approximation condition

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.

1: **for** $t = 1, 2, \ldots, T$ **do**
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 descending the stochastic gradient: $\nabla_{\theta_D} \frac{1}{b}\sum_{i=1}^b \left[\ln(1{+}e^{-D(x_i^*)}){+}\ln(1{+}e^{D(G_{t-1}(z_i))})\right]$
6: **end for**
7: $g_t(x) \leftarrow s_t(x)\nabla D(x)$ (e.g., $s_t(x) = 1$)
8: $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for $\eta_t > 0$.
9: **end for**
10: **return** generator G_T (and the updated D if so desired).

(Section 2.1) thus grossly pushing the generator in a wrong direction. Also, updating the discriminator to convergence is computationally impractical apart from whether or not doing so is helpful/harmful. 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 overfitting.

Note that here we broadly use the term "overfit" for failure to generalize to unseen data (after fitting to observed data). This includes cases on unseen data in the *low-density region* according to our assumption, which is *outside of manifolds* according to the view of disjoint low-dim data manifolds of (Arjovsky & Bottou, 2017), and so our observation is in line with (Arjovsky & Bottou, 2017) in that the shape of distributions could be problematic. However, we handle the issue differently. Instead of changing loss (leading to WGAN), we deal with it by *early stopping* of discriminator training (like GAN) and functional gradient learning in generator update (unlike GAN). The latter ensures improvement of generator so that it keeps challenging the discriminator, and we will later revisit this point.

ICFG shares nice properties with CFG. The generator is guaranteed to improve with each update to the extent that the assumptions hold; therefore, it is expected to be sta-

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: input pool size, (T, b, U) for ICFG.

1: **loop**
2: $S_z \leftarrow$ an input pool sampled according to p_z.
3: $q_z \leftarrow$ the uniform distribution over S_z.
4: G,$D \leftarrow$ output of ICFG using $S_*, q_z, \widetilde{G}, D$ as input.
5: **if** exit criteria are met **then return** generator G **fi**
6: Update $\widetilde{G}$ to minimize $\sum_{z \in S_z} \frac{1}{2} \|\widetilde{G}(z) - G(z)\|^2$
7: **end loop**

ble. There is no need to design a complex generator model. The generator model is automatically and implicitly derived from the discriminator and *grows* as training proceeds (see Figure 1). A shortcoming, however, is that the implicit generator network can become very large. At time t, computation of $G_t(z)$ starting from scratch is in $O(t)$; therefore, performing T iterations of training could be in $O(\sum_{t=1}^{T} t) = O(T^2)$. We found that image generation requires a large T to the extent that it is computationally problematic, causing slow training and slow generation.

(Nitanda & Suzuki, 2018) recently proposed a related method for fine-tuning WGAN based on different motivations. We note that their method would also suffer from the same issue if used for image generation from scratch.

A partial remedy, which speeds up training (but not generation), is to have an *input pool* of a fixed size. That is, we restrict the input distribution to be on a finite set S_z and for every input $z \in S_z$, maintain $G_{t'}(z)$ for the latest t' for which $G_{t'}(z)$ was computed. By doing so, when $G_t(z)$ needs to be computed, one can start from $G_{t'}(z)$ instead of starting over from $G_0(z)$, which saves computation time. However, this remedy solves only a part of the problem.

3.2. xICFG: Approximate incremental CFG

As a complete 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 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, T for xICFG can be small (e.g., $T = 10$), thus xICFG is efficient. We use the idea of an input pool above for speeding up training, and instead of keeping the same pool to the end, we refresh the pool S_z in every iteration of xICFG (Lines 2&3 of Algorithm 3). For speed, the values $G_{t'}(z)$ for $z \in S_z$ for the latest t' should be kept not only for use in ICFG but also for preparing the training data $\{(z, G(z)) \mid z \in S_z\}$ for the training of the approximator $\widetilde{G}$ (Line 6).

A small pool size $|S_z|$ and a small T would reduce the runtime of one iteration, 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, approximation typically causes some degradation, and so it is important to set T and $|S_z|$ 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 described later.

3.3. Relation to GAN

We show that GAN (Algorithm 4) with the logistic model (as is typically done) is closely related to a special case of xICFG that uses an extreme setting. This viewpoint leads to a new insight into GAN's instability.

We start with the fact that GAN 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 $d(x) = \frac{1}{1+\exp(-D(x))}$ into Line 5 of Algorithm 4.

Next, we show that *the generator update of GAN is equivalent to coarsely approximating a generator produced by ICFG with* T=1. 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

Algorithm 4 GAN (Goodfellow et al., 2014)

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

$[v]_i$ for the i-th component of vector v, the k-th component of this gradient can be written as:

$$[\nabla_{\theta_G} \ln(1 - d(G(z)))]_k = \left[\nabla_{\theta_G} \ln \frac{\exp(-D(G(z)))}{1+\exp(-D(G(z)))}\right]_k = -s_0(G(z)) \sum_j [\nabla D(G(z))]_j \frac{\partial [G(z)]_j}{\partial [\theta_G]_k} \quad (5)$$

where $s_0(x) = \frac{1}{1+\exp(-D(x))}$, 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 $G'(z) = G(z) + \eta g(G(z)) = G(z) + \eta s(G(z)) \nabla D(G(z))$, and then we update G to approximate G' so that $\sum_z \frac{1}{2} \|G'(z) - G(z)\|^2$ is minimized as in Line 6 of xICFG. To take one step of gradient descent for this approximation, we need the gradient $\nabla_{\theta_G} \frac{1}{2} \|G'(z) - G(z)\|^2$, and its k-th component is $-\sum_j [G'(z) - G(z)]_j \frac{\partial [G(z)]_j}{\partial [\theta_G]_k} = -\eta s(G(z)) \sum_j [\nabla D(G(z))]_j \frac{\partial [G(z)]_j}{\partial [\theta_G]_k}$. By setting the scaling factor $s(x) = s_0(x)/\eta$, this is exactly the same as (5), required for the GAN 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:

- Set T=1 so that ICFG updates the generator *just once*.
- To update the approximator, take *only one* gradient descent step with *only one* mini-batch, 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 by replacing $s_0(x) = \frac{1}{1+\exp(-D(x))}$ with $s_1(x) = \frac{1}{1+\exp(D(x))}$. When generated data x is very far from the real data and so $D(x) \ll 0$, we have $s_0(x) \approx 0$ (without the log$d$ trick), which would make the gradient (required for updating the GAN generator) vanish, as noted in (Goodfellow et al., 2014), 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, which is more sensible as well as more similar to our choice ($s(x) = 1$) for the xICFG experiments.

Why is GAN unstable? In spite of their connection, GAN is unstable, and xICFG with appropriate meta-parameters is stable (shown later). Thus, we figure that GAN's instability derives from what is unique to GAN, the two bullets above – *an extremely small* T and *coarse approximation*. Either can cause degradation of the generator, leading to instability.

We have contrasted GAN's generator update with xICFG's *approximator* update. Now we compare it with ICFG's *generator* update 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 *stochastically* and *approximately* updates θ_G using a *small sample* (one mini-batch SGD step backpropagating ∇D), and so GAN's update can be noisy, which can lead to instability through generator degradation.

4. Experiments

We tested xICFG on the image generation task.

4.1. Experimental setup

Baseline methods For comparison, we also tested the following three methods: the original GAN *without* the log$d$ trick (*GAN0* in short), GAN *with* the log$d$ trick (*GAN1*), and WGAN with the gradient penalty (*WGANgp*) (Gulrajani et al., 2017). The choice of GAN0 and GAN1 is due to its relation to xICFG as analyzed above. WGANgp was chosen as a representative of state-of-the-art methods, as it was shown to rival or outperform a number of previous methods such as the original WGAN with weight clipping (Arjovsky et al., 2017), Least Squares GAN (Mao et al., 2017), Boundary Equilibrium GAN (Berthelot et al., 2017), GAN with denoising feature matching (Warde-Farley & Bengio, 2017), and Fisher GAN (Mroueh & Sercu, 2017).

Evaluation metrics Making reliable likelihood estimates with generative adversarial models is known to be challenging (Theis et al., 2016), and we instead focused on evaluating the visual quality of generated images, using datasets that come with labels for classification. We measured the *inception score* (Salimans et al., 2016). 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., (Yang et al., 2017; Che et al., 2017), the probabilities were estimated by a classifier trained with the labels provided with the datasets (instead of the ImageNet-trained *inception* model used in (Salimans et al., 2016)) so that the image classes of interest were well represented in the classifier. We, however, call this score the 'inception score', following custom. We also compared the label distributions over generated data and real data, but we found that in our settings this measure roughly correlates to the inception score (generally, a very good match when the methods produce decent inception scores), and so we do not report it to avoid redundancy. We note that these metrics are limited, e.g., they would not detect mode collapse or missing modes

within a class. Apart from that, we found the inception score to generally correspond to human perception well.

Data We used MNIST, the Street View House Numbers dataset (SVHN) (Netzer et al., 2011), and the large-scale scene understanding (LSUN) dataset. These datasets are provided with class labels (digits '0' – '9' for MNIST and SVHN and 10 scene types for LSUN). A number of studies have used only one LSUN class ('bedroom'). Since a single-class dataset would preclude evaluation using class labels, we instead generated a balanced two-class dataset using the same number of 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 real images used for training was 60K (MNIST), 521K (SVHN), 2.6 million (LSUN BR+LR), and 1.4 million (LSUN T+B). The LSUN images were shrunk and cropped into 64×64 as in previous studies. The pixel values were scaled into $[-1, 1]$.

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 (Radford et al., 2015), adding 1×1 convolution layers. Larger (64×64) images of LSUN were found to benefit from more complex networks, and so we used a residual net (ResNet) (He et al., 2015) 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 the Appendix.

These networks include batch normalization layers (Ioffe & Szegedy, 2015). The original study 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, in each setting, we tested WGANgp with the options of either removing batch normalization only from D or from both D and G, and picked the best. (We also tried other normalizations such as layer normalization but did not see any merit.) In addition, we tested some 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.

b	mini-batch size	64
$\lvert S_z \rvert$	input pool set size	$10b$
U	discriminator update frequency	1
T	number of iterations in ICFG	25

Table 1. Meta-parameters for xICFG.

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 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 (Gaussian with 0 mean and 0.01 std-dev) to produce the desired dimensionality, and then trained $\widetilde{G}$ to approximate G_{rand}. The training time reported below includes the time spent for this initialization.

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 1 except when we pursued smaller values for T for practical advantages (described below). For GAN, we used the same mini-batch size as xICFG, and we set the discriminator update frequency U to 1 as other values led to poorer results. The SGD update was done with *rmsprop* (Tieleman & Hinton, 2012) for xICFG and GAN. The learning rate for rmsprop was fixed for xICFG, but we tried several values for GAN 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 (Kingma & Ba, 2015) as in the original study. We set the meta-parameters for WGANgp to the suggested values, except that we tried several values for the learning rate. Thus, the amount of tuning effort was about the same for all but WGANgp, which required additional search for the normalization options. Tuning was done based on the inception score on the validation set of 10K input vectors (i.e., 10K 100-dim Gaussian vectors), and we report inception scores on the test set of 10K input vectors, disjoint from the validation set.

4.2. Results

First, we report the inception score results. The scores of the real data (a held-out set of 10K images) 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. Figure 2 shows the score of generated images (in relation to training time) with the convolutional networks, including the two cases without batch normalization anywhere for all the methods (upper-

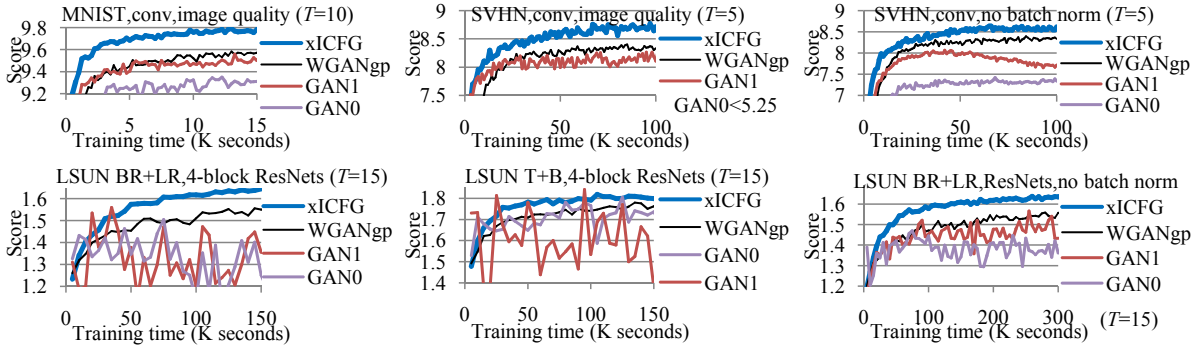


Figure 2. Image quality (measured by the inception score) in relation to training time. Convolutional networks. The legends are sorted from the best to the worst. The two graphs on the right are without batch normalization anywhere for all the methods.

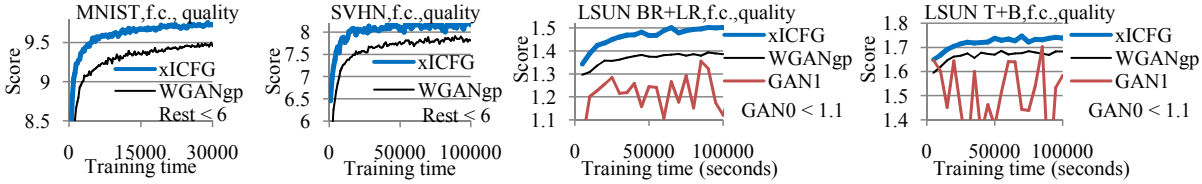
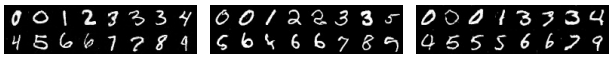


Figure 3. Image quality (measured by the inception score) in relation to training time. Fully-connected approximators/generators.

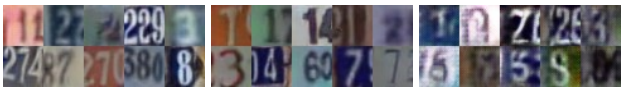


xICFG (9.78) Best baseline (9.57) Worst baseline (9.31)

Figure 4. MNIST. Using convolutional networks as in Fig 2. In this and all the image figures below, the images were randomly chosen and sorted by the predicted classes; the numbers are the inception scores averaged over 10K images.

xICFG (9.72) Best baseline (9.45) Worst baseline (1.08)

Figure 5. MNIST. Using fully-connected G or G as in Fig 3.



xICFG (8.72) Best baseline (8.33) Worst baseline (4.59)

Figure 6. SVHN. Using convolutional networks as in Fig 2.

xICFG (8.16) Best baseline (7.81) Worst baseline (3.37)

Figure 7. SVHN. Using fully-connected G or G as in Fig 3.

right and lower-right). Recall that 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 approximators, we explored values for T by a decrement of 5 starting from T=25 (which works well for all) and found that T can be reduced to 5 (SVHN), 10 (MNIST), and 15 (both LSUN) without negative consequences. The results in Figure 2 were obtained by using these smaller T. xICFG generally outperforms the others. Although on LSUN datasets GAN1 occasionally exceeds xICFG, inspection of generated images reveals that it suffers from severe mode collapse. The results with the simple but weak fully-connected approximator/generator are shown in Figure 3. 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.

Examples of generated images are shown in Figures 4–9. Note however that a small set of images may not represent the entire population well due to variability. Looking through larger sets of generated images, we found that roughly, when the inception score is higher, the images are sharper and/or there are fewer images that are harder to tell what they are, and that when the score fluctuates violently (as GAN1 does on LSUN), severe mode collapse is observed. Overall, we feel that the images generated by xICFG are better than or at least as good as those of the best-performing baseline, WGANgp, one of the state-of-the-art methods.

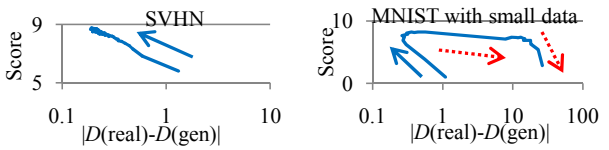
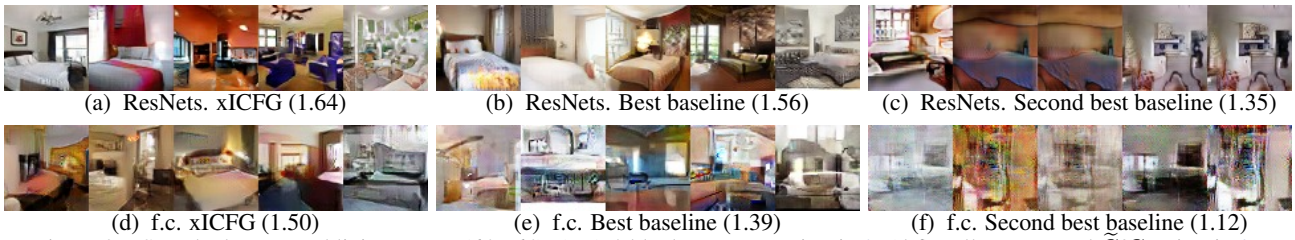


Figure 11. Image quality (y-axis) vs. $\Delta_D(=|D(\text{real})-D(\text{gen})|$, x-axis). xICFG. The arrows indicate the direction of time flow. A correlation is observed both when training is succeeding (blue solid arrows) and failing (red dotted arrows).

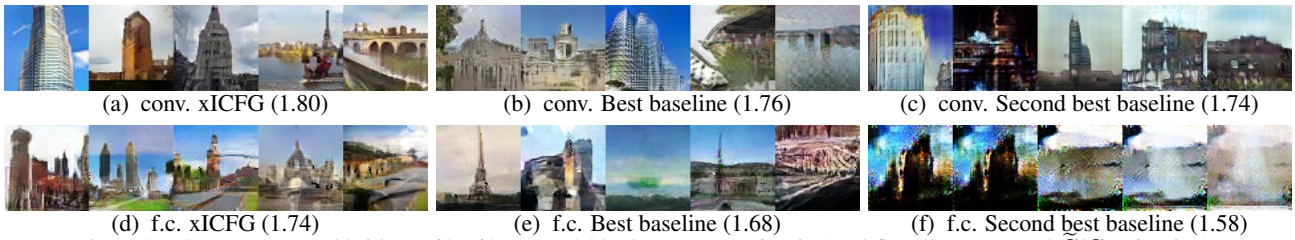
Discriminator output values Successful training should make it harder and harder for the discriminator to distinguish real images and generated images, which would man-



(a) ResNets. xICFG (1.64) (b) ResNets. Best baseline (1.56) (c) ResNets. Second best baseline (1.35)

(d) f.c. xICFG (1.50) (e) f.c. Best baseline (1.39) (f) f.c. Second best baseline (1.12)

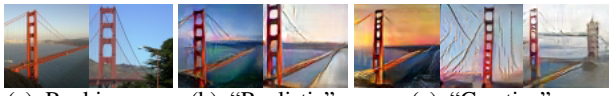
Figure 8. LSUN bedrooms and living rooms (64×64). (a-c) 4-block ResNets as in Fig 2. (d-f) Fully-connected $\widetilde{G}$/G as in Fig 3.



(a) conv. xICFG (1.80) (b) conv. Best baseline (1.76) (c) conv. Second best baseline (1.74)

(d) f.c. xICFG (1.74) (e) f.c. Best baseline (1.68) (f) f.c. Second best baseline (1.58)

Figure 9. LSUN towers and bridges (64×64). (a-c) 4-block ResNets as in Fig 2. (d-f) Fully-connected $\widetilde{G}$/G as in Fig 3.

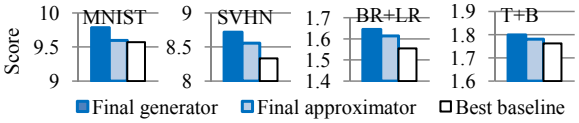


(a) Real images. (b) "Realistic" (c) "Creative"

Figure 10. (a) Real Golden Gate Bridge images in LSUN T+B; the red tower has 4 grids. (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 with more grids or connected with an object that is not there in reality.

ifest as the discriminator output values for real images and generated images becoming closer and closer. In Figure 11, the y-axis is the inception score, and the x-axis is '$|D(\text{real})-D(\text{gen})|$' ($\Delta_D$ 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. The arrows indicate the direction of time flow. 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 scores degrades rapidly (red dotted arrows in Fig 11 right). Here, the discriminator is overfitting to the small set of real data (1000 MNIST examples), violating the ϵ-approximation assumption. That slows down and eventually stops the progress of the generator, resulting in the increase of Δ_D. In practice, training should be stopped before the rapid growth of Δ_D. 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, similar to WGAN and in contrast to GAN.

Use of the approximator as a generator As noted above, a smaller generator resulting from a smaller T has practical advantages, but a larger T stabilizes training. One way to reduce generator size without reducing T is to use the final approximator $\widetilde{G}$ (after completing regular xICFG training) as a generator, at the expense of performance degradation. We show below the inception scores of the final approximator (G_0 in the final call of ICFG) in comparison with the final generator (G_T in the final call of ICFG) in the settings of Figure 2 (convolutional).



Although the final approximator underperforms the final generator as expected, it rivals and sometimes exceeds the best baseline method, whose generator has the same size as the approximator. Thus, use of the final approximator may be a viable option. The results also indicate that the good performance of xICFG is due to not only having a larger (and therefore more complex) generator but also stable and efficient training, which makes it possible to refine the approximator network to the degree that it can outperform the baseline methods.

Not memorization Finally, in Fig 10 we show examples indicating xICFG does something more than memorization.

5. 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

Arjovsky, M. and Bottou, L. Towards principled methods for training generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2017.

Berthelot, D., Schumm, T., and Metz, L. BEGAN: Boundary equilibrium generative adversarial networks. *arXiv:1703.10717*, 2017.

Che, T., Li, Y., Jacob, A. P., Bengio, Y., and Li, W. Mode regularized generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 2001. ISSN 0090-5364.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

Lazarow, J., Jin, L., and Tu, Z. Introspective neural networks for generative modeling. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017.

Li, C.-L., Chang, W.-C., Cheng, Y., Yang, Y., and Póczos, B. MMD GAN: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Smolley, S. P. Least squares generative adversarial networks. *arXiv:1611.04076*, 2017.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. Unrolled generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Mroueh, Y. and Sercu, T. Fisher GAN. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *Proceedings of NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Nitanda, A. and Suzuki, T. Gradient layer: Enhancing the convergence of adversarial training for generative models. *arXiv:1801.02227*, 2018.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.

Theis, L., van den Oord, A., and Bethge, M. A note on the evaluation of generative models. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

Tolstikhin, I., Gelly, S., Bousquet, O., Simon-Gabriel, C.-J., and Schölkopf, B. AdaGAN: Boosting generative models. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

Warde-Farley, D. and Bengio, Y. Improving generative adversarial networks with denoising feature matching. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Yang, J., Kannan, A., Batra, D., and Parikh, D. LR-GAN: Layered recursive generative adversarial networks for image generation. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Zhang, T. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32(1):56–85, 2004.