How to Train Your DRAGAN

Naveen Kodali[*]
School of Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
nkodali3@gatech.edu

Jacob Abernethy
EECS Department
University of Michigan
Ann Arbor, MI 48109
jabernet@umich.edu

James Hays
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA 30332
hays@gatech.edu

Zsolt Kira
Georgia Tech Research Institute
Georgia Institute of Technology
Atlanta, GA 30318
zkira@gatech.edu

Abstract

Generative Adversarial Networks have emerged as an effective technique for estimating data distributions. The basic setup consists of two deep networks playing against each other in a zero-sum game setting. However, it is not understood if the networks reach an equilibrium eventually and what dynamics makes this possible. The current GAN training procedure, which involves simultaneous gradient descent, lacks a clear game-theoretic justification in the literature. In this paper, we introduce regret minimization as a technique to reach equilibrium in games and use this to justify the success of simultaneous GD in GANs. In addition, we present a hypothesis that mode collapse, which is a common occurrence in GAN training, happens due to the existence of spurious local equilibria in non-convex games. Motivated by these insights, we develop an algorithm called DRAGAN that is fast, simple to implement and achieves competitive performance in a stable fashion across different architectures (150 random setups), datasets (MNIST, CIFAR-10, and CelebA), and divergence measures with almost no hyperparameter tuning. We show significant improvements over the recently proposed Wasserstein GAN variants.

1 Introduction

Generative modeling involves using a training set to learn probability distribution P_{model} that closely resembles the data generating distribution P_{real}. Generative Adversarial Networks (GANs) [9] are a class of implicit generative models that have achieved significant success in generating realistic samples for high-dimensional data. GANs find applications in a variety of domains, including settings demanding multi-modal outputs, use within model-based reinforcement learning algorithms, and incorporation of unlabeled data in semi-supervised settings [13]. At its core, GAN training is framed as a zero-sum game between two players - a generator G and discriminator D. The goal is to reach equilibrium of this game where the Jenson-Shannon divergence between P_{model} and P_{real} is minimized. The common practice in the community is to use simultaneous gradient descent to try and converge to this point. However, this procedure lacks a clear game theoretical justification from the literature and in fact, the original paper argues that ideally players should be trained to optimality at each step.

[*]Submitted to NIPS'17. Code - https://github.com/kodalinaveen3/DRAGAN

In this paper, we introduce regret-minimization (RM) [7, 6] as a technique to reach Nash Equilibrium in games. Leveraging the analysis of no-regret algorithms (specifically, online gradient descent [4]) in convex games, we motivate the use of simultaneous gradient descent in GAN training. This implies that current GAN training procedure will work provably under mild constraints in convex settings. A natural question then arises as to why it sometimes fails in practice. We hypothesize that the difficulty in training GANs, especially due to mode collapse, results from the existence of spurious local Nash Equilibria in non-convex games [8]. To address this problem, we propose a new algorithm (DRAGAN) that performs smoothing of the discriminator function by constraining its gradients around the real samples. Finally, we present a series of experiments to support our claims and demonstrate stable, competitive performance of DRAGAN in a wide variety of settings. We show results on the MNIST, CIFAR, and CelebA datasets.

2 Related Work

There have been several works aimed at finding a stable way to train GANs. Radford et al. [11] proposed a stable family of architectures called deep convolutional generative adversarial networks (DCGANs). We show that such constraints on architectures can be relaxed while still being able to achieve stability in the training process. In an alternate direction, a number of works have focused on developing specific objective functions that improve stability and performance of GANs. Salimans et al. [16] introduced a variety of techniques to improve the quality of samples. Che et al. [12] proposed a family of regularizers to address the missing modes problem in GANs. Zhao et al. [17] introduced energy based GAN framework which is more stable to train. Metz et al. [14] developed unrolled GANs taking inspiration from game theory literature. However, it suffers from slow performance due to the requirement of multiple unrolling steps in each iteration.

Recently, we have seen a series of works based on imposing a Lipschitz constraint on the discriminator function. Guo-Jun Qi [20] introduced LS-GAN with the idea of maintaining a margin between losses assigned to real and fake samples. Specifically, they enforce the following condition (the discriminator is used as the loss function in this setting) -

$$D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})) - D_{\boldsymbol{\theta}}(\mathbf{x}) \geq \Delta(\mathbf{x}, G_{\boldsymbol{\phi}}(\mathbf{z}))$$

where the $\Delta(.)$ is some distance metric. Further, they impose a Lipschitz constraint on both G and D using weight decay. This leads to D having non-vanishing gradients everywhere between all real and fake sample pairs in the limit. Arjovsky et al. [18] proposed Wasserstein GAN which uses Earth-Mover distance as the objective to address problems with the vanilla objective. Their procedure requires the discriminator to be a Lipschitz function and they use weight clipping to achieve that. Gulrajani et al. [19] proposed an extension to address various shortcomings of the original WGAN and they impose the following condition on D -

$$||\nabla_{\hat{\mathbf{x}}} D_{\boldsymbol{\theta}}(\hat{\mathbf{x}})||_2 \approx 1$$

where $\hat{\mathbf{x}} = (\epsilon)\mathbf{x} + (1-\epsilon)G_{\boldsymbol{\phi}}(\mathbf{z})$ is some point between randomly chosen real and fake sample pairs. This leads to D having norm 1 gradients everywhere between all real and fake sample pairs in the limit. Notice that this behavior is very similar to that of LSGAN's discriminator function. We argue that such constraints are too restrictive and can encourage poor generator functions. This is demonstrated in section 4 where improved WGAN fails to accurately capture a simple data distribution. Additionally, the requirement of multiple updates to D at each step makes it slow. Our proposed method address a fundamental game theoretic issue in the GAN training process and hence it can be used on top of a variety of objective functions. We show that it is possible to get excellent results using just the vanilla objective function and a single update to D.

3 Formulation and Proposed Algorithm

3.1 Game Formulation

The GAN framework consists of two players - the *generator* whose aim is to create realistic samples and the *discriminator* whose aim is to classify an instance as having come from training data or the generator. The generator model G is parameterized by $\boldsymbol{\phi}$, takes a latent variable $\mathbf{z}$ as input, and outputs sample $G_{\boldsymbol{\phi}}(\mathbf{z})$. The discriminator model D is parameterized by $\boldsymbol{\theta}$, takes a sample $\mathbf{x}$ as input and outputs $D_{\boldsymbol{\theta}}(\mathbf{x})$ which represents the probability that it is real. The models G, D are usually represented by deep networks and their cost functions are defined as -

$$J^{(D)}(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{real}} \log D_{\boldsymbol{\theta}}(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})))$$

$$J^{(G)}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})))$$

The complete game can be specified as -

$$\min_{\boldsymbol{\phi}} \max_{\boldsymbol{\theta}} \{g(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}\sim p_{real}} \log D_{\boldsymbol{\theta}}(\mathbf{x}) + \mathbb{E}_{\mathbf{z}} \log(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})))\}$$

At equilibrium, JS divergence between the data and model distributions is minimized meaning G has converged to P_{real} and D outputs 1/2 for all $\mathbf{x}$.

3.2 Proposed Algorithm

Given this background, we first describe our algorithm and then provide our theoretical motivation and justification for it in Section 4. Algorithm 1 shows pseudo-code for our proposed approach. We deviate from the vanilla algorithm in how we update the discriminator. Our proposed method involves constraining points in local regions around real examples to have norm 1 gradients. To achieve this, we generate three different mini-batches in each iteration: Real data examples from the training set x^i, samples from the normal distribution z^i, and real data examples perturbed with small noise $x^i + \delta^i$ (the magnitude of noise decides the size of local regions). Ideally, we would like to be able to impose the constraint everywhere in these local regions but to keep it tractable, we only apply it at randomly chosen points (as shown in step 6). The objective function of D is appended with a regularization term that penalizes violations of the above constraint.

Algorithm 1 DRAGAN

```
1: Initial weights $(\boldsymbol{\phi}_0, \boldsymbol{\theta}_0)$ (for the generator and discriminator, respectively)
2: for number of training iterations do
3:     Get a mini-batch of real examples $\{x^1, x^2, ..., x^m\}$ from training data.
4:     Get a mini-batch of samples $\{z^1, z^2, ..., z^m\}$ from $\mathcal{N}(0,1)$
5:     Get $\{x^1 + \delta^1, x^2 + \delta^2, ..., x^m + \delta^m\}$ where each $\delta^i$ is small noise vector.
6:     Define $\hat{x}^i = \alpha \cdot x^i + (1-\alpha) \cdot (x^i + \delta^i)$ for random $\alpha \in \mathcal{U}[0,1]$
7:     Update the discriminator by descending its gradient (using Adam):
8:         $\nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} \Big[ -\log D_{\boldsymbol{\theta}}(x^i) - \log(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(z^i))) + \boldsymbol{\lambda} \cdot (||\nabla_{\hat{x}^i} D_{\boldsymbol{\theta}}(\hat{x}^i)||_2 - 1)^2 \Big]$
9:     Update the generator by descending its gradient (using Adam):
10:        $\nabla_{\boldsymbol{\phi}} \frac{1}{m} \sum_{i=1}^{m} \log \Big[ 1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(z^i)) \Big]$
11: end for
```

We now provide some implementation details for our algorithm:

Hyperparameters We use Adam [10] with default parameters everywhere. The hyperparameter $\boldsymbol{\lambda}$ is set to 10 in all of our experiments.

Perturbed Minibatch In step 6 of our algorithm, we get a perturbed minibatch of size m by adding small noise vectors $\{\delta^1, ..., \delta^m\}$ to the real minibatch. As described earlier, we do this to obtain points in local regions of $\{x^1, ..., x^m\}$ and constrain D to have norm 1 gradients there. Gulrajani et al. [19] use a similar penalty term albeit for different reasons. Moreover, we apply this constraint in local regions and show that this performs better experimentally in the next few sections.

In our code, we perform the operation shown below to perturb minibatches -

$$\textit{perturbed minibatch} = \textit{minibatch} + C \cdot \textit{minibatch.std()} \cdot \mathcal{U}[0,1]$$

We set $C = 0.5$ in all of the experiments.

Batch Normalization (BN) We did not use batch normalization in most of our experiments. This is because BN introduces correlations between samples in a minibatch and this affects our ability to impose local constraints. We replace it with layer normalization in some experiments and show one experiment with BN.

4 Theoretical Foundations

In the present section we describe how the training procedure for GANs in general can be viewed through the lens of equilibrium computation in zero-sum convex/concave games. The simultaneous gradient descent procedure corresponds to regret minimization techniques which are well-studied in the online learning and game theory communities. We observe, of course, that the lack of convexity leads to multiple saddle points (Nash equilibria), but we argue that the existence of such non-optimal solutions can be mitigated by the appropriate *smoothing* of the payoff function via regularization. This idea was used by Nash himself in his early work [2]. We argue in favor of this hypothesis empirically in Section 5.

4.1 Equilibrium Computation and Generalization via Regret Minimization

Von Neumann established the minimax theorem for zero-sum games in [1] or equivalently the existence of *mixed Nash equilibria*. Nash and von Neumann originally envisioned two players selecting probability distributions over a finite strategy set, and receiving randomized rewards according to the game payoff function and these distributions. The minimax theorem has been significantly generalized since this original formulation, and the work of Sion [3] provides a broad class of scenarios in which $\inf\sup\{\} = \sup\inf\{\}$. For example, when we are given convex and compact subsets or $\Theta \subset \mathbb{R}^n$ and $\Phi \subset \mathbb{R}^m$ and a function $g : \Theta \times \Phi \to \mathbb{R}$ that is convex in its first argument and concave in its second, then Sion's theorem implies that

$$\min_{\theta\in\Theta}\max_{\phi\in\Phi} g(\theta,\phi) = \max_{\phi\in\Phi}\min_{\theta\in\Theta} g(\theta,\phi).$$

For such min/max optimization scenarios, we will use the term *Nash equilibrium pair* to refer to any θ^*, ϕ^* such that $\theta^* \in \arg\min_{\theta\in\Theta}\max_{\phi\in\Phi} g(\theta,\phi)$ and $\phi^* \in \arg\max_{\phi\in\Phi}\min_{\theta\in\Theta} g(\theta,\phi)$.

Many problems, including the development of GANs, can be posed as a min/max formulation and hence a natural question is how we can find such equilibrium pairs. The most straightforward procedure by which players might search for an equilibrium is best-response dynamics (BRD). In each round, best-responding players play their optimal strategy given their opponent's current strategy. Despite its simplicity, BRD does not necessarily converge even in simple convex settings and can lead to oscillations/instability. At least in the setting where the payoff $g(\cdot,\cdot)$ is convex/concave, we have a technique that is both efficient and provably works: *no-regret learning algorithms* (See, e.g., Chapter 4 of [7]). If both players treat the selection of their strategy parameters θ_t, ϕ_t as a sequential game ($t = 1, 2, \ldots$), and they update these parameters via no-regret dynamics, then it is easy to show that the time-average parameter vectors will converge to a Nash equilibrium pair.

Let us recall the definition of a no-regret algorithm. Given a sequence of convex loss functions $L_1, L_2, \ldots : \Theta \to \mathbb{R}$, an algorithm that selects a sequence of θ_t's, each of which may only depend on previously observed $L_1, \ldots, L_{t-1}$, is said to have *no regret* if $\frac{R(T)}{T} = o(1)$, where we define

$$R(T) := \textstyle\sum_{t=1}^T L_t(\boldsymbol{\theta}_t) - \min_{\boldsymbol{\theta}\in\Theta}\sum_{t=1}^T L_t(\boldsymbol{\theta})$$

We apply no-regret learning to the problem of equilibrium finding in a game $g(\cdot,\cdot)$ as follows. The θ player imagines the function $g(\cdot,\phi_t)$ as his loss function on round t, and similarly the ϕ player imagines $-g(\theta_t,\cdot)$ as her loss function at t. After T rounds of play, each player computes the average iterates $\bar\theta_T := \frac{1}{T}\sum_{t=1}^T \theta_t$ and $\bar\phi_T := \frac{1}{T}\sum_{t=1}^T \phi_t$. If V^* is the equilibrium value of the game, and the θ and ϕ players suffer regret $R_1(T)$ and $R_2(T)$ respectively, then it is easy to show that

$$V^* - \tfrac{R_2(T)}{T} \le \max_{\phi\in\Phi} g(\bar\theta_T,\phi) - \tfrac{R_2(T)}{T} \le \min_{\theta\in\Theta} g(\theta,\bar\phi_T) + \tfrac{R_1(T)}{T} \le V^* + \tfrac{R_1(T)}{T}.$$

In other words, $\bar\theta_T$ and $\bar\phi_T$ are "almost optimal" solutions to the game, where the "almost" approximation factor is given by the average regret terms $\frac{R_1(T)+R_2(T)}{T}$. Under the no-regret condition, the former will vanish, and hence we can guarantee convergence in the limit.

Let us consider a specific family of no-regret algorithms, and in particular we look at the well-studied family [6] known as Follow The Regularized Leader (FTRL). Specifically, FTRL selects θ_t on round t by solving for $\arg\min_{\theta\in\Theta}\{\sum_{s=1}^{t-1} L_s(\theta) + \frac{1}{\eta}\Omega(\theta)\}$, where $\Omega(\cdot)$ is some convex "regularization function" and η is a learning rate. Roughly speaking, if you select the regularization as $\Omega(\cdot) = \frac{1}{2}\|\cdot\|^2$, then FTRL becomes the well-known Online Gradient Descent (OGD) [4]. Ignoring the case of constraint violations, OGD can be written in a simple iterative form: $\theta_t = \theta_{t-1} - \eta \nabla L_t(\theta_{t-1})$.

Of course the min/max objective function in GANs involves a stochastic payoff function, with two randomized inputs given on each round, $\mathbf{x}$ and $\mathbf{z}$ which are sampled from the data distribution and a standard multivariate normal, respectively. Let us write $g_{\mathbf{x},\mathbf{z}}(\theta, \phi) := \log D_\theta(\mathbf{x}) + \log(1 - D_\theta(G_\phi(\mathbf{z})))$. Taking expectations with respect to $\mathbf{x}$ and $\mathbf{z}$ we define the full (non-stochastic) game as $g(\theta, \phi) = \mathbb{E}_{\mathbf{z}} \mathbb{E}_{\mathbf{x} \sim p_{real}} [g_{\mathbf{x},\mathbf{z}}(\theta, \phi)]$. But the above online training procedure is still valid with stochastic inputs. That is, the equilibrium computation would proceed similarly, where on each round we sample $\mathbf{x}_t$ and $\mathbf{z}_t$, and follow the updates

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta g_{\mathbf{x}_t, \mathbf{z}_t}(\theta_t, \phi_t) \quad \text{and} \quad \phi_{t+1} \leftarrow \phi_t - \eta \nabla_\phi g_{\mathbf{x}_t, \mathbf{z}_t}(\theta_t, \phi_t).$$

A huge benefit of this stochastic perspective is that we immediately get a generalization bound on the mean parameters $\bar{\theta}_T$ after T rounds of optimization. The celebrated "online-to-batch conversion" [5], now a standard result in online learning theory, implies that $\mathbb{E}_{\mathbf{x},\mathbf{z}}[g_{\mathbf{x},\mathbf{z}}(\bar{\theta}_T, \phi)]$, for any ϕ, is no more than the optimal value $\mathbb{E}_{\mathbf{x},\mathbf{z}}[g_{\mathbf{x},\mathbf{z}}(\theta^*, \phi)]$ plus an "estimation error" bounded by $\mathbb{E}\left[\frac{R_1(T)+R_2(T)}{T}\right]$, where the expectation is taken with respect to the sequence of samples observed along the way, and any randomness in the algorithm. A limitation of this result, however, is that it requires a fresh sample $\mathbf{x}_t$ to be used on every round.

4.2 Local Nash equilibria and Smoothing in games

The discussion above, and the very general results one can prove using no-regret techniques, still relies on the crucial assumption that the payoff function $g(\cdot, \cdot)$ is convex-concave. Indeed convexity is used in a number of places in the proofs: vanishing regret relies on a sequence of convex loss functions, and the online-to-batch conversion rests on Jensen's inequality to show $\frac{1}{T}\sum_{t=1}^{T} g(\theta_t, \phi) \geq g\left(\frac{1}{T}\sum_{t=1}^{T} \theta_t, \phi\right)$. However, due to the non-convex nature of GAN settings, we are not guaranteed that a sole Nash equilibrium exists – there may be many non-optimal saddle points, which are essentially local minima of the game. On the other hand, *if* the no-regret dynamics exhibits convergence of $\bar{\theta}_T$ and $\bar{\phi}_T$ to some fixed points $\hat{\theta}$ and $\hat{\phi}$, then at the very least we know this pair will be a saddle point, albeit potentially non-optimal. In the GAN literature, it has been widely observed that *mode collapse* is quite commonly observed in practice and has been attributed to the non-convex nature of the model.

One of the striking features of the experimental results presented herein is that mode collapse is significantly rarer under the modified game we have considered, and this feature of the optimization may be a significant contributor to our robust performance gains. The novel aspect of our algorithm is the additional penalization of the gradients in feature space, which is (roughly) $\lambda \mathbb{E}_{\mathbf{x}} \mathbb{E}_{(\delta \sim N(0, \epsilon I))}[||\nabla_{\mathbf{x}} D_\theta(\mathbf{x} + \delta)|| - 1)^2]$. This is, in effect, biasing the discriminator function to have norm-(≈ 1) gradients in a region around the real-data manifold. We recall an important fact, that *any smooth function with norm-1 gradients on a compact set is necessarily a linear function*, which suggests that our procedure encourages "close to linearity" at least along the data manifold. The space of linear functions, of course, forms a convex set and would therefore involve only a single global optimum. We believe the following conjecture accounts for the dramatic reduction in mode collapse: *the local norm-1 gradient regularization penalty provides sufficient smoothing of the game payoff to significantly reduce the space of non-optimal saddle points.* We present some experimental validation on MNIST that the penalty increases as we enter mode collapse events in supplementary Sections 2.1 and 2.2.

4.3 Regularization Scheme

Several recent works have proposed regularized models of GANs to improve stability . Our proposed approach roughly falls under this class of models and specifically, improved WGAN is the closest related approach to ours. Despite being developed from completely different motivations (Wasserstein distance and game theory), both impose a similar form of constraint on the gradients of D(x). However, there are differences with important implications for generative modeling performance.

The most important distinction is that we only impose these constraints in local regions around real samples, while improved WGAN imposes them everywhere. We claim that the resulting class of functions that improved WGAN can model is highly restricted compared to our method. This can be demonstrated with a simple experiment. In Figure 1, we track the performance of vanilla GAN, improved Wasserstein GAN and our algorithm on the swissroll dataset[2] over the training period. Observe that the vanilla algorithm quickly captures the data density, our algorithm captures it approximately but improved WGAN fails to accurately capture this distribution (we show another experiment with 8-Gaussians dataset in the supplementary Section 2.3). The experimental results in section 5.1 further support our claim.

The proposed regularization, which restricts the discriminator function only along the real-data manifold, allows for broader flexibility of $D_\theta(\cdot)$ in the vastly larger region outside of true density. Indeed the natural image space is mostly empty, i.e. there are low probability regions between any two modes. Modes that are far away from each other (for instance images of cars vs animals) can be equally realistic and modes close to each other (for instance images of cats with four legs vs cats with three legs) can have very different probabilities of being real. Such variations in data density cannot be modeled accurately when norm 1 gradients are imposed everywhere.

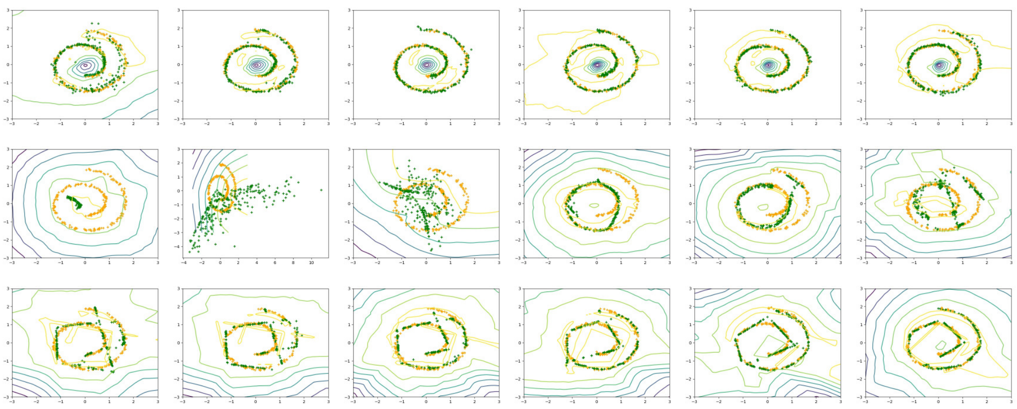


Figure 1: Comparing the performance of different algorithms on swissroll dataset - Vanilla GAN (top), Improved WGAN (middle), and our algorithm (bottom). The contours represent regions where the discriminator $D_{\boldsymbol{\theta}}(x)$ assigns low weight (purple) to high weight (yellow) of predicted likelihood of "real". We also plot real samples in orange, and generated samples in green.

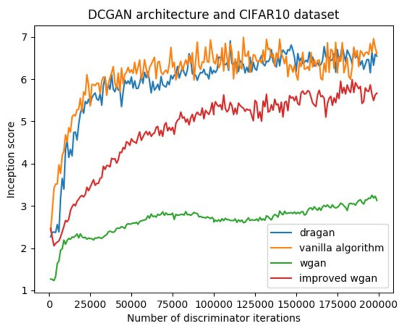
5 Experimental Results

We present a series of experiments using the MNIST, CIFAR-10, and CelebA datasets demonstrating competitive inception score [16] results and sample quality compared to the baseline algorithms. Importantly, we also demonstrate significant improvement in stability for our algorithm across a large randomized set of network architectures and across a range of divergence measures. For the set of randomized architectures, we perform qualitative and quantitative assessment ranking of our algorithm compared to the vanilla GAN algorithm to assess improvements in stability, i.e. performance characteristics related to mode collapse or failure to learn.

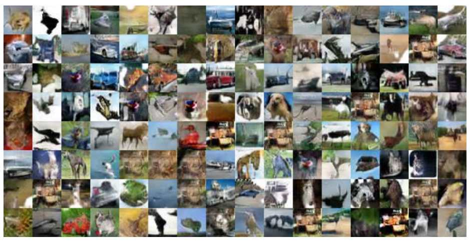
5.1 Inception Scores on CIFAR-10 using DCGAN architecture

The DCGAN is a family of architectures designed to perform well with the vanilla training procedure [11]. They are ubiquitous in the GAN literature owing to the instability of the vanilla algorithm in general settings. We use this architecture to model the CIFAR-10 dataset and compare to the vanilla GAN, WGAN, and improved WGAN. We did not add normalization layers in the discriminator for our algorithm and improved WGAN. As can be seen from Figure 2, our algorithm performs competitively with the vanilla procedure in terms of speed, stability and best inception score. Our final sample quality is superior in some cases. In contrast, improved WGAN achieves a significantly

[2]We use experimental setups from https://github.com/igul222/improved_wgan_training



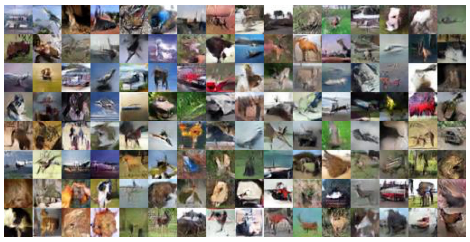
(a) Inception Score Plot



(c) Vanilla GAN

	Inception Score (CIFAR-10)
Real Images	11.24
WGAN	3.25
Improved WGAN	5.99
Vanilla GAN	6.99
Our algorithm	6.90

(b) Inception scores



(d) Our algorithm

Figure 2: Performance using DCGAN on CIFAR-10 across several algorithms. Right: Samples from Vanilla GAN and our algorithm.

lower score as shown in the table on Figure 2. Note that the vanilla procedure is hard to beat in this setting since DCGANs are built with architectural constraints that make them inherently stable. We present samples for all algorithms and some latent space walks, including for the CelebA dataset, in supplementary Section 1.

5.2 BogoNet score to measure stability across architectures

One of the key advantages of our algorithm is that it can achieve better stability in a wider range of architectures (other than DCGANs). Similar to [18, 19] we have removed stabilizing components of DCGAN and demonstrated improved reliability compared to the vanilla GAN algorithm (see Section 2.4 in supplementary). However, to measure this to a larger extent, we introduce a metric termed the BogoNet score to compare stability of different training procedures in the GAN setting. The basic idea is to choose random architectures for players G and D independently and evaluate the performance of different algorithms in the resulting games. A good algorithm should give stable performance without failing to learn and without mode collapse despite the potentially imbalanced architectures.

In our experiment, each player is assigned a network from a diverse pool of architectures from three families (MLP, ResNet, DCGAN). See Section 3 in supplementary for details. To demonstrate that our algorithm is more stable compared to the vanilla procedure, we created 150 such instances of hard games. For each instance, we trained using both the Vanilla and our algorithm for the CIFAR-10 dataset and plotted the inception score over time. We performed both qualitative and quantitative analysis of the resulting graphs to measure stability in the performance of both algorithms.

For qualitative analysis, we used a bounty model to assign scores to each algorithm. We used 50 of the total 150 instances generated for this analysis. Each instance is worth 5 points and we split this bounty between the two algorithms depending on their performance. If both perform well or perform poorly, they get 2.5 points each. However, if one algorithm achieves stable performance compared to the other (in terms of lack of learning or mode collapses), we assign it higher portions of the bounty. Results were judged by two of the authors in a blind manner: The curves were shown

Algorithm	**Final Inception Score**		**Area Under Curve**		**Linear Fit**		**Qual. Score**
	Mean	Std	Mean	Std	Mean Intercept	Mean Slope	Total
Vanilla	2.91	1.44	277.72	126.09	2.48	0.00654	92.5
Our Algorithm	**3.70**	1.71	**312.15**	135.35	2.33	**0.01658**	**157.5**

Table 1: Summary of inception score statistics across 100 architectures

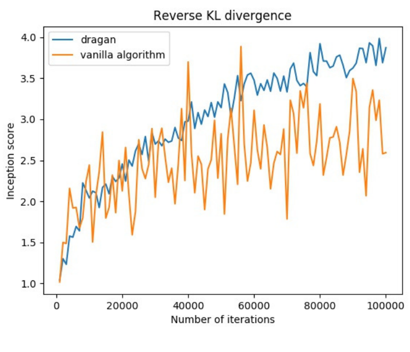
side-by-side with the choice of algorithm for each side being randomized and unlabeled. The Vanilla GAN received an average score of 92.5 while our algorithm achieved an average score of 157.5, demonstrating much better stability properties across a range of architectures.

For quantitative analysis, we used the remaining 100 instances. For each algorithm we calculated statistics of final inception scores and area under the curve (AUC) over all 100 instances. However, these statistics do not capture the notion of stability well. To alleviate this, we perform linear regression on the series of inception scores observed in each instance to capture the overall trend. Ideally, we want consistent increase in the inception score over time and this would indicate stable performance. This means the resulting linear fit should have a positive slope with high magnitude. We calculate statistics of this slope over all 100 instances. The results are shown below in Table 1.

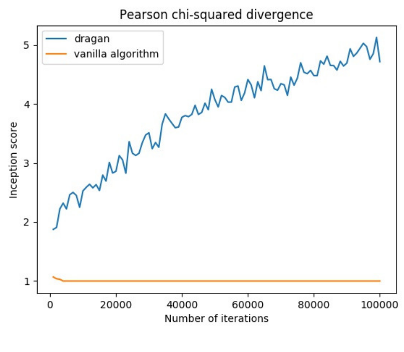
Notice that while the mean and y-intercept in the linear fit of both the algorithms is almost the same, the mean slope of inception score linear fits is 2.5 times higher for our algorithm. Clearly, this is the result of mitigating mode collapse and other stability issues. In some architectures, the vanilla algorithm does well but it goes into mode collapse in a lot of the cases. The average slope of linear fit captures this behavior well. Another way to interpret this is that given a random architecture, we are likely to get 2.5 times higher inception score, which is a non-trivial result. To summarize, both the qualitative and the quantitative analysis demonstrate that we achieve more stable performance compared to the vanilla procedure and solve stability issues to some extent.

5.3 Robustness across Divergence Measures

Nowozin et al. [15] show that any f-divergence can be used for training GANs with the help of an appropriate $generator function$. We show experiments using Forward KL-Divergence, Reverse KL, Pearson χ^2, Squared Hellinger, and Total Variation divergences. We use the case "4-layer 512-dim ReLU MLP generator" from previous subsection to demonstrate this in a challenging setting. Our algorithm is stable in all cases except for Total Variation while the vanilla algorithm failed in all cases (see Figure 3 for two examples and supplementary 2.5 for all five). Hence, practitioners can now use a larger set of objective functions that are appropriate for their application (unlike the LSGAN or WGAN family of algorithms).



(a) Reverse KL



(b) Pearson χ^2

Figure 3: Inception score curves for two divergence measures, demonstrating superior stability for our algorithm.

6 Conclusions

In this paper, we draw upon the game theory literature to justify the current GAN training procedure and propose an improved algorithm. We connect the idea of regret minimization to GANs and provided a novel way to reason about dynamics in this game. Given this background, we analyze mode collapse from a game-theoretic perspective and hypothesize that spurious local equilibria are responsible for this issue. Based on this, we propose a novel regularization scheme as part of our algorithm DRAGAN. Unlike previous extensions, our algorithm is simple to implement, fast, and improves stability in a wide variety of settings.

References

[1] J. von Neumann. "Zur Theorie der Gesellschaftsspiele". In: *Mathematische Annalen* 100 (1928), pp. 295–320. URL: http://eudml.org/doc/159291.

[2] John Nash. "Two-person cooperative games". In: *Econometrica: Journal of the Econometric Society* (1953), pp. 128–140.

[3] Maurice Sion. "On general minimax theorems". In: *Pacific J. Math* 8.1 (1958), pp. 171–176.

[4] Martin Zinkevich. "Online convex programming and generalized infinitesimal gradient ascent". In: (2003).

[5] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. "On the generalization ability of on-line learning algorithms". In: *IEEE Transactions on Information Theory* 50.9 (2004), pp. 2050–2057.

[6] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[7] Noam Nisan et al. *Algorithmic game theory*. Vol. 1. Cambridge University Press Cambridge, 2007.

[8] Lillian J Ratliff, Samuel A Burden, and S Shankar Sastry. "Characterization and computation of local nash equilibria in continuous games". In: *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*. IEEE. 2013, pp. 917–924.

[9] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[10] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[11] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *arXiv:1511.06434 [cs]* (Nov. 2015). arXiv: 1511.06434. URL: http://arxiv.org/abs/1511.06434 (visited on 05/16/2017).

[12] Tong Che et al. "Mode Regularized Generative Adversarial Networks". In: *arXiv preprint arXiv:1612.02136* (2016).

[13] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *arXiv preprint arXiv:1701.00160* (2016).

[14] Luke Metz et al. "Unrolled Generative Adversarial Networks". In: *CoRR* abs/1611.02163 (2016). URL: http://arxiv.org/abs/1611.02163.

[15] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. "f-GAN: Training generative neural samplers using variational divergence minimization". In: *Advances in Neural Information Processing Systems*. 2016, pp. 271–279.

[16] Tim Salimans et al. "Improved Techniques for Training GANs". In: *CoRR* abs/1606.03498 (2016). URL: http://arxiv.org/abs/1606.03498.

[17] Junbo Zhao, Michael Mathieu, and Yann LeCun. "Energy-based generative adversarial network". In: *arXiv preprint arXiv:1609.03126* (2016).

[18] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein gan”. In: *arXiv preprint arXiv:1701.07875* (2017).

[19] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *arXiv preprint arXiv:1704.00028* (2017).

[20] Guo-Jun Qi. “Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities”. In: *CoRR* abs/1701.06264 (2017). URL: `http://arxiv.org/abs/1701.06264`.

Supplementary Material

1 Samples and Latent Space Walks

In this section, we provide additional samples across algorithms and datasets. Further, Radford et al. [1] suggest that walking on manifold learned by the generator can expose signs of memorization. We use DCGAN to model MNIST and CelebA datasets using our algorithm and the results shown below demonstrate that our generator learns smooth transitions between different images.

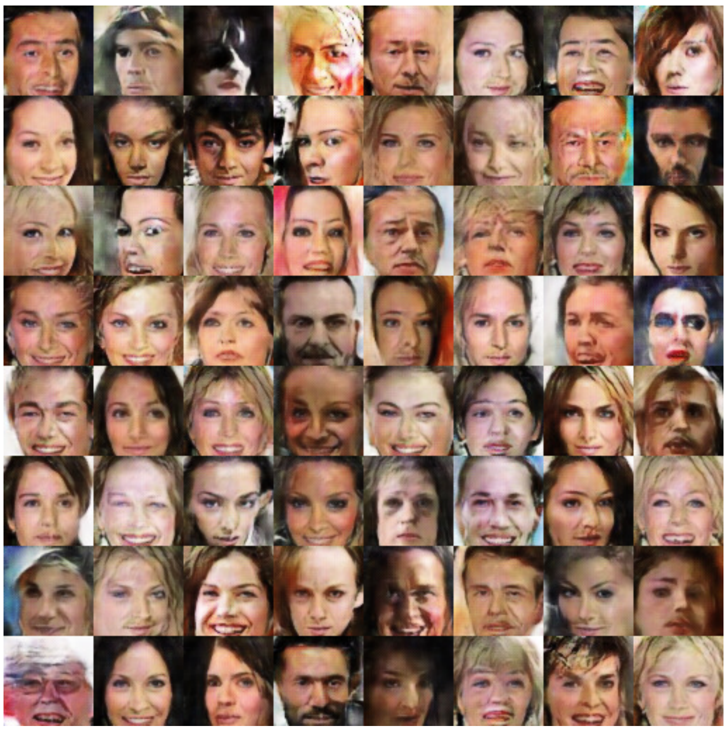
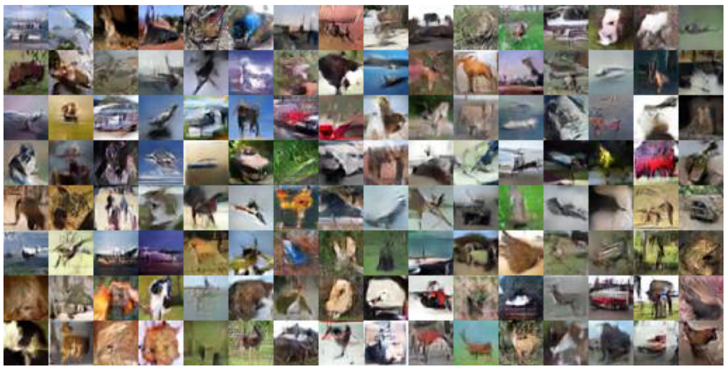
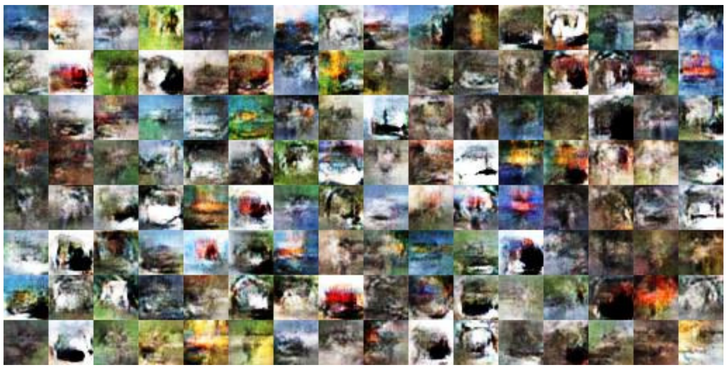


Figure 1: Samples for CelebA dataset using our algorithm with DCGAN architecture.

(a) Vanilla GAN


(b) Our Algorithm


(c) WGAN

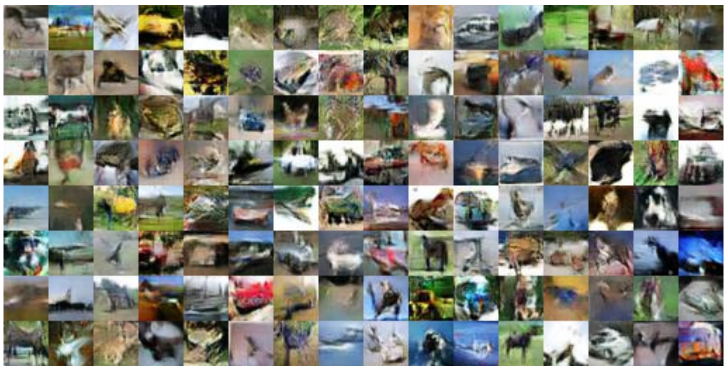
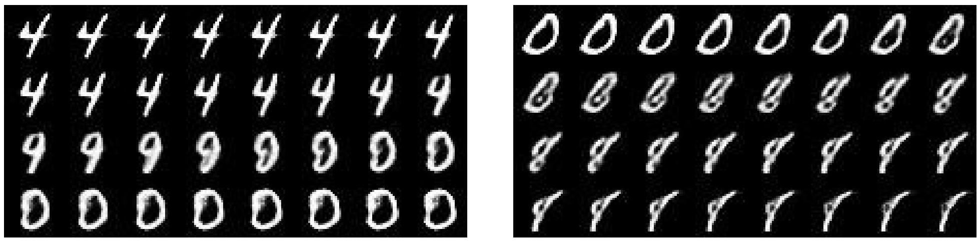

(d) Improved WGAN

Figure 2: Samples for CIFAR-10 dataset using DCGAN architecture



(a)

(b)

Figure 3: Latent space walk for MNIST using DCGAN architecture and our algorithm

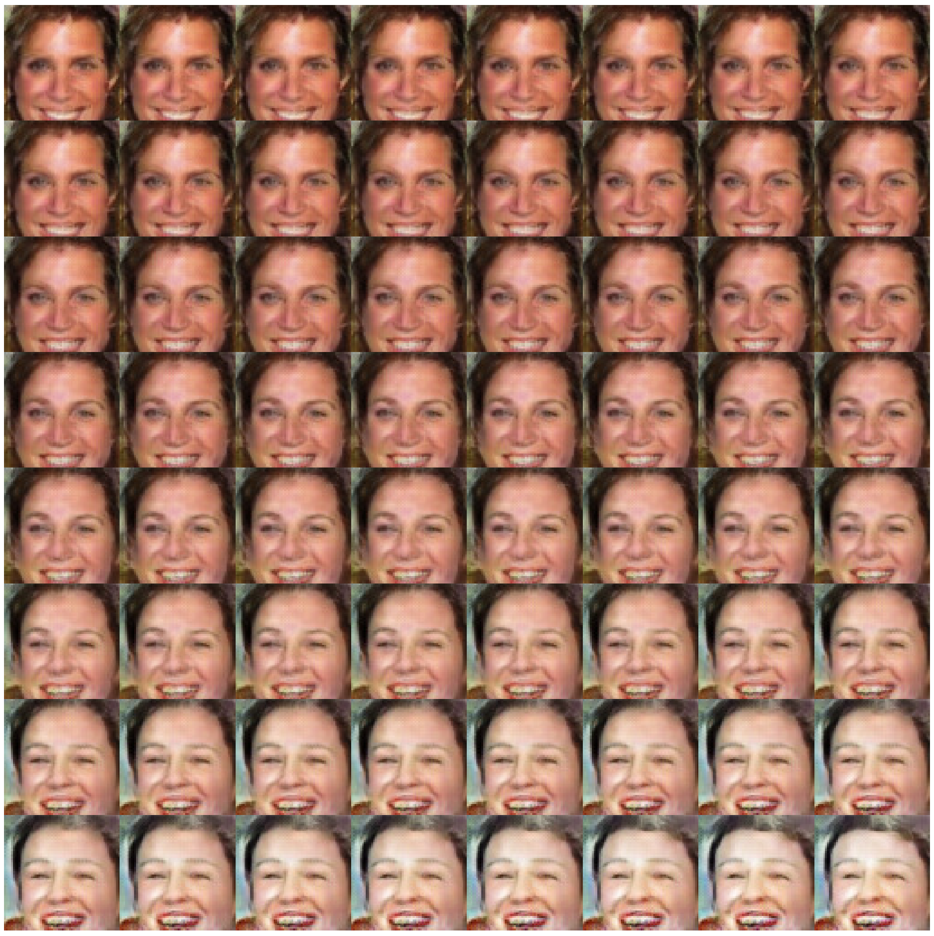


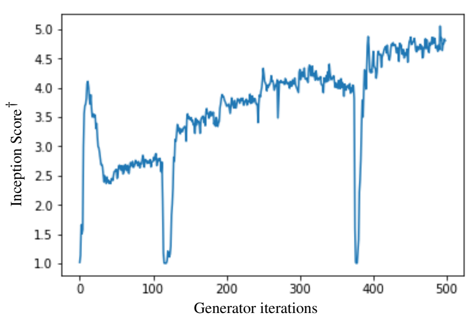
Figure 4: Latent space walk for CelebA dataset using DCGAN architecture and our algorithm

2 Additional Experiments

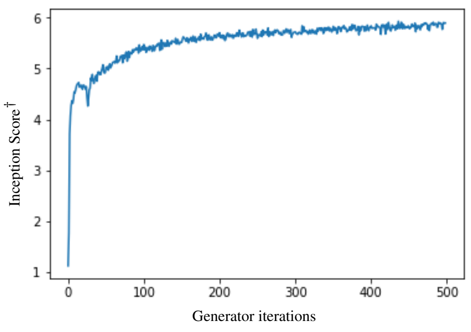
The inception score was introduced as a rough guide to evaluate generative models. To compute this score, we need a classifier model and the original paper suggests use of inception model [2] for this purpose. However, this model is too powerful for smaller datasets like MNIST and this makes it hard to analyze differences in the modeling capabilities of various algorithms. Therefore, we use a custom classifier built for the MNIST dataset to calculate inception scores in the next two subsections (we use a dagger to make this distinction).

2.1 One layer networks with MNIST dataset

We design a simple experiment where G and D are both fully connected networks with just one hidden layer. Vanilla GAN performs poorly even in this simple setting and we observe severe mode collapse events. In contrast, our algorithm is stable throughout and obtains decent quality samples despite the constrained setup.

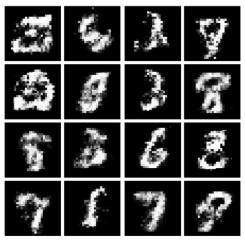


(a) Vanilla GAN

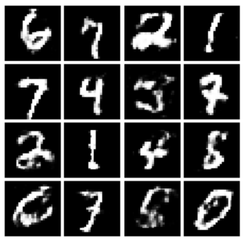


(b) Our algorithm

Figure 5: One layer networks with MNIST dataset - Inception score† plots



(a) Vanilla GAN

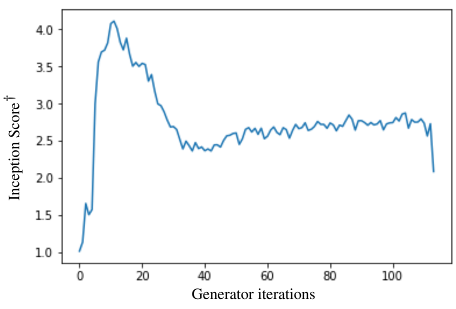


(b) Our algorithm

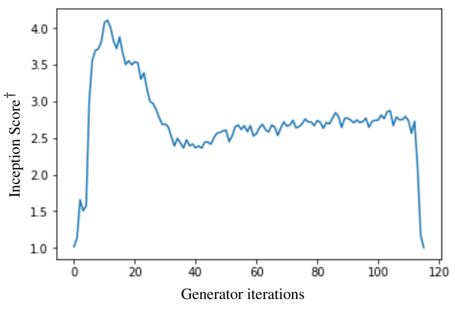
Figure 6: One layer networks with MNIST dataset - Samples

2.2 Smoothing penalty tracking

The stability in our algorithm comes from regularizing D with the penalty $(||\nabla_{\hat{\mathbf{x}}} D_{\boldsymbol{\theta}}(\hat{\mathbf{x}})||_2 - 1)^2$ as formulated in step 9 of Algorithm 1. We now take a closer look at mode collapse events that occur in the vanilla GAN and track the penalty in parallel (see Figure 7). Notice how the penalty increases as we enter mode collapse and drops as we recover. This indicates that the nature of the discriminator's gradients has a strong correlation with stability in GANs.



(i) Gradient penalty = 4.103e+03

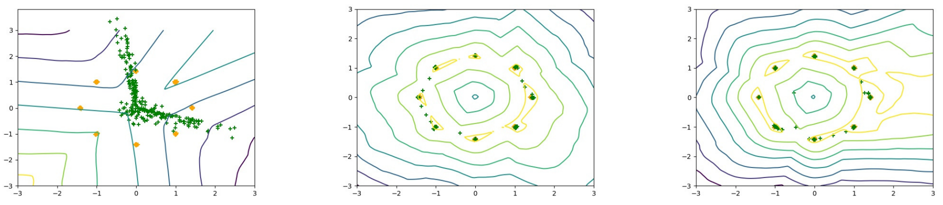


(ii) Gradient penalty = 6.267e+03

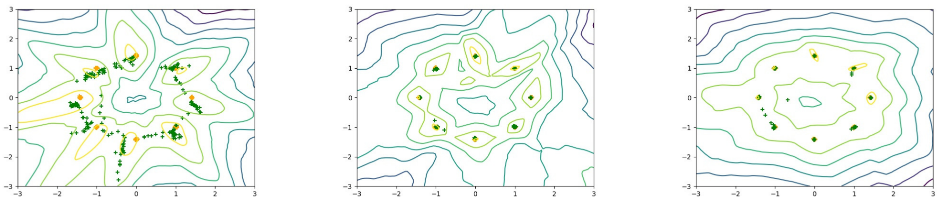
Figure 7: Penalty values at two points in the training of a vanilla GAN.

2.3 8-Gaussians Experiment

We track the performance of improved WGAN and our algorithm on the 8-Gaussians dataset over time. As seen in Figure 8, both of them approximately converge to the real density but notice that in case of improved WGAN, the discriminator function is more constrained throughout the training period. .



Improved WGAN



Our algorithm

Figure 8: Comparing the performance of different algorithms on 8-Gaussians dataset. Orange is real samples, green is generated samples and the contour plot represents $D_{\boldsymbol{\theta}}(x)$ function

2.4 Robustness across DCGAN Architecture Variations

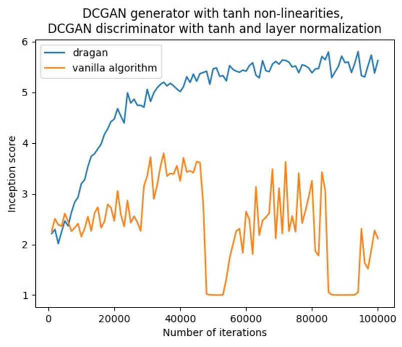
DCGANs have to be designed following specific guidelines to make them stable [1]. We restate the suggested rules here.

1. Use all convolutional networks which learn their own spatial downsampling (discriminator) or upsampling (generator)
2. Remove fully connected hidden layers for deeper architectures
3. Use batch normalization in both the generator and the discriminator
4. Use ReLU activation in the generator for all layers except the output layer, which uses $tanh$

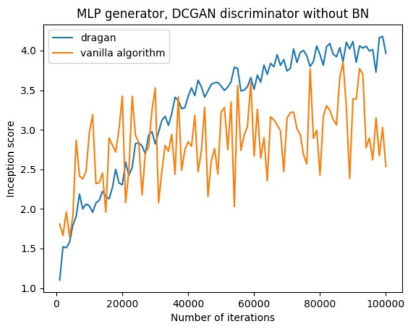
5. Use LeakyReLU activation in the discriminator for all layers

We show that such constraints can be relaxed for our algorithm and hence, practitioners are free to choose from a more diverse set of architectures. Below, we present a series of experiments in which we remove different stabilizing components from DCGAN architecture and analyze the performance of our algorithm. In each case, our algorithm is stable while the vanilla procedure fails. A similar approach is used to establish the robustness of training procedures in [3, 4]. Note that we add layer normalization to the discriminator (in our architecture experiments) in place of batch normalization. We chose the following four architectures which are difficult to train (in each case, we start with base DCGAN architecture and apply the changes) -

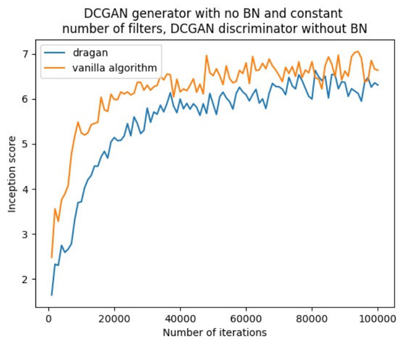
- No BN and a constant number of filters in the generator
- 4-layer 512-dim ReLU MLP generator
- $tanh$ nonlinearities everywhere
- $tanh$ nonlinearity in the generator and 4-layer 512-dim LeakyReLU MLP discriminator



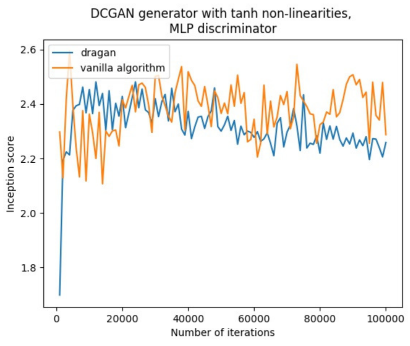
(a) tanh activation



(b) FC generator



(c) Constant filter

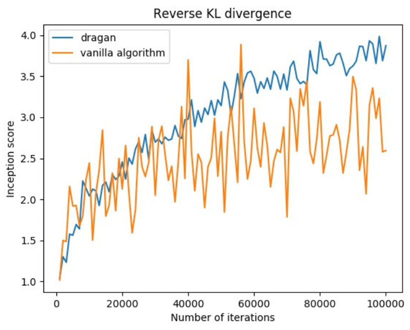


(d) FC discriminator

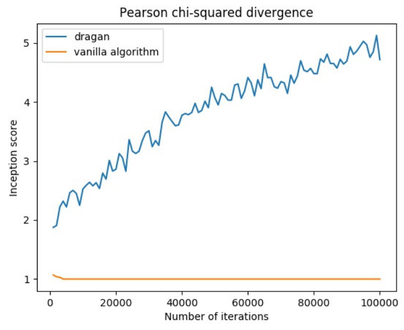
Figure 9: Inception performance on variations of DCGAN architectures.

2.5 Robustness across Divergence Measures (Complete)

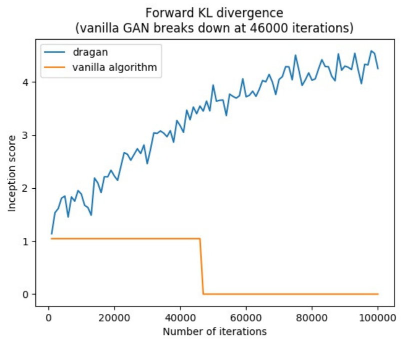
Due to space limitations, we only showed two examples for varying the divergence measures. Below we show results for all five measures.



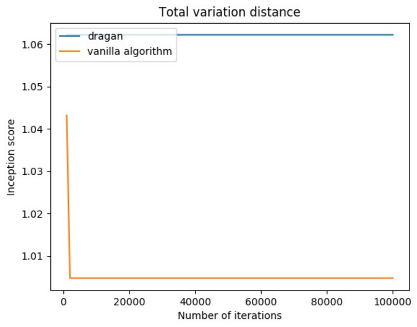
(a) Reverse KL



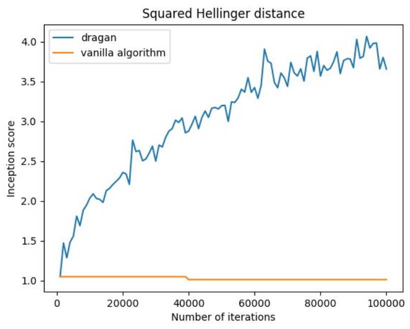
(b) Pearson χ^2



(c) Forward KL



(d) Total Variation



(e) Squared Hellinger

Figure 10: Inception score curves for five divergence measures, demonstrating superior stability for our algorithm in all but one case.

3 BogoNet Details

First, we choose from three families of architectures with probabilities - DCGAN (0.6), ResNet (0.2), MLP (0.2). Next, we further parameterize each family to create additional variation. For instance, the DCGAN family can result in networks with or without batch normalization, have LeakyReLU or Tanh non-linearities. The number and width of filters, latent space dimensionality are some other variations. Similarly, the number of layers and hidden units in each layer for MLP are chosen randomly. For

ResNets, we choose their depth randomly. This creates a set of hard games which test the stability of a given training algorithm.