Stochastic Variance Reduced Gradient Optimization of Generative Adversarial Networks

Tatjana Chavdarova [1 2] **Sebastian Stich** [2] **Martin Jaggi** [2] **François Fleuret** [1 2]

Abstract

Generative Adversarial Networks (GANs) are a class of algorithms that produce deep generative models. The quality of the generated samples as well as the algorithm's time efficiency–in training and inference–made GAN widely applied. However, this class of methods also earned a reputation for being notoriously difficult to train.

We consider Stochastic Variance Reduced Gradient (SVRG) methods for optimizing GANs. With marginal additional computation per parameter update, SVRG-GAN substantially improves the stability and the final performance of the algorithm on less challenging datasets and avoids the mode collapse problem.

1. Introduction

Generative Adversarial Networks (Goodfellow et al., 2014) are a class of unsupervised generative algorithms that take advantage of deep learning technologies. Given samples from a fixed data distribution p_d, a GAN learns a generative neural network whose distribution p_g mimics p_d and is fast to sample from. The samples' quality, the inference speed, and the ability to learn from unlabeled data all contributed to GAN being a popular choice for multiple applications: image synthesis (Goodfellow et al., 2014; Berthelot et al., 2017), super resolution (Ledig et al., 2017), image to image translation (Isola et al., 2017), text to image synthesis (Zhang et al., 2017; Reed et al., 2016), image inpainting (Yeh et al., 2017; Pathak et al., 2016), semi-supervised tasks (Salimans et al., 2016), 3D object reconstruction (Choy et al., 2016) and others.

The core components of GAN are two deep neural networks: The *generator* represents a mapping $G\colon z \mapsto x$, where z follows a known distribution p_z such as uniform, and x follows (ideally) the distribution p_d of the data. The *discriminator* is a classifier $D\colon x \mapsto y \in [0,1]$, whose output represents an estimated probability that x originates from the real dataset. Given "real" samples $x \sim p_d$ and "fake" ones $x \sim p_g$, D is trained to distinguish between the two, using "implicit" labels corresponding to the true origins of the samples. G is trained so as to "fool" D that its samples $G(z)$ are real: it aligns $D(G(z))$ with the label used for real samples. D in effect acts as a loss to train the generative model G, which is notably less restrictive compared to classical reconstruction loss. The discriminative loss directly models what we aim at–generating realistic looking samples, in contrast to replicating those we have.

One way to formalize the above involves the classical *minimax* game between two players (Goodfellow et al., 2014):

$$\min_G \max_D \mathop{\mathbb{E}}_{x\sim p_d} [\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_z} [\log(1-D(G(z)))]. \quad (1)$$

The Nash equilibrium of Eq. 1, corresponds to the modelled distribution fully recovering the real distribution $p_g = p_d$, derived in functional space (Goodfellow et al., 2014). Given an optimal discriminator, the loss function for G reveals the Jensen Shanon divergence between the targeted and the modelled distribution $D_{JS}(p_d||p_g)$ (see (Goodfellow et al., 2014), §4.1).

In practice, GANs are often reported as difficult to optimize (Arjovsky & Bottou, 2017; Mescheder et al., 2017). Several empirically obtained techniques for improving the training stability are applied by practitioners (see (Radford et al., 2015) for generating images). In particular, the training may fail under some choices, perform poorly, or performances may oscillate through the iterative procedure of the training. A typical failure is that the generator produces samples from a strict *subset* of the full dataset, which is referred to as *mode collapse*.

As p_g and p_d lie in low dimensional manifolds (Arjovsky & Bottou, 2017), it is unlikely that these would overlap in high dimensions (e.g. two planes are unlikely to overlap in $3D$) (see Arjovsky & Bottou, 2017). The parameters θ_G and θ_D of the two deep neural networks G and D respectively, are optimized iteratively. Recall that D_{JS} is constant if the two distributions have non-overlapping supports. In

[1]Idiap Research Institute [2]École Polytechnique Fédérale de Lausanne. Correspondence to: <firstname.lastname@{idiap[1],epfl[2]}.ch>.

Presented at the *International Conference on Machine Learning–ICML* 2018 workshop on *Theoretical Foundations and Applications of Deep Generative Models*.

turn, the gradient of D_{JS} would not allow for optimizing the networks $G(z;\theta_G)$ and $D(x;\theta_D)$. This line of thought motivated series of works that consider an alternative distance to D_{JS}, such as the Wasserstein distance (Arjovsky et al., 2017; Gulrajani et al., 2017)(see § 2), or the Cramer distance (Bellemare et al., 2017).

Note from Eq. 1 how the slope of the generator's loss to minimize $\log(1 - D(G(z)))$ at initial iterations would be close to zero as D would provide high confidence estimates in favor of correctly classifying its input. Indeed, this will cause the generator's gradient to vanish. Observe that, $|\nabla \log(D(G(z)))| \gg |\nabla \log(1 - D(G(z)))|$. Hence, the implementations of the vanilla-GAN algorithm (Goodfellow et al., 2014; Radford et al., 2015) instead used a "non-saturating" loss (as referred by Goodfellow): the generator, given a fixed D, aims at *maximizing*:

$$\mathscr{L}_G(G, D) = \max_G \mathop{\mathbb{E}}_{z\sim p_z} [\log(D(G(z)))]. \qquad (2)$$

To our knowledge, the generator loss shown in Eq. 2 is used in all the implementations of the vanilla GAN algorithm. The theoretical justifications of the algorithm in function space motivate D to be trained up to convergence at each iteration of the algorithm. This is in contrast to the empirical results that show that doing so has either no benefit or worsens the performances (Arjovsky & Bottou, 2017). This discrepancy is also addressed in the initial implementations of vanilla GAN (Goodfellow et al., 2014; Radford et al., 2015), as D is updated once per iteration (hence, as many times as G). Because of this, it is not easy to see that the gradients would vanish and whether replacing the divergence would help.

In this paper, we empirically replicate this problem. The results show that *in case of non-overlapping supports the generator's gradient does not vanish* and the de-facto implementation of vanilla GAN (which uses the non-saturating loss–Eq.2) continues to converge. The results give rise to an important question: *what are the key problems of the alternating optimization* of D and G reported by practitioners. Addressing this and improving accordingly would be useful because training can be notoriously difficult and the performance evaluation is time-consuming too.

Finally, to our knowledge GANs perform relatively well on more "simplistic" datasets (of low sample diversity, *e.g.* MNIST (Lecun & Cortes) or SVHN (Netzer et al., 2011)– Street View House Numbers). However, performances deteriorate on datasets with high diversity such as ImageNet (Russakovsky et al., 2015) or (equivalently) require a large number of iterations for the algorithm to converge. From classical classification problems, we know that low and high sample variance implicate fast (almost linear) and slow (sublinear) convergence of *Stochastic (Mini-batch) Gradient Descent* which is the de-facto method for optimizing neural networks. Motivated by this, in this paper we point out that *the stochastic optimization of this algorithm could be an important cause of the training difficulties*.

Empirically, we first present results which indicate: (i) in case of non-overlapping supports (under typical experimental set-up) the non-saturating version of the Jensen Shannon divergence yet converges; and (ii) the stochastic optimization is an important potential cause of mode collapse as well as slow convergence on datasets with high sample diversity. Regarding the latter, we conduct vanilla Gradient Descent training of GANs on MNIST (Lecun & Cortes), as well as larger batch size experiments on CIFAR10 (Krizhevsky, 2009) and ImageNet (Russakovsky et al., 2015).

We propose Stochastic Variance Reduced Gradient (Johnson & Zhang, 2013) optimization of GANs, and we show empirical results in favor of it. We observe that this approach offers three major advantages: (i) the mode collapse problem is naturally handled as the generated samples at each iteration are of higher diversity compared to classical training; (ii) the performances do not oscillate; and (iii) the training saturates when an optimum is reached, as the gradient updates vanish. The latter is not the case if SGD is used, as the generator may start to diverge, making it difficult to determine the number of iterations of the algorithm. However, on more challenging datasets, this approach is more prominent to converging to local optima, where the training saturates, and the generated samples are of a lower quality. These results indicate a *need of stochastic variance reduced optimization methods that perform well in a non-convex setting*, as we later discuss.

The organization is as follows. We review related works in § 2. After briefly summarizing Stochastic Variance Reduced Optimization (Johnson & Zhang, 2013) § 3, we propose an extension of it to GANs, § 4. The experiments presented in § 5 first motivate SVRG-based optimization and then list preliminary results of SVRG optimization of GANs. Directions for extensions are discussed in § 6.

2. Related works

GAN variants. When one uses the GAN framework to train a model that generates *images*, particular Convolutional Neural Network (CNN) architectures are used for G and D, called the Deep Convolutional Generative Adversarial Networks (**DCGAN**) (Radford et al., 2015). Also, common implementations of GAN often follow several gudelines enumerated by Radford et al., as these could be critical.

To train G, the mapping represented by D, besides being differentiable, shall have non-zero gradient. To ensure this holds when the supports of p_g and p_d are disjoint (see also § 1 or (Arjovsky et al., 2017) for details) D_{JS} can be re-

placed by the Wasserstein distance (Villani, 2008), which accounts for the Euclidean structure–a variant called Wasserstein GAN (**WGAN**) (Arjovsky et al., 2017). Interestingly, the computationally tractable version of it, based on the Kantorovich Rubinstein duality principle (Villani, 2008), requires a K-Lipschitz discriminator (here called *critic* as its output is no longer a probability estimate). The result is intuitive from a practical standpoint: strongly regularizing the discriminator prevents the gradient from vanishing through it.

A computationally cheap way of enforcing the Lipschitz continuity is through weight clipping (Arjovsky et al., 2017). However, this is shown to degrade performances (Gulrajani et al., 2017). Alternatively, it can be implemented in a smooth manner by adding an extra term in D's loss so that gradients whose norm is higher then K are penalized–a variant called WGAN with Gradient Penalty (**WGAN-GP**) (Gulrajani et al., 2017). Note how the latter is computationally more demanding as of the need to compute second order derivative.

Motivated by game theory principles, Kodali et al. derive similar solution. Based on the vanilla GAN loss, the proposed algorithm named Deep Regret Analytic GAN (**DRAGAN**) forces the constraint on the gradients of $D(x)$ solely in local regions *around real samples*.

Spectral Normalization GAN (**SNGAN**) (Miyato et al., 2018) obtains Lipschitz discriminator using the power iteration method to normalize the parameters of the discriminator, and uses the original D_{JS} divergence.

GAN optimization. Mescheder et al. and Nagarajan & Kolter discuss the algorithm's convergence and stability through the Jacobian of the vector field that corresponds to the parameter updates. Both consider *simultaneous* gradient ascent/descent, *i.e.* at each iteration D and G are updated *at the same time*. Mescheder et al. points out that simultaneous gradient *ascent* may require *infeasibly small* learning rates for convergence, when the eigenvalues of the Jacobian have small/no real part and big imaginary part (also pointed out by Alain & Bengio–§ 3.6 in the context of autoencoders). It is not immediately clear if this problem arises if doing *alternating* gradient descent, which is the optimization method used in practice. Importantly, Nagarajan & Kolter investigate if GAN and the WGAN variant are stable *at the equilibria*. Under certain assumptions, Nagarajan & Kolter show that vanilla-GAN *is* locally exponentially stable if performing simultaneous gradient descent optimization.

Multi-Network approaches. Several works report improved stability when using either multiple generators versus single discriminator (Ghosh et al., 2017; 2016), or multiple discriminators versus one generator (Durugkar et al., 2017). Wang et al. propose building a "self-ensemble" out of copies of the generator taken at different iterations while training a single pair. Chavdarova & Fleuret train a "global" pair against a statistically independent ensemble of adversarial pairs. These works are motivated by the high variance of the updates and the oscillations observed in practice.

Algorithm 1 Pseudocode for SVRG.

Input: dataset $\mathcal{D}$, epochs E, update frequency m, learning rate η, objective function $\mathscr{L}$
Initialize: Θ
for $e = 0$ **to** $E-1$ **do**
  if $e\%m == 0$ **then**
    $\Theta^{\mathcal{S}} = \Theta$
    $\mu = \frac{1}{|\mathcal{D}|}\sum_{n=1}^{|\mathcal{D}|} \nabla_{\Theta^{\mathcal{S}}} \mathscr{L}(\mathcal{D}[n])$
  end if
  for $i = 0$ **to** $|\mathcal{D}|-1$ **do**
    $n \sim U(1, |\mathcal{D}|)$
    $\Theta = \Theta - \eta\big(\nabla_{\Theta} \mathscr{L}(\mathcal{D}[n]) - \nabla_{\Theta^{\mathcal{S}}} \mathscr{L}(\mathcal{D}[n]) + \mu\big)$
  end for
end for
Output: Θ

3. Stochastic Variance Reduced Gradient

We are often interested in developing efficient algorithms that learn a model, given a finite size of annotated dataset $\mathcal{D}$. The training of the model's parameters Θ aims at:

$$\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \mathscr{L}(\mathcal{D}[i]; \Theta), \tag{3}$$

where $N = |\mathcal{D}|$, and $\mathscr{L}(\cdot)$ is a differentiable loss function. For example, a typical choice is the squared loss: $\mathscr{L}(\mathcal{D}[i], \Theta) = (f(x_i; \Theta) - y_i)^2$, where f is the function represetned by our model, and $\mathcal{D}[i] = (x_i, y_i), i = 1, \dots, N$ are training examples where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$. For clearity, we omit the parameters Θ of $\mathscr{L}(\cdot; \Theta)$.

Gradient Descent (GD) based methods are the workhorse of deep learning. The randomly initialized model parameters are iteratively updated using the gradients of $\mathscr{L}(\cdot)$: $\Theta = \Theta - \eta\big(\frac{1}{B}\sum_{i=1}^{B} \nabla_{\Theta} \mathscr{L}(\mathcal{D}[\sigma(i)])\big)$, where η is the learning rate. At each iteration, GD and Stochastic Gradient Descent (SGD) use the full dataset $\mathcal{D}$ ($B = N$, $\sigma = 1_N$) and a sample of the dataset $x \sim U(\mathcal{D})$ ($B = 1$, σ is a random permutation of $[1, N]$), respectively. While the former (deterministic) optimization requires significantly larger computation per each update–$O(Nd)$, it has the advantage over SGD of a linear convergence rate.

Stochastic Variance Reduced Gradient (SVRG) (Johnson & Zhang, 2013) is an optimization method which speeds up SGD through reducing the variance of the updates of

Algorithm 2 Pseudocode for *alternating* GAN.

Input: dataset $\mathcal{D}$, batch size B, known distribution p_z, iterations I, learning rate η, generator loss $\mathscr{L}_G$, discriminator loss $\mathscr{L}_D$
Initialize: $D(\cdot;\Theta_D)$, $G(\cdot;\Theta_G)$
for $e = 1$ **to** I **do**
  $x_1 \dots x_B \sim U(\mathcal{D})$, $z_1 \dots z_B \sim p_z$
  $\Theta_D = \Theta_D - \eta\nabla_{\Theta_D}\mathscr{L}_D(D, G, \mathbf{x}, \mathbf{z})$
  $z_1 \dots z_B \sim p_z$
  $\Theta_G = \Theta_G - \eta\nabla_{\Theta_G}\mathscr{L}_G(D, G, \mathbf{z})$
end for
Output: Θ

the model parameters, and provably converges under strong convexity and smoothness assumptions. As SGD, SVRG iteratively updates Θ using a sample of the dataset of size B, where $0 < B \ll |\mathcal{D}|$.

Alg. 1 summarizes the SVRG algorithm, where for simplicity $B = 1$. At particular iterations (with a predefined frequency m), SVRG computes a vector $\mu \in \mathbb{R}^{|\Theta|}$ as an average of the gradients of the full dataset $\mathcal{D}$ with respect to Θ. It stores a copy of the model's parameters at the point μ is calculated, denoted as $\Theta^{\mathcal{S}}$, where $\mathcal{S}$ stands for *snapshot*. Finally, each update of Θ is done using a randomly picked sample of the dataset $n \sim U(\mathcal{D})$, and is calculated as $\nabla_\Theta\mathscr{L}(\mathcal{D}[n]) - \nabla_{\Theta^{\mathcal{S}}}\mathscr{L}(\mathcal{D}[n]) + \mu$. Note how the two extra terms vanish in expectation, what explains the need of storing $\Theta^{\mathcal{S}}$. Under the identical assumptions, SVRG converges at a *linear* rate (Johnson & Zhang, 2013).

4. Stochastic Variance Reduced Gradient optimization of GAN

GAN optimization. The most common implementation of Eq. 1 is the *alternating* optimization of G and D, see Alg. 2. In Alg. 2, for vanilla-GAN we have $\mathscr{L}_D(D, G, \mathbf{x}, \mathbf{z}) = \log(D(\mathbf{x})) + \log(1 - D(G(\mathbf{z})))$ and $\mathscr{L}_G(D, G, \mathbf{z}) = \log(D(G(\mathbf{z})))$. Observe that contrary to classical training of the form Eq. 3 where we minimize a single training criterion, in Alg. 2 we optimize the two networks stochastically, and each affects the opponent optimization in the next iteration. This leads to optimization *trajectories* that may largely differ based on the sampling.

SVRG-GAN. Alg. 3 summarizes the SVRG optimization extended to GAN. To obtain that $\mathbb{E}\left[\nabla_{\Theta^{\mathcal{S}}}\mathscr{L}(D^{\mathcal{S}}, G^{\mathcal{S}}, \cdot) - \mu\right]$ vanishes, when updating Θ_D and Θ_G where the expecation is over samples of $\mathcal{D}$ and $\mathcal{Z}$ respectively, we use the snapshopt networks $D^{\mathcal{S}}$ and $G^{\mathcal{S}}$ for the second term in lines 12 & 14. Moreover, the noise dataset $\mathcal{Z} \sim p_z$, where $|\mathcal{Z}| = |\mathcal{D}|$, is fixed. Nonetheless, sampling from p_z should not impact the performance, as $|\mathcal{Z}|$ is usually high.

Algorithm 3 Pseudocode for SVRG-GAN.

1: **Input:** dataset $\mathcal{D}$, noise dataset $\mathcal{Z}$ ($|\mathcal{Z}| = |\mathcal{D}|$), epochs E, update frequency m, learning rate η, generator loss $\mathscr{L}_G$, discriminator loss $\mathscr{L}_D$
2: **Initialize:** $D(|x \sim \mathcal{D}|;\Theta_D)$, $G(z;\Theta_G)$
3: **for** $e = 0$ **to** $E-1$ **do**
4: **if** $e\%m == 0$ **then**
5: $\Theta_D^{\mathcal{S}} = \Theta_D$
6: $\Theta_G^{\mathcal{S}} = \Theta_G$
7: $\mu_D = \frac{1}{|\mathcal{D}|}\sum_{n=1}^{|\mathcal{D}|}\nabla_{\Theta_D^{\mathcal{S}}}\mathscr{L}_D(D^{\mathcal{S}}, G^{\mathcal{S}}, \mathcal{D}[n_d], \mathcal{Z}[n])$
8: $\mu_G = \frac{1}{|\mathcal{Z}|}\sum_{n=1}^{|\mathcal{Z}|}\nabla_{\Theta_G^{\mathcal{S}}}\mathscr{L}_G(D^{\mathcal{S}}, G^{\mathcal{S}}, \mathcal{Z}[n])$
9: **end if**
10: **for** $i = 0$ **to** $|\mathcal{D}|-1$ **do**
11: $n_d \sim U(1, |\mathcal{D}|)$, $n_z \sim U(1, |\mathcal{Z}|)$
12: $\Theta_D = \Theta_D - \eta\big(\nabla_{\Theta_D}\mathscr{L}_D(D, G, \mathcal{D}[n_d], \mathcal{Z}[n_z]) - \nabla_{\Theta_D^{\mathcal{S}}}\mathscr{L}_D(D^{\mathcal{S}}, G^{\mathcal{S}}, \mathcal{D}[n_d], \mathcal{Z}[n_z]) + \mu_D\big)$
13: $z \sim U(1, |\mathcal{Z}|)$
14: $\Theta_G = \Theta_G - \eta\big(\nabla_{\Theta_G}\mathscr{L}_G(D, G, \mathcal{Z}[n_z]) - \nabla_{\Theta_G^{\mathcal{S}}}\mathscr{L}_G(D^{\mathcal{S}}, G^{\mathcal{S}}, \mathcal{Z}[n_z]) + \mu_G\big)$
15: **end for**
16: **end for**
17: **Output:** Θ

5. Experiments

Unless otherwise ephasized, our experiments use vanilla-GAN. In this case, always the *non staurating* loss for the generator (Eq. 2) is used. See App. A for details on our implementation.

Datasets. We used few datasets: (i) **MNIST** (Lecun & Cortes) and **FASHION-MNIST** (Xiao et al., 2017), (ii) **CIFAR10** (Krizhevsky §3), (iii) **ImageNet** (Russakovsky et al., 2015).

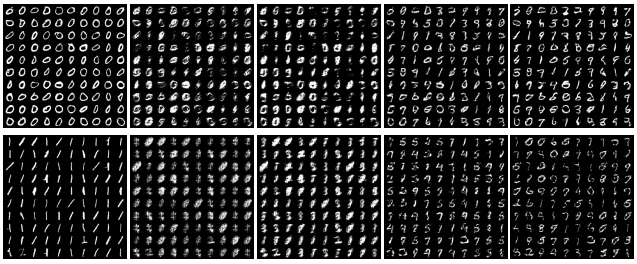


Figure 1: GD-GAN *mode recovery* experiment (see § 5.1) on **MNIST**. For the first step we used the digits *0* and *1*, shown in the first and second row, respectively. Left–to–right columns show samples taken at the iterations: 1, 50, 100, 500, and 1000, respectiely.

Metrics. A limitation to improve GANs is the lack of a proper performance evaluation (Lucic et al., 2017). Ideally, the model should generate samples that do not necessarily exist in the dataset at hand, which yet "look alike" these. Note how this is highly subjective, what implies that even an evaluation made by humans could be noisy.

In case of image synthesis, most widely adopted is the **Inception score** (IS) (Salimans et al., 2016). This metric feeds a pre-trained model named *Inception* (Szegedy et al., 2015) with a large sample of generated images and measures the Kullback–Leibler divergence between the predicted conditional label distribution and the actual class probability distribution. The mode collapse failure is reflected by the mode's class being less frequent, making the conditional label distribution more deterministic. As the *Inception* network operates on an input of 3 channels (RGB), for the 1 channel MNIST dataset, we replace the module with a classifier trained on this dataset. For datasets of 3 input channels, we use the original implementation of the Inception Score (Salimans et al., 2016) and a sample of p_g of size $50{\cdot}10^3$, whereas for the latter case we use our own implementation in PyTorch (Paszke et al., 2017).

The **Fréchet Inception Distance** (FID) (Heusel et al., 2017) also relies on the *Inception* model (Szegedy et al., 2015), but uses it to embed samples into a "good" feature space. It consists of first estimating the means μ_g and μ_d, and covariances C_g and C_d, respectively for p_g and p_d in that feature space, and computing $D_{FID}(p_d, p_g) \approx d^2((\mu_d, C_d), (\mu_g, C_g)) = ||\mu_d - \mu_g||_2^2 + Tr(C_d + C_g - 2(C_d C_g)^{\frac{1}{2}})$, where d^2 denotes the Fréchet Distance.

For experiments on MNIST, using an independently trained classifier distinguishing its 10 classes, we also plot the **entropy** of the generated samples' mode distribution, as well as the **total variation** between the class distribution of the generated samples and a uniform one.

5.1. Disjoint supports

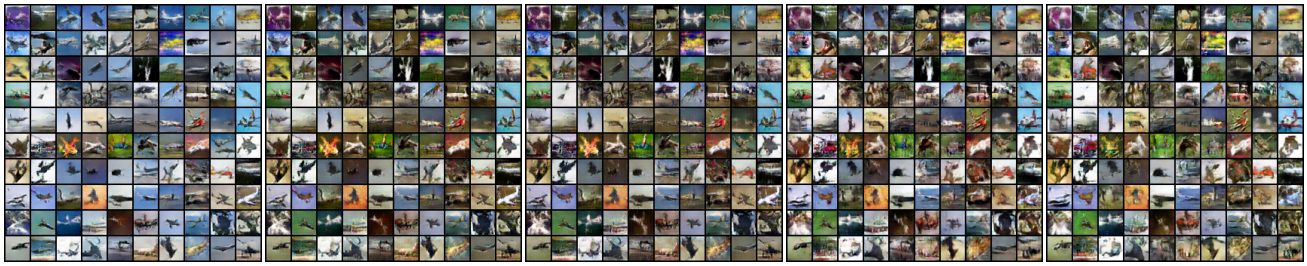
To assess if the learning saturates for the generator in case of non-overlapping supports we conduct the following two-step experiments: (i) enforcing mode collapse using a subset of the dataset, and (ii) full-dataset training. The former step, which uses a *single* class of the dataset, produces G_c & D_c. The latter starts from G_c & D_c and trains these networks using the full dataset. The second step was carried out in: (i) standard GAN training set-up: training stochastically using *Adam*–denoted as *SGD-GAN*; and (ii) using vanilla GD optimization–denoted as *GD-GAN*. The latter ensures that the obtained results are consistent, in a sense that these do not depend on the random sampling and is more rigorous. We call this experiment *mode recovery*.

Figures 1 and 2 illustrate the results on MNIST and CIFAR10. We observe that the learning does *not* saturate, and the non-saturating GAN continues to converge. Note that this is not the case if the original D_{JS} is used. Moreover, we experimented with large learning rate for the discriminator that is *not* used in practice, and we observe that in this case, the learning *does* saturate as the loss quickly goes to 0. In summary, the de-facto implementation of GAN is able to recover gradients and the learning continues.

5.2. Larger batch size and Gradient Descent optimization

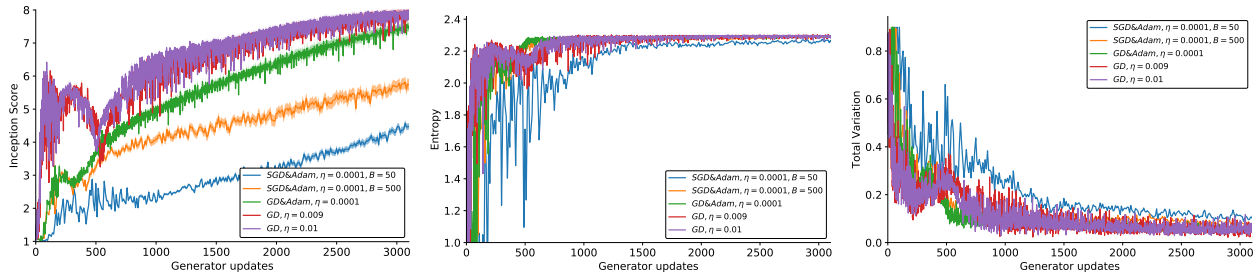
We conduct preliminary experiments that may point out if variance reduced optimization could improve performances. To this end, we used vanilla GD optimization on MNIST. As GD is infeasible for larger datasets, we conducted experiments using varying batch size on two datasets: CIFAR10 and ImageNet.

Fig. 3 quantitatively compares the performances obtained on MNIST. Primarily, the results show that increasing the batch size improves performances. It is important to note that the x-axis of Fig. 3 shows the number of parameters' updates, hence each iteration of GD consumes significantly more computation compared to cases where $B \ll |\mathcal{D}|$. Besides the expected results that gradient descent converges faster in terms of the number of updates of the parameters, we observe from Fig. 3b that GD training on this dataset impressively avoids mode collapse, as the across class dis-



initial fake samples | 1000-*th* iteration, GD | 2000-*th* iteration, GD | 1000-*th* iteration, *Adam*, B=64 | 2000-*th* iteration, *Adam*, B=64

Figure 2: *Mode recovery* experiment (see § 5.1) on **CIFAR10**. Selected class: *airplane*.



(a) IS (higher is better) (b) Entropy (higher is better) (c) Total variation (lower is better)

Figure 3: Different optimization methods of GAN on **MNIST** (best seen in color). *SGD* emphasizes that the optimization is stochastic, in which case we use *Adam* (Kingma & Ba, 2014). η and B denote learning rate and batch size, respectively. The input space is $1\times28\times28$. See App. A for implementational details.

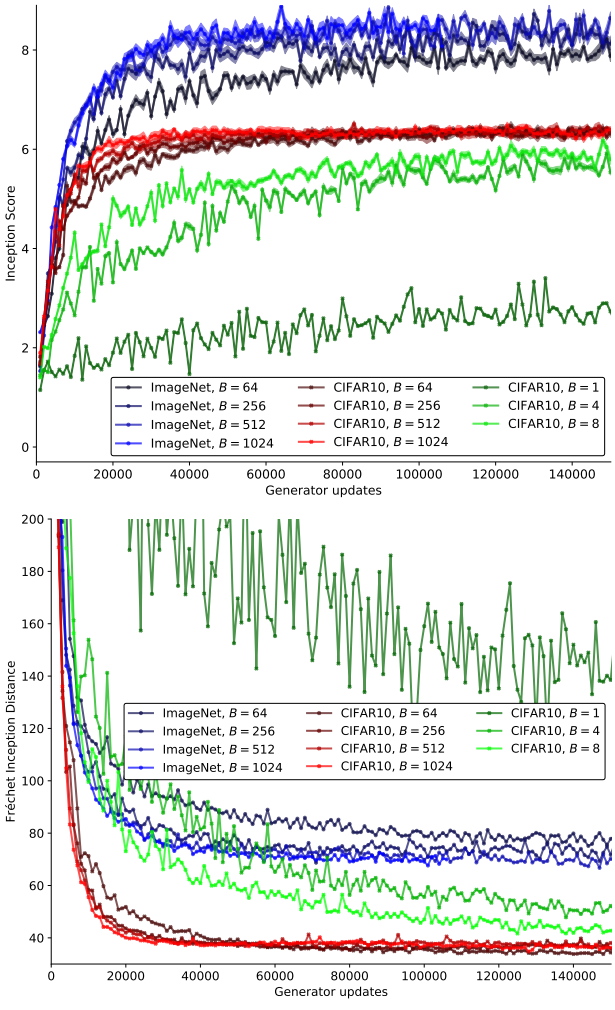


Figure 4: Top: IS (higher is better), and bottom: FID (lower is better) on **ImageNet** and **CIFAR10** (best seen in color). Using **Adam**, η=0.0001 and different batch sizes B.

50-Adam, η=10^{-4} | *500-Adam*, η=10^{-4} | *GD-Adam*, η=10^{-4} | *GD*, η=10^{-3} | *GD*, η=10^{-2}

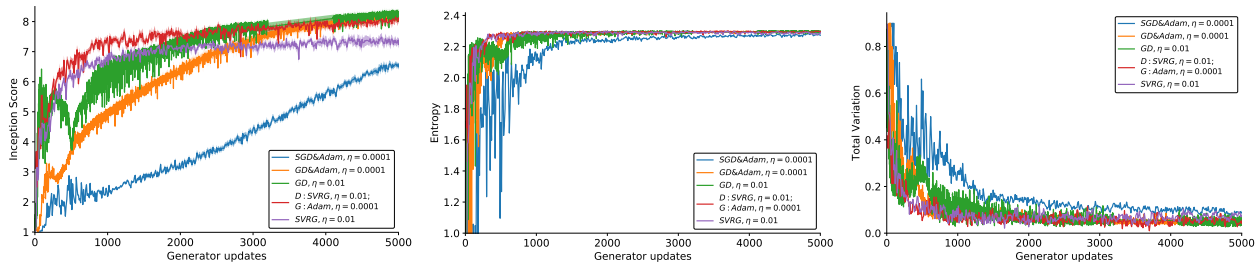
Figure 5: Random sample of p_g of the experiments illustrated in Figure 3. The samples are taken at the 5000-*th* iteration of each of the experiments. [B-] followed by the optimization method denotes the minibatch size.

tribution of the generated samples is uniform. Fig. 5 is complementary and shows that the metrics in Figure 3 are in line with the observed quality and diversity of the generated samples.

Fig. 4 shows quantitative results on ImageNet and CIFAR10 for different batch size, obtained using the IS and FID metrics. Increasing the batch sizes improves performances for ImageNet, which has 1000 classes. The performance gain is smaller for CIFAR10 when varying the batch sizes between $64 - 1024$, as this dataset has 10 classes. However, if we reduce the batch size to similar ratio C/B where C denotes the number of classes, as for ImageNet, we observe that performances drastically decrease. Moreover, if using $B = 1$ the algorithm does *not* converge, which experiment we run for $900,000$ iterations.

5.3. SVRG-GAN

In Fig. 6 we plot the Inception Score, Entropy and Total Variation metrics (see § 5) of SVRG-GAN on MNIST. Interestingly, SVRG-GAN is comparable to GD-GAN, while the latter is significantly more computation demanding. Figures 7 and 8 show samples generated from SVRG-GAN, on MNIST and FASHION-MNIST, respectively.

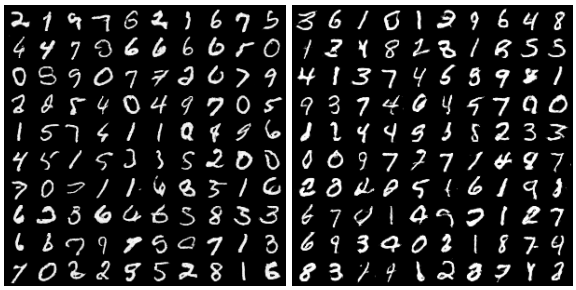


(a) IS (Salimans et al., 2016) (higher is better)

(b) Entropy (higher is better)

(c) Total variation (lower is better)

Figure 6: SVRG-GAN experiments on **MNIST** (best seen in color). Except for GD-GAN, B=50 for the rest of the experiments.



(a) D: SVRG, η=0.01; G: Adam, η=0.0001;

(b) SVRG, η=0.01

Figure 7: Generated samples on **MNIST**, using SVRG.

Figure 8: Generated samples on **FASHION-MNIST**, using SVRG $\eta = 0.0001$, at the 30000-*th* iteration.

6. Conclusion

The feasible way of optimizing the two deep neural networks tied as opponents – stochastically, introduces variance for the parameter updates. As the generator represents a mapping from latent code to the space of the real data, a high variance may cause large oscillations and slow convergence, as well as imbalance or repetition of how the modes of the real data are mapped to the latent codes. This line of reasoning was confirmed empirically, as vanilla Gradient Descent GAN optimization on MNIST demonstrated no mode collapse and increasing the size of the mini-batches improves performances on more challenging datasets.

We proposed an SVRG extension to GANs, and we showed that such extension greatly outperforms *Adam* in terms of convergence speed and reducing mode collapse. In this paper, we focused on vanilla SVRG which provably converges under strong convexity assumption. Nonetheless, there exists multiple variants of this method (Allen-Zhu, 2016; Shang et al., 2018) which on the theory side require less strict assumptions, and practically outperform vanilla SVRG on more challenging non-convex problems. Our future work is towards addressing key properties required for variance reduced optimization of GANs on challenging datasets.

More precisely, we observe two main drawbacks: (i) sensitivity to the choice of the initial learning rate, as well as (ii) convergence halt due to snapshot networks which are off the mean parameter values in the current epoch. The latter occurs as GAN optimization is significantly more challenging than the classical one, as the dynamics constantly change (see App.). The above can be addressed using momentum based adaptation of the learning rate, and using an average of the past parameter values for the snapshot network (as Shang et al. recently propose), respectively.

Acknowledgements

This work was supported by the Swiss National Science Foundation, under grant CRSII2-147693 “WILDTRACK” and the Hasler Foundation through the “MEMUDE” project.

References

Alain, G. and Bengio, Y. What Regularized Auto-Encoders Learn from the Data Generating Distribution. *ArXiv e-prints*, November 2012.

Allen-Zhu, Z. Katyusha: The First Direct Acceleration of Stochastic Gradient Methods. *ArXiv e-prints*, March 2016.

Arjovsky, M. and Bottou, L. Towards Principled Methods for Training Generative Adversarial Networks. *ArXiv e-prints*, January 2017.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 214–223, 2017. URL http://proceedings.mlr.press/v70/arjovsky17a.html.

Bellemare, M. G., Danihelka, I., Dabney, W., Mohamed, S., Lakshminarayanan, B., Hoyer, S., and Munos, R. The Cramer Distance as a Solution to Biased Wasserstein Gradients. *ArXiv e-prints*, May 2017.

Berthelot, D., Schumm, T., and Metz, L. BEGAN: Boundary Equilibrium Generative Adversarial Networks. *ArXiv e-prints*, March 2017.

Chavdarova, T. and Fleuret, F. SGAN: An alternative training of generative adversarial networks. In *Proceedings of the IEEE international conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

Durugkar, I. P., Gemp, I., and Mahadevan, S. Generative multi-adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Ghosh, A., Kulharia, V., and Namboodiri, V. P. Message passing multi-agent GANs. *CoRR*, abs/1612.01294, 2016. URL http://arxiv.org/abs/1612.01294.

Ghosh, A., Kulharia, V., Namboodiri, V. P., Torr, P. H. S., and Dokania, P. K. Multi-agent diverse generative adversarial networks. *CoRR*, abs/1704.02906, 2017. URL http://arxiv.org/abs/1704.02906.

Goodfellow, I. NIPS 2016 Tutorial: Generative Adversarial Networks. *ArXiv e-prints*, December 2017.

Goodfellow, I., 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*, pp. 2672–2680. 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pp. 5767–5777. 2017.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30*, pp. 6626–6637. 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 315–323. Curran Associates, Inc., 2013.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Kodali, N., Abernethy, J. D., Hays, J., and Kira, Z. How to train your DRAGAN. *CoRR*, abs/1705.07215, 2017. URL http://arxiv.org/abs/1705.07215.

Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009. URL http://www.cs.toronto.edu/~{}kriz/learning-features-2009-TR.pdf.

Lecun, Y. and Cortes, C. The MNIST database of handwritten digits. URL http://yann.lecun.com/exdb/mnist/.

Ledig, C., Theis, L., Huszr, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 105–114, July 2017. doi: 10.1109/CVPR.2017.19.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. Are GANs Created Equal? A Large-Scale Study. *ArXiv e-prints*, November 2017.

Mescheder, L., Nowozin, S., and Geiger, A. The numerics of GANs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 1825–1835. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6779-the-numerics-of-gans.pdf.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1QRgziT-.

Nagarajan, V. and Kolter, J. Z. Gradient descent GAN optimization is locally stable. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5585–5595. Curran Associates, Inc., 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 *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

Paszke, A., Gross, S., Chintala, S., and Chanan, G. PyTorch. https://github.com/pytorch/pytorch, 2017.

Pathak, D., Krähenbühl, P., Donahue, J., Darrell, T., and Efros, A. Context encoders: Feature learning by inpainting. 2016.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL http://arxiv.org/abs/1511.06434.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. Generative adversarial text to image synthesis. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1060–1069, New York, New York, USA, 20–22 Jun 2016. PMLR.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X., and Chen, X. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29*, pp. 2234–2242, 2016.

Shang, F., Zhou, K., Cheng, J., Tsang, I. W., Zhang, L., and Tao, D. VR-SGD: A simple stochastic variance reduction method for machine learning. *CoRR*, abs/1802.09932, 2018. URL http://arxiv.org/abs/1802.09932.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

Villani, C. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer, 2009 edition, September 2008. ISBN 3540710493. URL http://www.worldcat.org/isbn/3540710493.

Wang, Y., Zhang, L., and van de Weijer, J. Ensembles of generative adversarial networks. *CoRR*, abs/1612.00991, 2016. URL http://arxiv.org/abs/1612.00991.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

Yeh, R. A., Chen, C., Lim, T.-Y., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. Semantic image inpainting with deep generative models. pp. 6882–6890, 2017.

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017.

A. Implementational details

Architectures. We used architectures based on the DCGAN (Radford et al., 2015) implementation, or identical to it when using input space of 64×64.

For experiments conducted on **MNIST**, we used the original dimensions of this dataset of 28×28. Hence, we modified the DCGAN networks accordingly to this input space and reduced their depth for one layer, see Tab. 1. Similarly, for **CIFAR10** we used its original dimensions of 32×32. The implementation of DCGAN (Radford et al., 2015) uses Batch Normalization layers (Ioffe & Szegedy, 2015).

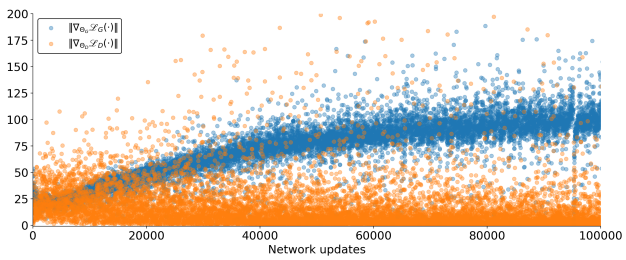
Hyperparameters. We used a batch size of 50 and 64 for (FASHION)MNIST and the rest of the datasets, respectively. When using the *Adam* optimizer (Kingma & Ba, 2014), we fixed the two hyperparameters (one parameter used for computing running averages of gradient and another for its square) to 0.5 and 0.999, as in (Radford et al., 2015).

B. Analyses

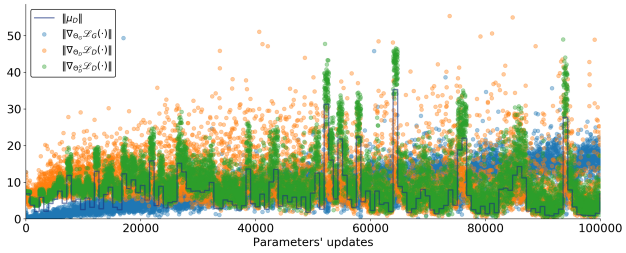
Fig. 9 depicts the ℓ^2-norm of the gradient updates using standard training, as well as SVRG training. For the latter we considered SVRG applied only to the discriminator, and full SVRG-GAN training, denoted as SVRG-Discriminator-GAN and SVRG-GAN, respectively. The performance evaluation of these experiments is given in Fig. 6. Primerely, when using *Adam* we observe that the ℓ^2-norm of the gradient updates of the discriminator and the generator largely differ, *i.e.* $\|\nabla_{\Theta_D}\mathscr{L}_D(\cdot)\| \ll \|\nabla_{\Theta_G}\mathscr{L}_G(\cdot)\|$. On the contrary, this is not the case for SVRG(-Discriminator)-GAN. Note from Fig. 6 that this is not due to slower convergence, as SVRG shows outperforming results throughout the iterations. This could indicate that the parameters of the models when using SVRG are towards the optimal one (faster convergence rate), hence the observed lower ℓ^2-norm of $\delta\Theta$.

Generator
Input: $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
transposed conv. (kernel: 3×3, $128 \rightarrow 512$; stride: 1)
Batch Normalization
ReLU
transposed conv. (kernel: 4×4, $512 \rightarrow 256$, stride: 2)
Batch Normalization
ReLU
transposed conv. (kernel: 4×4, $256 \rightarrow 128$, stride: 2)
Batch Normalization
ReLU
transposed conv. (kernel: 4×4, $128 \rightarrow 1$, stride: 2, pad: 1)
$Tanh(\cdot)$
Discriminator
Input: $x \in \mathbb{R}^{1\times28\times28}$
conv. (kernel: 4×4, $1 \rightarrow 64$; stride: 2; pad:1)
LeakyReLU (negative slope: 0.2)
conv. (kernel: 4×4, $64 \rightarrow 128$; stride: 2; pad:1)
Batch Normalization
LeakyReLU (negative slope: 0.2)
conv. (kernel: 4×4, $128 \rightarrow 256$; stride: 2; pad:1)
Batch Normalization
LeakyReLU (negative slope: 0.2)
conv. (kernel: 3×3, $256 \rightarrow 1$; stride: 1)
$Sigmoid(\cdot)$

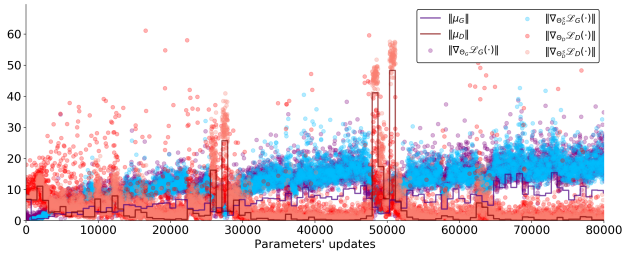
Table 1: Architectures for experiments on **MNIST**.



(a) **Vanilla GAN.** *Adam*, η=$1e{-}4$.



(b) **SVRG-Discriminator-GAN**. G: *Adam*, η=0.0001; D: *SVRG* η=0.01



(c) **SVRG-GAN**. η=0.01.

Figure 9: ℓ^2-norm of the gradient updates on **MNIST**.