SGAN: An Alternative Training of Generative Adversarial Networks

Tatjana Chavdarova and François Fleuret
Machine Learning group, Idiap Research Institute & École Polytechnique Fédérale de Lausanne
{firstname.lastname}@idiap.ch

Abstract

The Generative Adversarial Networks (GANs) have demonstrated impressive performance for data synthesis, and are now used in a wide range of computer vision tasks. In spite of this success, they gained a reputation for being difficult to train, what results in a time-consuming and human-involved development process to use them.

We consider an alternative training process, named SGAN, in which several adversarial "local" pairs of networks are trained independently so that a "global" supervising pair of networks can be trained against them. The goal is to train the global pair with the corresponding ensemble opponent for improved performances in terms of mode coverage. This approach aims at increasing the chances that learning will not stop for the global pair, preventing both to be trapped in an unsatisfactory local minimum, or to face oscillations often observed in practice. To guarantee the latter, the global pair never affects the local ones.

The rules of SGAN training are thus as follows: the global generator and discriminator are trained using the local discriminators and generators, respectively, whereas the local networks are trained with their fixed local opponent.

Experimental results on both toy and real-world problems demonstrate that this approach outperforms standard training in terms of better mitigating mode collapse, stability while converging and that it surprisingly, increases the convergence speed as well.

1. Introduction

An important research effort has recently focused on improving the convergence analysis of the Generative Adversarial Networks (GANs) [6]. This family of unsupervised learning algorithms provides powerful generative models, and have found numerous and diverse applications [11, 17, 22, 34].

Different from traditional generative models, a GAN generator represents a mapping $G : z \mapsto x$, such that if z follows a known distribution p_z, then x follows the distribution p_d of the data. Notably, this approach omits an explicit representation of $p_g(x)$, or the ability to apply directly a maximum-likelihood maximization for training. This is aligned with the practical need, which is that we do not need an explicit formulation of $p_g(x)$, but rather a mean to sample from it, preferably in a computationally efficient manner. The training of the generator involves a discriminative model $D : x \mapsto y \in [0, 1]$, whose output represents an estimated probability that x originates from the dataset.

More precisely, the algorithm consists of two training steps. Given that there is a probability 0.5 that the input x originates from the dataset and 0.5 that it was generated by G, the discriminator D is trained to distinguish between "real" and "fake" inputs, respectively. On the other hand, G is trained to "fool" D by generating synthetic samples indistinguishable from the real ones.

Formally, the two competing models play the following two-player minimax–alternatively zero-sum–game:

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

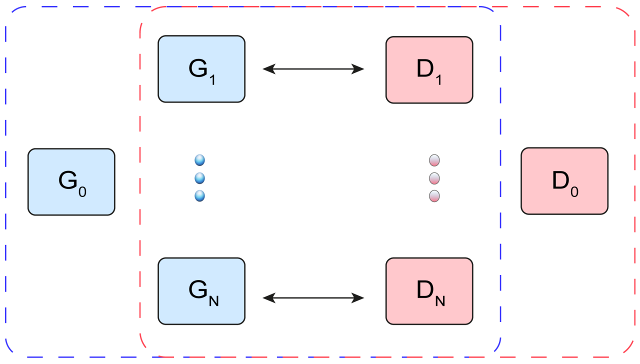


Figure 1: Conceptual illustration of SGAN. There are $N+1$ pairs, of which the pair (G_0, D_0) is not trained directly. D_0 is trained with G_i, $i=1,\ldots,N$, and G_0 is trained with D_i, $i=1,\ldots,N$, as illustrated with the dashed line rectangles.

The two models are parametrized differentiable functions $G(z; \theta_g)$ and $D(x; \theta_d)$, implemented with neural networks, whose parameters θ_g and θ_d are optimized iteratively. In functional space, the competing models are guaranteed to reach a Nash Equilibrium, in particular under the assumptions that we optimize directly p_g instead of θ_g and that the two networks have enough capacity. At this equilibria point, D outputs probability 0.5 for any input.

In practice, GANs are difficult to optimize, and practitioners have amassed numerous techniques to improve stability of the training process [25]. However, the neglected inherited problems of the neural networks such as the lack of convexity, the numerical instabilities of some of the involved operations, the limited representation capacity, as well as the problem of vanishing and exploding gradients often emerge in practice.

Current state-of-the-art GAN variants [1, 8] eliminate gradient instabilities, which mitigated the discrepancy between the theoretical requirement that D should be trained up to convergence before updating G, and the practical procedures of vanilla GAN for which this is not the case. These results are important, as vanishing or exploding gradients results in G to produce samples of noise.

However, oscillations between noisy patterns and samples starting to look like real data while the algorithm is converging, as well as failures of capturing p_d, are not resolved. In addition, in practical applications, it is very difficult to assess the diversity of the generated samples. "Fake" samples may look realistic but could be similar to each other – indicating that the modes of p_d have been only partially "covered" by p_g. This is a problem that arises and is referred as *mode collapse*. As a result, a golden rule remains that one does multiple trials of combinations of hyperparameters, architectural and optimization choices, and variants of GANs. As under different choices the performances vary [19], this is followed by tedious and subjective assessments of the quality of the generated samples in order to select a generator.

As a summary, what made GAN distinctly powerful is the opponent-wise engagement of two networks belonging to an already outperforming class of algorithms. Such a framework–the discriminator being a deep neural network–allows for time-efficient training of the generative model and directly formulates what we aim at–to generate samples that resemble those we have, in contrast to memorizing these. Intuitively, the more we enforce constraints, either architectural or functional, the better the coherency of the learning dynamics. On the other hand, this may result in reduced sample quality or increased convergence time at the minimum. A question arises if we can improve training stability, guarantees of "successful" training, and performances, without imposing restrictions on the architectures of G or D.

We propose a novel way of training a *global* pair (G_0, D_0), such that the optimization process will make use of the "flow of information" generated by training an ensemble of N adversarial pairs $(G_1, D_1), \dots, (G_N, D_N)$, as sketched in Figure 1.

The rules of this game are as follows: G_0 and D_0 can solely be trained with $\{D_1, \dots D_N\}$ and $\{G_1, \dots G_N\}$, respectively, and local pairs do not have access to outputs or gradients from G_0 and D_0.

The most prominent advantages of such a training are:

1. if the training of a particular pair degrades or oscillates, the global networks continue to learn with higher probability;
2. it is much more likely that training one pair will fail than training all of them, hence the choice of not letting global models to affect the ensemble;
3. if the models' limited capacity is taken into account *i.e.* p_g can capture a limited number of modes of p_d (which increases with the number of training iterations), and under the assumption that each mode of p_d has a non-zero probability of being captured, then the modeled distribution by the ensemble is closer to p_d in some metric space due to the statistical averaging; and conveniently
4. large chunks of the computation can be carried out in parallel making the time overhead negligible.

In what follows, we first review in § 2 GAN variants we use in the experimental evaluation of our SGAN algorithm, which to the best of our knowledge are the current state-of-the-art methods. We then describe SGAN in detail in § 3, and present thorough experimental evaluation in § 4 as well as in the Appendix. We then present some methods that although unrelated to the SGAN approach, do propose a multi-agent structure in § 5.

2. Related work: Variants of the GAN algorithm

Optimizing Eq. 1 amounts to minimizing the Jensen-Shannon divergence between the data and the model distribution $JS(p_d, p_g)$ [6]. More generally, GANs learn p_d by minimizing a particular f-divergence between the real samples and the generated samples [23].

With a focus on generating images, [25] proposes specific architectures of the two models, named Deep Convolutional Generative Adversarial Networks–**DCGAN**. [25] also enumerates a series of practical guidelines, critical for the training to succeed. Up to this point, when the architecture and the hyper-parameters are empirically selected, DCGAN demonstrates outperforming results both in terms of quality of generated samples and convergence speed.

To ensure usable gradient for optimization, the mapping $\theta_d \mapsto p_d$ should be differentiable, and to have a non-zero gradient everywhere. As the JS divergence does not take into account the Euclidean structure of the space, it may fail to make the optimization move distributions closer to each other if they are "too far apart" [1]. Hence, [1] suggests the use of the Wasserstein distance, which precisely accounts for the Euclidean structure. Through the Kantorovich Rubinstein duality principle [30], this boils down to having a K-Lipschitz discriminator.

From a purely practical standpoint, this means that strongly regularizing the discriminator prevents the gradient from vanishing through it, and helps the optimization of the generator by providing it with a long-range influence that translates into a non-zero gradient.

In **WGAN** [1] the Lipschitz continuity is forced through weight clipping, which may make the optimization of D harder–as it makes the gradient with respect to D's parameters vanish–and often leads to degrading the overall convergence. It was later proposed to enforce the Lipschitz constraint smoothly by adding a term in the loss which penalizes gradients whose norm is higher than one–**WGAN-GP** [8].

Motivated by game theory principles, [13] derives combined solution of vanilla GAN and WGAN with gradient penalty. In particular, the authors aim at smoothing the value function via regularization by minimizing the regret over the training period, so as to mitigate the existence of the multiple saddle points. Finally, while building on vanilla GAN, the proposed algorithm named **DRAGAN**–Deep Regret Analytic GAN–forces the constraint on the gradients of $D(x)$ solely in local regions around real samples.

3. Method

Structure. We use a set $\mathcal{G} = \{G_1, \dots G_N\}$ of N generators, a set $\mathcal{D} = \{D_1 \dots D_N\}$ of N discriminators, and a global generator-discriminator pair (G_0, D_0), as sketched in Figure 1.

Summary of a simplified-SGAN implementation. The pairs (G_n, D_n), $n = 1, \dots N$ are trained individually in a standard approach. In parallel to their training, D_0 is optimized to detect samples generated by any of the local generators $G_1, \dots, G_N$, and similarly G_0 is optimized to fool all of the local discriminators $D_1, \dots, D_N$.

Note that, to satisfy the theoretical analyses of minimizing the Wasserstein distance and the Jensen-Shannon divergence for WGAN and GAN, respectively, the above procedure of training implies that each $\{D_1 \dots D_N\}$ *should be trained with* G_0 at each iteration of SGAN. Solely by following such a procedure G_0 follows the principles of the GAN framework [6], which trains it with gradients "meaningful" for it.

Algorithm 1: Pseudocode for SGAN.

```
Input : $\mathcal{X}_{inf}$, $N$, I, $I_D$
1  $\mathcal{G}, \mathcal{D}$ = init($N$);
2  $G_0, D_0$ = init(1)
3  for $i \in \{1 \dots I\}$ do
4      for $n \in \{1 \dots N\}$ do
5          for $j \in \{1 \dots I_D\}$ do
6              zeroGradients($\mathcal{D}[n]$);
7              backprop($\mathcal{G}[n], \mathcal{D}[n], \mathcal{X}_{inf}$);
8              updateParameters($\mathcal{D}[n]$);
9          end
10         zeroGradients($\mathcal{G}[n]$);
11         backprop($\mathcal{G}[n], \mathcal{D}[n], \mathcal{X}_{inf}$);
12         updateParameters($\mathcal{G}[n]$);
13     end
14     $\mathcal{D}^{msg}$ = copy($\mathcal{D}$);
15     for $n \in \{1 \dots N\}$ do
16         for $j \in \{1 \dots I_D\}$ do
17             zeroGradients($\mathcal{D}^{msg}[n]$);
18             backprop($G_0, \mathcal{D}^{msg}[n], \mathcal{X}_{inf}$);
19             updateParameters($\mathcal{D}^{msg}[n]$);
20         end
21     end
22     zeroGradients($G_0$);
23     for $n \in \{1 \dots N\}$ do
24         backprop($G_0, \mathcal{D}^{msg}[n], \mathcal{X}_{inf}$);
25     end
26     updateParameters($G_0$);
27     zeroGradients($D_0$);
28     for $n \in \{1 \dots N\}$ do
29         backprop($\mathcal{G}[n], D_0, \mathcal{X}_{inf}$);
30     end
31     updateParameters($D_0$);
32 end
Output: $G_0, D_0$
```

Introducing "messengers" discriminators for improved guarantees. To prevent that one of the network pairs "influences" the ensemble, and thus keep the guarantees of successful training, we propose to train G_0 against herein referred as "messengers" discriminators $D_1^{msg}, \dots, D_N^{msg}$, which at re-created at every iteration as clones of $D_1, \dots, D_N$, optimized against G_0.

We empirically observed that this strategy helps consecutive steps to be more coherent, and improves drastically the convergence. It is worth noting that, despite the increased complexity in terms of obtaining the theoretical analyses, such an approach is practically convenient since it allows for training G_0 in parallel to the local pairs.

3.1. Description of SGAN

More formally, let $\mathcal{X}_{inf}$ be a sampling operator over the dataset, which provides mini-batches of i.i.d. samples $x \sim p_d$.

Let *backprop* be a function that given a pair G and D, buffers the updates of the networks' parameters, *updateParameters* that actually updates the parameters using these buffers, and *zeroGradients* resets these buffers. Also, let *init* be a function that initializes a given number of pairs of G and D. Let N be the number of pairs to be used. The algorithm iterates for a given number of iterations I, and depending on the used GAN variant, each discriminator network is updated either once or several times, hence the I_D input parameter.

At each iteration, foremost the local models are being updated (lines 4 - 12).

Then, to obtain meaningful gradients for G_0, without affecting the local models, we first make a copy of the latter (line 14) into the "messenger discriminators" $\mathcal{D}^{msg}$, and update them against G_0 (lines 15 - 21). We then update G_0 jointly versus all of the discriminators (lines 22 - 26).

As D_0 does not affect generators it is trained with, it is directly updated jointly versus all of the local generators (lines 27 - 31).

Note that for clarity in Alg. 1 we present SGAN sequentially. However, each iteration of the training can be parallelized since G_0 is trained with a copy of $\mathcal{D}$, and the local pairs can be trained independently. In addition, Alg. 1 can be used with different GAN variants.

SGAN can also be implemented with weight-sharing (see § 4.1) since low-level features can be learned jointly across the networks. This motivates the training of D_0 in Alg. 1. In addition, whether the discriminator can be made use of is not a closed topic. In fact, a recent work answers in the affirmative [15].

4. Experiments

Datasets. As toy problems in $\mathbb{R}^2$ we used (i) mixtures of M Gaussians (M-GMM) whose means are regularly positioned either on a circle or a grid, with $M = 8, 10$, or 25, and (ii) the classical Swiss Roll toy dataset [20]. In the former case, we manually generate such datasets, by using a mixture of M Gaussians with modes that are uniformly distributed in a circle or in a grid. With such an evaluation, we follow related works–for *e.g* [8, 13, 29] since GANs in prior work often failed to converge even on such simplistic datasets [21].

To assess SGAN or real world applications, we used:

1. small scale datasets: CIFAR10 [14, chapter 3], STL-10 [2], MNIST [16], as well as the recent FASHION-MNIST [32];
2. large scale datasets: CelebA [18], LSUN [33] using its "bedroom" class, and ImageNet [26]; as well as
3. large language corpus of text in English, known as One Billion Word Benchmark [12].

Methods. As WGAN with gradient penalty [8] outperformed WGAN with weight clipping [1] in our experiments, herein as "WGAN" we refer to the former. Regarding vanilla GAN, instead of minimizing $\log(1 - D(G(z)))$, we train G to maximize $\log(D(G(z)))$, as it is recommended in [6], and done in practice [6, 25]. For conciseness, let us adopt the following notation regarding SGAN: we prefix the type of GAN with N-S, where N is the number of local pairs being used. For example, SGAN with 5 WGAN local pairs and one global WGAN pair would be denoted as 5-S-WGAN.

Implementation. For experiments on toy datasets, we used separate $2{\times}(N{+}1)$ neural networks. Regarding experiments on real-world datasets, we experimented with two implementations: using separate networks, as well as sharing parameters. In the latter case, we used approximately half of the parameters to be shared among the generators, and analogously same quantity among the discriminators. For further details on our implementation, see Appendix.

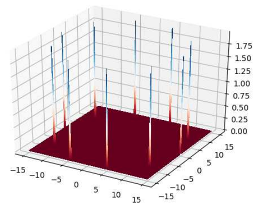
As a deep learning framework we used PyTorch [24].

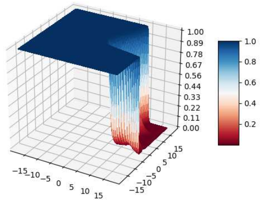
Metrics. A serious limitation to improve GANs is the lack of a proper means of evaluating them [19]. When dealing with images, most commonly used measure is the **Inception score** (IS) [27]. This metric feeds a pre-trained model named Inception [28] with generated images and measures the Kullback–Leibler 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.

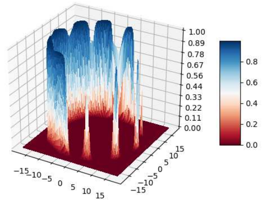
Using real data images of ImageNet, LSUN-bedroom, CIFAR10, and CelebA, we obtain the following Inception scores: 46.99 (3.547), 2.37 (0.082), 10.38 (0.502), 2.50 (0.082), respectively. The high variance across the datasets, suggests that training the model on the dataset at hand may improve the estimate. Hence, we adopt it as is for CIFAR10–as it turned into a standard GAN metric, whereas for MNIST we use a classifier specifically trained on this dataset. In the former case, we use the original implementation of it [27] and a sample of p_g of size $50{\cdot}10^3$, whereas for the latter we use our own implementation in PyTorch [24].

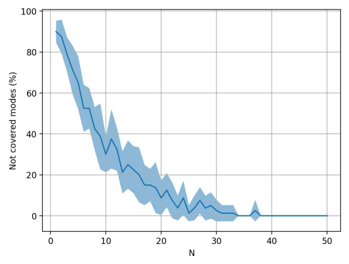
The **Fréchet Inception Distance** (FID) [9] also relies on the Inception model [28], but uses it to embed samples into a "good" feature space. It consists of first estimating the mean μ_g and μ_d, and covariances C_g and C_d, respectively for p_g and p_d in that feature space, and computing $D_{FID}(p_d, p_g) \approx d^2((\mu_d, C_d), (\mu_g, C_g)) = ||\mu_d - \mu_g||_2^2 + Tr(C_d + C_g - 2(C_d C_g)^{\frac{1}{2}})$, where d^2 denotes the Fréchet Distance.

For experiments on MNIST, using a separately trained classifier, we also plot the **entropy** of the generated sam-


(a) Real data (10-GMM)


(b) Discriminator output


(c) S-Discriminator output



(d) Not-covered modes (%)

Figure 2: Figures (a-c) depict a toy experiment with vanilla GAN. Figure (d) depicts the percentage of not covered modes (y-axis) by the generators, as more pairs are used (x-axis). See text for details, § 4.1.

ples' mode distribution, as well as the **total variation** between the class distribution of the generated samples and a uniform one. For toy experiments on mixtures of Gaussians, we also used the log-likelihood of the generated samples.

For more results and details on our implementation, see the Appendix.

4.1. Experimental results on toy datasets

Independently trained ensemble of GANs. To motivate the idea of favoring information from the independent ensemble to train a single pair, we conduct the following experiment. We train in parallel few pairs of networks, as well as *two* additional pairs: (i) **SGAN**–trained with the local independent pairs, as well as (ii) **GAN**–a regularly trained pair. In addition to training these two pairs with equal frequency, we used the *identical real-data and noise samples*.

Figure 2 depicts vanilla-GAN experiment on the 10-GMM dataset (Figure 2a). We recall that the only difference between the two discriminators is that the GAN discriminator is trained with fake samples from his tied single opponent (Figure 2b), whereas the one of SGAN is trained with fake samples from the ensemble (Figure 2c).

Figure 2d depicts that the probability that a mode will not be covered (y-axis) by the ensemble, at a random iteration, goes down exponentially with the number of pairs (x-axis). For this experiment, we used the 8-GMM toy dataset and the vanilla-GAN algorithm.

Performance of the global pair in SGAN. In Figure 3 we use WGANs, where each network is a multilayer perceptron (MLP) of 4 fully connected layers (see App. 1). The first column depicts the 10 local pairs: generators' samples and discriminators' contours (level sets) are displayed in varying and transparent colors, respectively. The rightmost column depicts the 10-S-WGAN pair (trained with the networks of the first column): samples from G_0 are drawn in green, whereas the illustrated contours are from D_0. We observe that S-WGAN exhibits higher stability and faster convergence speed. Figure 3 also depicts samples from a generator updated N-fold times more (middle column), what indicates that an SGAN generator is comparable with these.

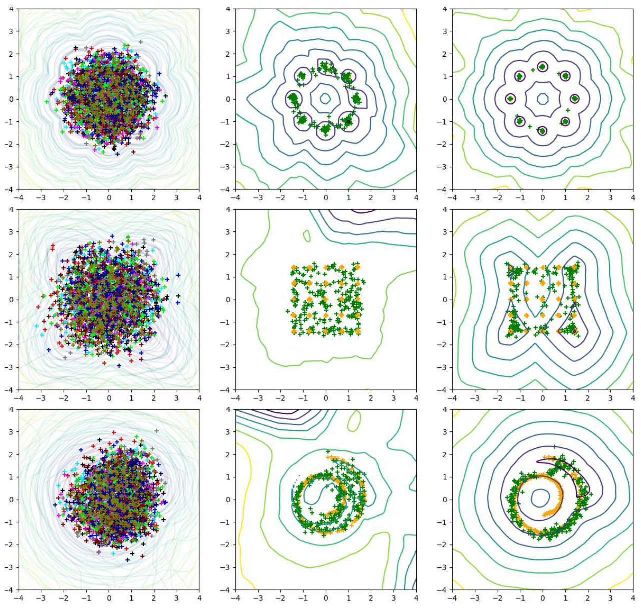


Figure 3: (10-S-)WGAN on (top to bottom row): circle 8-GMM, grid 25-GMM, Swiss Roll (best seen in color). Real data-points are shown in orange. The level sets of the output of the discriminator(s) are shown with yellow to purple contours which denote low and high, respectively. See § 4.1.

Figure 4 depicts 10-S-WGAN experiment on the 8-GMM toy dataset. At each iteration, after training the local pairs, the global generator is trained with the local discriminators, samples of which are displayed on the left and right columns, respectively. We observe that in the earlier iterations samples from the global generator may lie in $\mathbb{R}^2$ regions distinct from the real data samples. Nonetheless, it converges notably quicker, and through the early iterations, its generated samples lie in regions which often do not intersect with those of the local generators (jointly) at the same iteration.

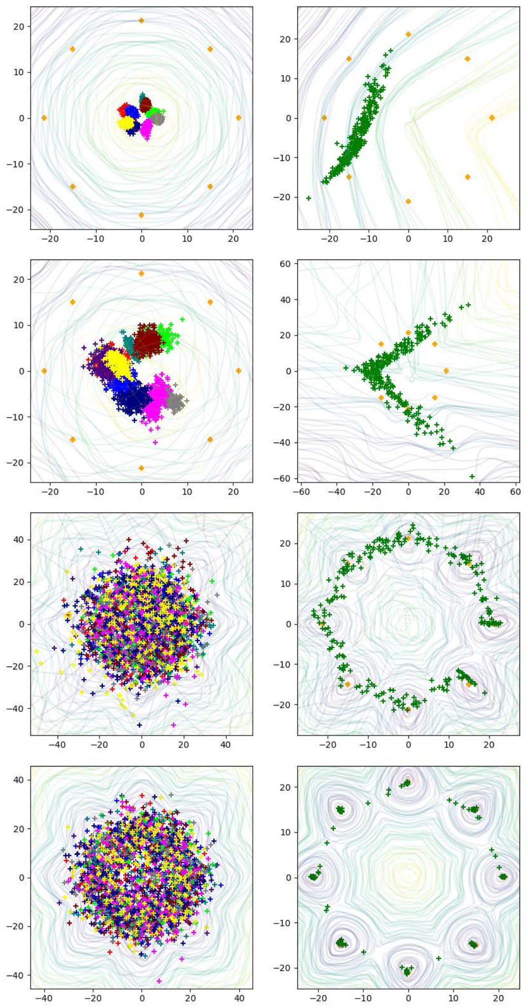


Figure 4: 10-S-WGAN on the 8-GMM toy dataset (best seen in color). Samples from p_d are displayed in orange. Each row is a particular iteration (top to bottom): 5th, 10th, 100th, and 400th iteration. Samples from the local generators and the global one are illustrated on the left (in separate colors) and right (in green), respectively. The displayed contours represent the level sets of $\mathcal{D}$ and $\mathcal{D}^{msg}$–illustrated on the left and right, respectively, where yellow is low and purple is high.

Value of N. Figure 5 depicts the log-likelihood on 8-GMM for different values of N, where "simplified-SGAN" denotes the SGAN variant without the messengers discriminators (see § 3). We observe that increasing N helps, but that the trade-off performance gain versus computation resources starts to saturate compared to the gain obtained of SGAN relative to regular training.

4.2. Experimental results on real-world datasets

In Figures 6 and 8 we show experimental results on image datasets. In the latter, samples are taken at a random

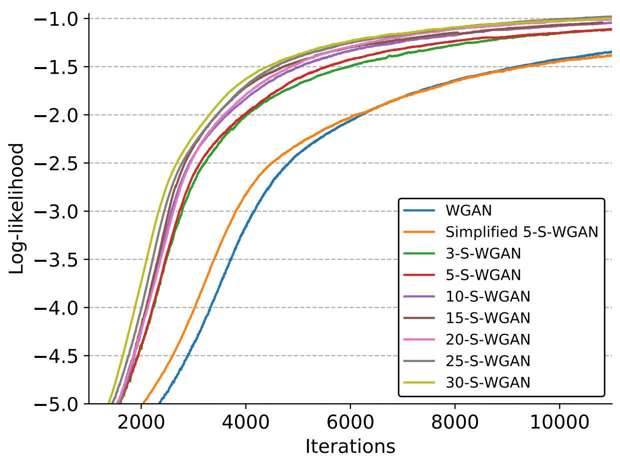


Figure 5: Log-likelihood on 8-GMM toy dataset (see § 4.1).

iteration, *prior to final convergence*, so as the difference in the quality of the samples is clearer. Figure 7 depicts quantitative results on CIFAR10 using the Inception Score (Figure 7a) and the Fréchet Inception Distance (Figure 7b), under identical hyperparameter setup and architecture choice (see App. 2 for details on our implementation).

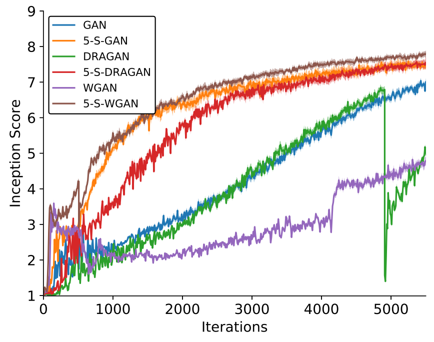
In Table 1 we show fake samples of the global generators of S-WGAN on the One Billion Word Benchmark dataset. The output of a standard WGAN training is *a single character (white space) for all of the first 660 iterations*, what indicates slower than N-fold convergence speed compared to an N-S-Generator. In addition, it is interesting to observe similar behavior as on toy datasets: at the first iteration (first row in Table 1) the SGAN generators are pushed far from the modes of the real data samples, as they generate non-commonly used letters.

Figure 9 depicts samples taken from each generator in 5-S-DCGAN at the $100{\cdot}10^3$-*th* iteration, as well as samples from the generator of a separately trained $DCGAN$ pair at $100{\cdot}10^3$-*th* and $500{\cdot}10^3$-*th* iterations. Besides that the global generator of SGAN shows no visible mode collapse for the human eye (compared to samples taken from the rest of the generators), we also observed that its performance did not oscillate through the iterations.

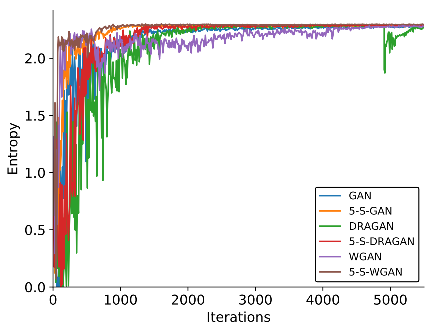
5. Related work: Multi-network GAN methods

Independently, Boosted Generative Models [7] and AdaGAN [29] propose the iterative boosting algorithm to solve the mode collapse problem. At each step, a new component is added into a mixture of models, by updating the samples' weights, while using the vanilla GAN algorithm.

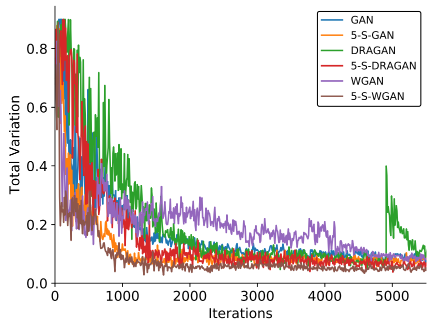
With the similar motivation of increasing the mode coverage [5] and [10] propose to instead train multiple generators versus a single discriminator.



(a) Inception score [27] (higher is better)

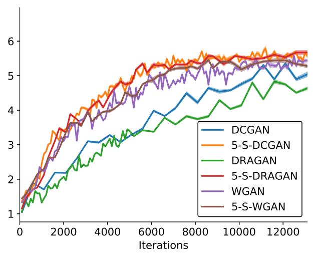


(b) Entropy (higher is better)

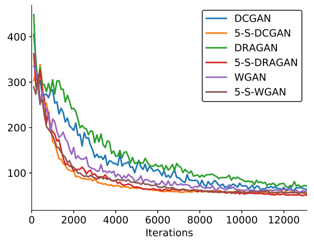


(c) Total variation (lower is better)

Figure 6: Results on MNIST using (5-S-) GAN/WGAN/DRAGAN (best seen in color).



(a) IS [27] (higher is better)



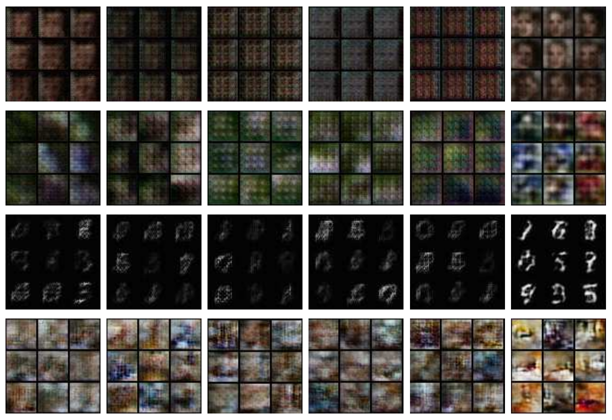
(b) FID [9] (lower is better)

Figure 7: Results on CIFAR10 (best seen in color).

Table 1: Output snippets of the global generators trained on the One Billion Word Benchmark. The top to bottom rows depict the 1-*st*, 100-*th* and the 200-*th* iteration. See § 4.2.

5-S-WGAN	10-S-WGAN
łqłł\{m&łłł&u 8&uqA{?Mł88łq łqłqłM{ł&8łłYł ’A&H{m7(HqłH^Hł{ qłłł&u&łłM8mł?ł mł8ł&łtHqq9{łmvł8 łłłqłłq!m{ł	A8qqqH& H&włw Ass8qŁmsqv mAsmm` Y1&s&1u8 młW6]9m&? AuWH1&m&
aaa aaaaaaaaaaaaaa aaaa aaaaaaaa a aaaaaa aaaaaaaa aaaaa aaaaaa aaaaaa a aaaaaaaaa aaaaa aaaaaa aaa aaaaaaaa aaaaaaaaaaaaaaaa a a aaaaaaaaaaa aaaaaaaaa aaaaaaaa aaaaaa aaaa a aaaaaaaaaa aaaaaa a aaa aa aaaa aaaaaaaaaaaaaaa	ii iii iiiii iiiii i iiiii ii ii iiiii ii iiii i iiiii ii iiiii ii ii iii ii iiii i iiii iiii iii i i iiii iiiiii i iii iiiii i iii i i ii i iii ii ii iii i iii ii ii iii ia iiiiii i i iiiii i i i ii
hieq as ieq aa dhhie as ie t eq shiq as heq aaa hheq asheq t ieqq aasheq dsheq aa dd dhhie t iq ddq as ie sie ashieq as e t eq as hheq aas ie s heq as e t q hheq aas ieq dshieq as hie at eq asid ddd as hieq as heq diq t	SAeS Aer areS SnSSSharSonS Soe AS SSSSer oeS SarSonSSS Ss ShS i S tes SrSsne SoerhaS SsnS Soar o MSSSS S Sha tos ShS aoS as Sha i s She tAeS SsS SsnSSSoes ShS Son eS es S SoS Ss SoSSS Ss SSS Son ASSSS tSa SoarSonS Ssne Soar osn

In [5] the discriminator is trained against N generators which share parameters in all layers except the last one, and it outputs probability estimate for $N+1$ classes representing whether the input is a real sample, or by whom of the generators it originates. To enforce diversity between the generated samples, a penalty term is added with a user-defined similarity based function.



(a) G_1 (b) G_2 (c) G_3 (d) G_4 (e) G_5 (f) G_0

Figure 8: **5-SGAN**. In the top to bottom rows we use DCGAN on CelebA, DRAGAN on ImageNet, WGAN on MNIST and DCGAN on LSUN, respectively. Each of the above samples are taken at the earlier iterations, in particular at the 100-*th*, 500-*th*, 500-*th* and the 1000-*th* iteration, respectively for each row. In columns (a-e) we show samples from the local generators, whereas in (f) from the global generator. We used separate networks and real data space of 32×32.

Similarly, [10] proposes multiple generators that share parameters versus single discriminator whose output is fake versus real, as well as training an additional model that classifies by whom of the generators a given fake input was generated. The output of the classifier is used in an additional penalty term that forces diversity between the generators. [3] proposes utilizing multiple discriminators versus one generator, in aim to stabilize the training.

[4] proposes multiple generators versus single discriminator, where the generators communicate through two types of messages. Namely, there are both co-operation and competing objectives. The former ensures the other generator to

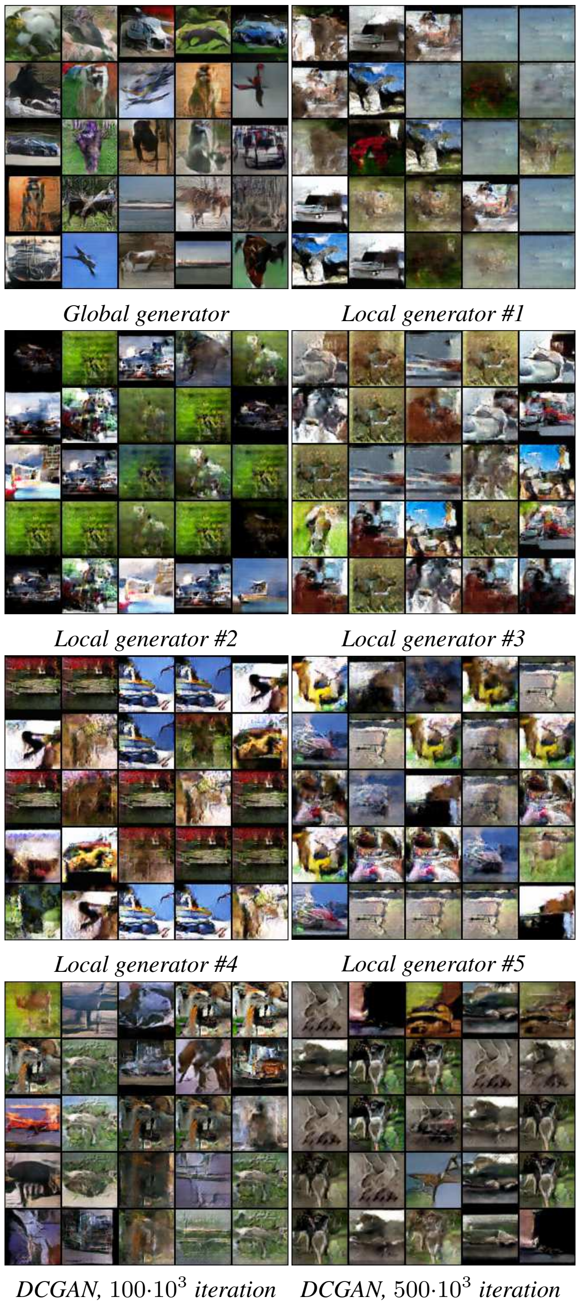


Figure 9: Samples of the generators of 5-S-DCGAN on the STL-10 dataset at the $100{\cdot}10^3$-*th* iteration (rows $1-3$), as well as separately trained DCGAN at the $100{\cdot}10^3$ and $100{\cdot}10^3$-*th* iteration (bottom row).

generate images better than itself, and the latter encourages each generator to generate better samples than its counterpart. Motivated by the observed oscillations, in [31] a so-called "self-ensembles" is proposed. Non-traditionally, this self-ensemble is built out of copies of the generator taken from a different iteration while training a single pair.

Hence, SGAN depicts different structures and solutions to the problem of training GANs. Regarding the former, none of the above methods utilizes explicitly multiple pairs trained independently. Instead, most commonly a structure of one-to-many is used, either for the generator or for the discriminator. Compared to AdaGAN, SGAN is applicable to any GAN variant, runs in parallel, and produces a single generator. Concerning the latter, SGAN uses "supervising" models and prevents an influence of one pair towards all.

6. Discussion

We proposed a general framework dubbed SGAN for training GANs, applicable to any variant of this algorithm. It consists of training several adversarial pairs of networks independently and uses them to train a global pair that combines the multiple learned representations.

A key idea in our approach is maintaining the statistical independence between the individual pairs, by preventing any flow of information between them, in particular through the global pairs it aims at training eventually. Maintaining this makes the probability of a failure to go down exponentially with the number of pairs involved in the process.

The motivations of such a training methodology originate from the discrepancy between the theoretical justifications being derived in functional space, and the fact that we optimize the parameters of the deep neural networks [6]. More precisely, training the generator finally produced by SGAN in such a way aims at addressing if the limited representational capacity (more prominent at the early iterations) affects the trajectories taken during the optimization procedure, which could itself be an important factor causing the training difficulties. The presented empirical evaluation indicates that it is indeed the case, as it was shown that such a training follows different trajectories and that for realistic datasets, it eliminates oscillations observed with a standard training under an identical set-up.

Experimental results on diverse datasets demonstrate systematic improvements upon classical algorithms as well as increased stability of the framework regarding real-world applications. Furthermore, SGAN is convenient for many applications, as it produces a single generator.

Some future extensions of SGAN include improving the covering behavior by forcing diversity between the local pairs and re-casting the analysis in the context of multi-player game theory.

Acknowledgment

This work was supported by the Swiss National Science Foundation, under grant CRSII2-147693 "WILDTRACK" and the Hasler Foundation through the "MEMUDE" project. We also gratefully acknowledge NVIDIA's support through their academic GPU grant program.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 214–223, 2017. 2, 3, 4

[2] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In G. Gordon, D. Dunson, and M. Dudk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. 4

[3] I. P. Durugkar, I. Gemp, and S. Mahadevan. Generative multi-adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2017. 7

[4] A. Ghosh, V. Kulharia, and V. P. Namboodiri. Message passing multi-agent gans. *CoRR*, abs/1612.01294, 2016. 7

[5] A. Ghosh, V. Kulharia, V. P. Namboodiri, P. H. S. Torr, and P. K. Dokania. Multi-agent diverse generative adversarial networks. *CoRR*, abs/1704.02906, 2017. 6, 7

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014. 1, 2, 3, 4, 8

[7] A. Grover and S. Ermon. Boosted generative models. *CoRR*, abs/1702.08484, 2017. 6

[8] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777. 2017. 2, 3, 4

[9] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30*, pages 6626–6637. 2017. 4, 7

[10] Q. Hoang, T. Dinh Nguyen, T. Le, and D. Phung. Multi-Generator Generative Adversarial Nets. *ArXiv e-prints*, Aug. 2017. 6, 7

[11] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[12] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *ArXiv e-prints*, 2016. 4

[13] N. Kodali, J. D. Abernethy, J. Hays, and Z. Kira. How to train your DRAGAN. *CoRR*, abs/1705.07215, 2017. 3, 4

[14] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009. 4

[15] W.-S. Lai, J.-B. Huang, and M.-H. Yang. Semi-supervised learning for optical flow with generative adversarial networks. In *Advances in Neural Information Processing Systems 30*, pages 353–363. 2017. 4

[16] Y. Lecun and C. Cortes. The MNIST database of handwritten digits. 4

[17] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016. 1

[18] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015. 4

[19] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are GANs Created Equal? A Large-Scale Study. *ArXiv e-prints*, Nov. 2017. 2, 4

[20] S. Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009. 4

[21] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016. 4

[22] A. Nguyen, J. Yosinski, Y. Bengio, A. Dosovitskiy, and J. Clune. Plug & play generative networks: Conditional iterative generation of images in latent space. *CoRR*, abs/1612.00005, 2016. 1

[23] S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. *ArXiv e-prints*, June 2016. 2

[24] A. Paszke, S. Gross, S. Chintala, and G. Chanan. PyTorch. `https://github.com/pytorch/pytorch`, 2017. 4

[25] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 2, 4

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 4

[27] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29*, pages 2234–2242, 2016. 4, 7

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. 4

[29] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf. AdaGAN: Boosting generative models. In *Advances in Neural Information Processing Systems 30*. 2017. 4, 6

[30] C. Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer, 2009 edition, Sept. 2008. 3

[31] Y. Wang, L. Zhang, and J. van de Weijer. Ensembles of generative adversarial networks. *CoRR*, abs/1612.00991, 2016. 8

[32] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017. 4

[33] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *ArXiv e-prints*, 2015. 4

[34] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017. 1

SGAN: An Alternative Training of Generative Adversarial Networks – Supplementary Material –

Tatjana Chavdarova and François Fleuret
Machine Learning group, Idiap Research Institute & École Polytechnique Fédérale de Lausanne
{firstname.lastname}@idiap.ch

The implementation details, as well as additional results of experiments on toy and realistic datasets, are given in Sections 1 and 2. In Section 3 we discuss two viewpoints of SGAN. We use notations as in the paper (see § 4).

1. Experiments on toy datasets

1.1. Details on the implementation

For experiments conducted on toy datasets, we used separate $2{\cdot}(N{+}1)$ networks. The architecture and the hyperparameters are as follows. Each network is a multilayer perceptron (MLP) of 4 fully connected layers and LeakyReLU non-linearity [6] with the PyTorch's default value for the negative slope of 0.01 [7]. The number of hidden units for each of the layers is 512, whereas the dimension of the input noise vector for the generator network is 100.

We use learning rate of $1{\cdot}10^{-5}$, as well as the *Adam* optimization method [5]. Using *RMSProp* [12] as optimization method did not give obvious improvements in our conducted experiments.

1.2. Experiments

Figure 1 depicts several image pairs, of: (i) samples generated by the local generators (left); and (ii) samples from the global one (right). The illustrated contours are obtained with GMM Kernel Density Estimation (KDE) [10], whose bandwidth is cross-validated. We used sample of p_g of size 500 (in Figure 1, N denotes the sample size). C in Figure 1 denotes the *Coverage* metric [13].

Figure 4 depicts experiment in which the parameters of the global pair are updated after each update of a local pair.

2. Experiments on real-world datasets

2.1. Details on the implementation

Regarding experiments on real-world applications, we considered: (i) using separate $2{\cdot}(N{+}1)$ networks; as well as (ii) using parameter sharing of the networks. In the latter, approximately half of the parameters of each network are shared among the corresponding other N networks (discriminators or generators). To distinguish the two, in the sequel we denote the former and the latter case as **N-S-** and **N-SW-**, respectively. For *DCGAN, we recommend using separate networks*, as sharing parameters makes the generators to produce similar samples, thus the performance gain of SGAN can be marginal.

We used learning rate of $1{\cdot}10^{-5}$, and a batch size of 50 and 64 for (FASHION)MNIST and the rest of the datasets, respectively. Unless otherwise stated, we used the *Adam* optimizer [5] whose hyperparameters (one parameter used for computing running averages of gradient and another for its square) we fixed to 0.5 and 0.999, as in [9].

Implementation of the experiments on image datasets. For **MNIST** we did experiments using both MLPs and CNNs for the generators and the discriminators. In the former case, the architectures were almost identical to those used for the toy experiments, except that the first layer was adjusted for input space of $28{\times}28$. In the latter case, we used input space of $28{\times}28$ and we started with the DCGAN implementation [9] and changed it accordingly to the input space. In particular, we reduced the number of 2D transposed convolution layers from 5 to 4 and adjusted the hidden layers' sizes accordingly to the dimensions used for the real data space.

For **CIFAR10** unless otherwise emphasized, we used $32{\times}32$ image space. For the rest of the image datasets– unless otherwise stated, we used $64{\times}64$ input space and the original DCGAN [9] architecture, as provided by the authors. The implementation of DCGAN [9] uses Batch Normalization layers [4].

Implementation of the experiments on one Billion Word Benchmark. We started from the provided implementation of [2] and implemented our method. In particular, the character-level generative language model is implemented as a $1D$ CNN using 4 ResNet blocks [3], which network maps a latent vector into a sequence of one-hot character

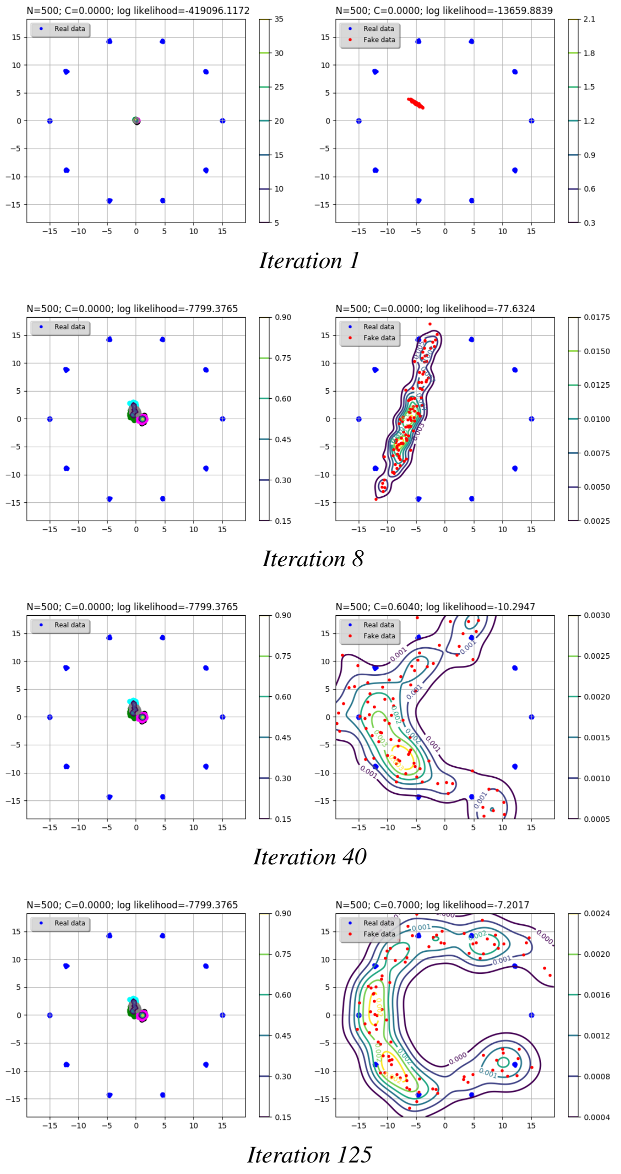


Figure 1: **5-S-WGAN** on the **10-GMM** dataset. Samples from the five local generators and from the global generator, are shown on the left (in separate color) and on the right (in red color), respectively. See § 1.2.

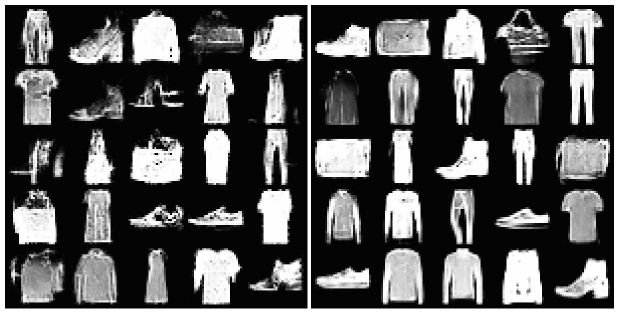


Figure 2: Samples of **DCGAN** and **5-S-DCGAN** on **FASHION-MNIST** taken at the 6000-*th* iteration, on the left and right, respectively. The input dimension is 28×28.

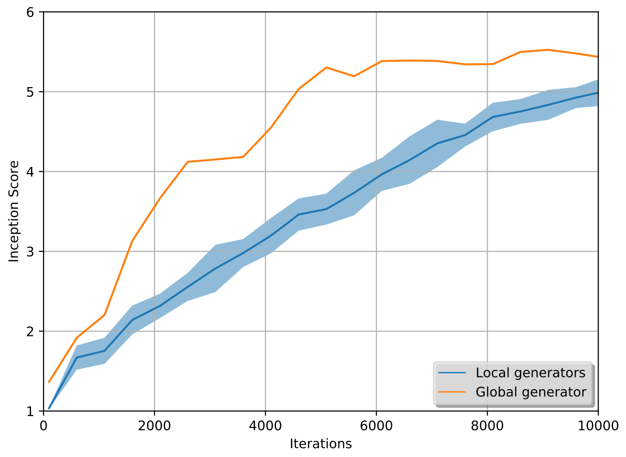


Figure 3: **10-S-DCGAN**, on **CIFAR10** (best seen in color). We plot the Inception Score [11] of the global generator (orange) as well as the scores of the local generators (blue). The input dimension is 32×32.

vectors of dimension 32. The discriminator is also a $1D$ CNN, that takes as input sequences of such character embeddings of size 32.

As optimization method we used *RMSProp* [12].

Separate networks. In Figure 3 we show the Inception score [11] (using its original implementation in TensorFlow [1]), of the global generator and the local generators.

In Figure 2 we show samples of **5-S-DCGAN** on **FASHION-MNIST** (on the right), as well as of **DCGAN** (on the left). Figures 5 & 6 depict samples using **DRAGAN** and **DCGAN**, respectively. We see that the global generator converges much earlier then the local ones.

Shared parameters. In Figure 9 we show samples when training **DRAGAN** and **5-SW-DRAGAN** on **LSUN-bedroom** with input dimension of 64×64. Finally, in Figure 10 we show samples when training on the **Billion Word** dataset.

3. Different viewpoints of SGAN

Connecting SGAN to Actor-critic methods. In [8] the authors argue that at an abstract level GANs find similarities with actor-critic (AC) methods, which are widely used in reinforcement learning. Namely, the two have a feed-forward

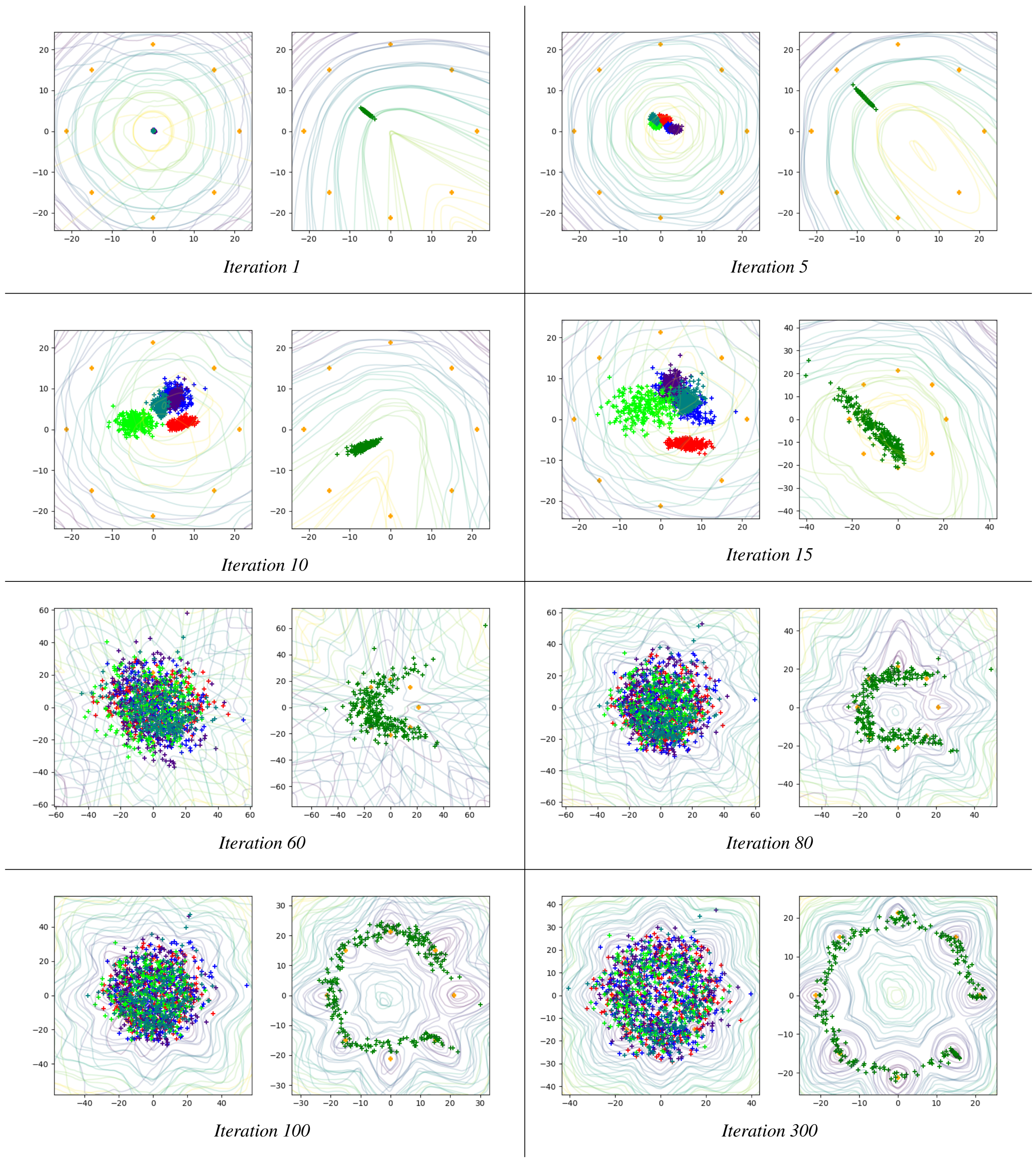
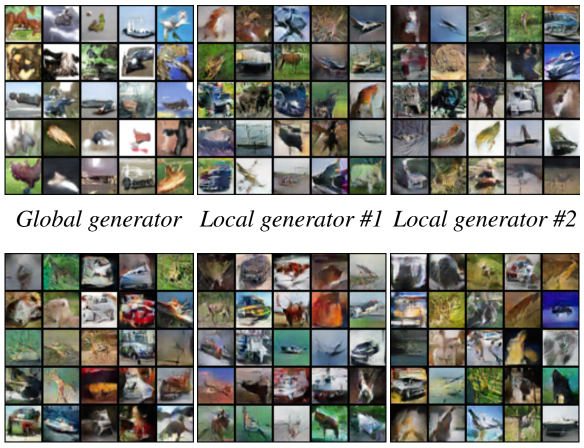


Figure 4: **5-S-WGAN** experiment on the **8-GMM** toy dataset (best seen in color). Real data samples are illustrated in orange. In each image pair, we illustrate samples from the five local generators and from the global generator, on the left (in separate color) and on the right (in green), respectively. The displayed contours represent the level sets of the discriminators $\mathcal{D}$ and $\mathcal{D}^{msg}$–illustrated on the left and right of each image pair, respectively, where yellow is low and purple is high.



Local generator #3 *Local generator #4* *Local generator #5*

Figure 5: **5-S-DRAGAN** on **CIFAR10** at $40{\cdot}10^3$-*th* iteration, and $32{\times}32$ real data space.

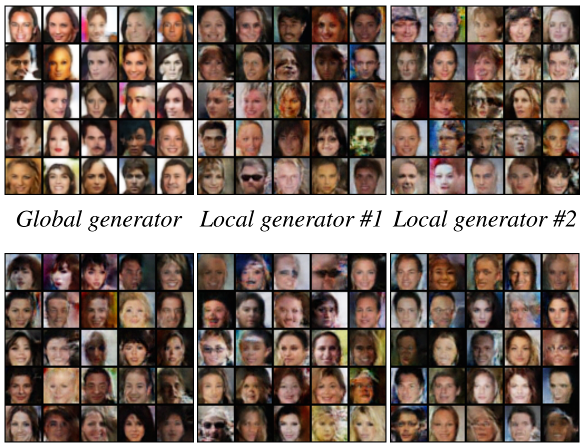


Figure 6: **5-S-DCGAN** on **CelebA** at $1{\cdot}10^3$-*th* iteration, and $32{\times}32$ real data space.

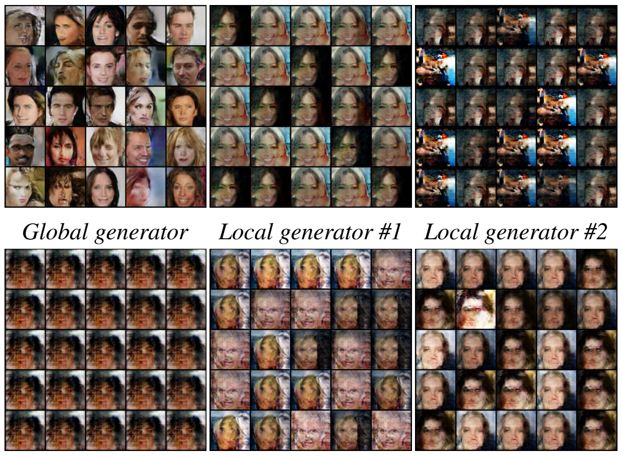


Figure 7: Samples of the generators of **5-S-DRAGAN** on the **CelebA** dataset at the $50{\cdot}10^3$-*th* iteration. The input dimension is $64{\times}64$.

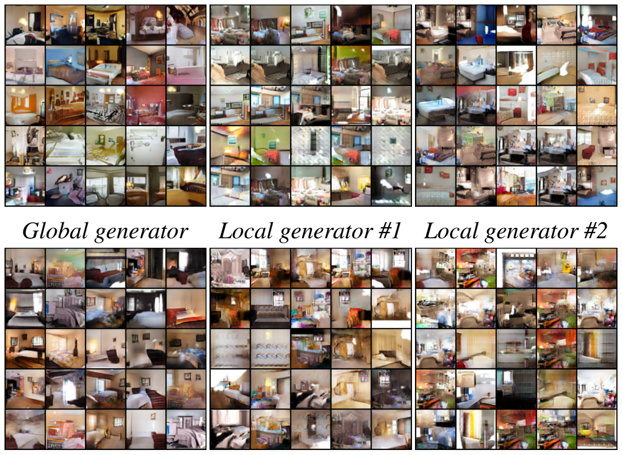


Figure 8: Samples of the generators of **5-S-DCGAN** on the **LSUN-bedroom** dataset at the $100{\cdot}10^3$-*th* iteration. The input dimension is $64{\times}64$.

model which either takes an action (AC) or generates a sample (GAN). This acting/generating model is trained using a second one. The latter model is the only one that has direct access to information from the environment (AC) or the real data (GAN), whereas the former has to learn based on the signals from the latter. We refer the interested reader to [8] which further elaborates the differences and finds connections that both the methods encounter difficulties in training.

We make use of the graphical illustration proposed in [8] of the structre of the GAN algorithm illustrated in Figure 11a, and we extend it to illustrate how SGAN works,

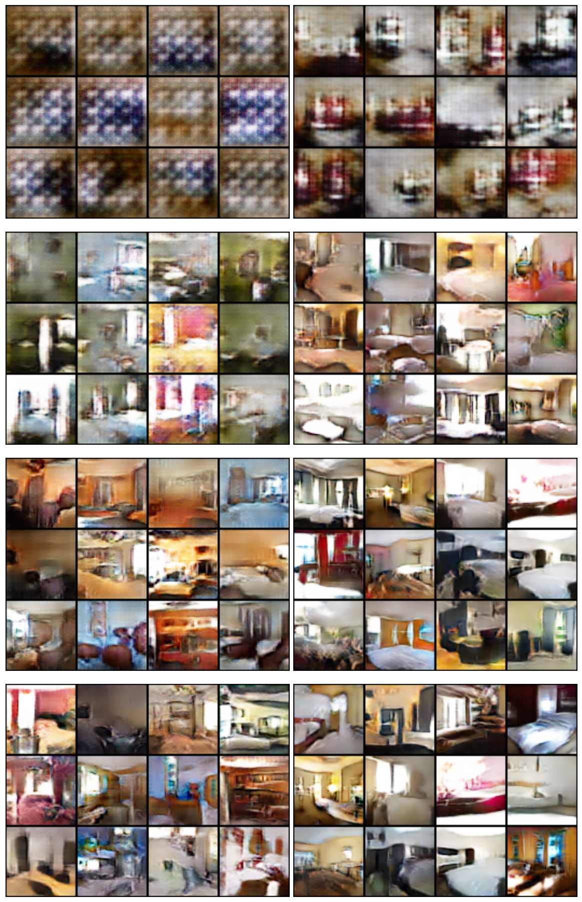
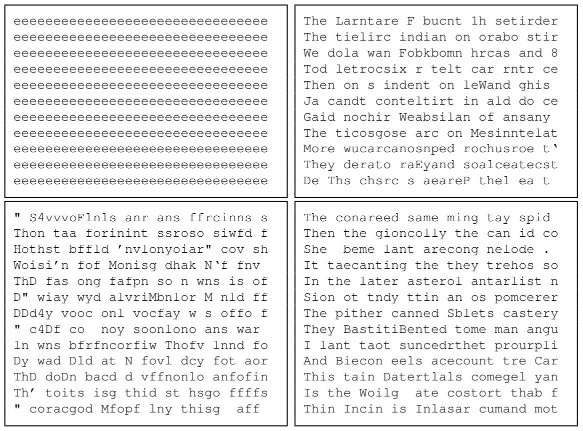


Figure 9: **DRAGAN** and **5-SW-DRAGAN** on **LSUN-bedroom** at the 1000-*th*, $5{\cdot}10^3$-*th*, $10{\cdot}10^3$-*th* and $14{\cdot}10^3$-*th* iteration, from top to bottom row, respectively. Using $64{\times}64$ real data space.

Figure 11b, where nodes with index i can be multiple. Empty circles represent models with a distinct loss function. Filled circles represent information from the environment. Diamonds represent fixed functions, both deterministic and stochastic. Solid lines represent the flow of information, while dotted lines represent the flow of gradients used by another model. In SGAN, D_0 is being trained with samples from the multiple generators whose input is in the real-data space. For clarity, we omited D^{msg} in the illustration–used to train G_0, as the arrows already indicate that these two "global" models do not affect the ensemble.

Game theoretic interpretation. We can define a game that describes the training of G_0 and D_0 in the SGAN framework as follows. Let us consider a tuple $(\mathcal{P}, \mathcal{A}, u)$, where $\mathcal{P} = \{G, D\}$ is the set of new players that we introduce. Let us assume that G and D, at each iteration can



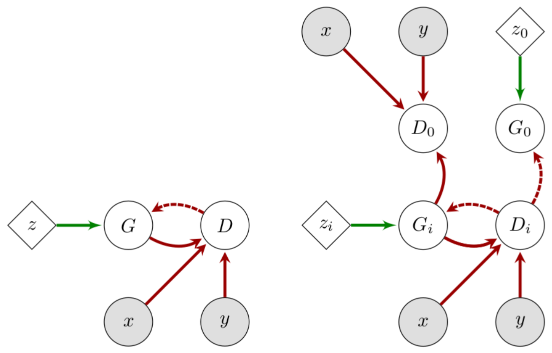
```
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
```

```
The Larntare F bucnt 1h setirder
The tielirc indian on orabo stir
We dola wan Fobkbomn hrcas and 8
Tod letrocsix r telt car rntr ce
Then on s indent on leWand ghis
Ja candt conteltirt in ald do ce
Gaid nochir Weabsilan of ansany
The ticosgose arc on Mesinntelat
More wucarcanosnped rochusroe t`
They derato raEyand soalceatecst
De Ths chsrc s aeareP thel ea t
```

```
" S4vvvoFlnls anr ans ffrcinns s
Thon taa forinint ssroso siwfd f
Hothst bffld ’nvlonyoiar" cov sh
Woisi’n fof Monisg dhak N`f fnv
ThD fas ong fafpn so n wns is of
D" wiay wyd alvriMbnlor M nld ff
DDd4y vooc onl vocfay w s offo f
" c4Df co  noy soonlono ans war
ln wns bfrfncorfiw Thofv lnnd fo
Dy wad Dld at N fovl dcy fot aor
ThD doDn bacd d vffnonlo anfofin
Th’ toits isg thid st hsgo ffffs
" coracgod Mfopf lny thisg  aff
```

```
The conareed same ming tay spid
Then the gioncolly the can id co
She  beme lant arecong nelode .
It taecanting the they trehos so
In the later asterol antarlist n
Sion ot tndy ttin an os pomcerer
The pither canned Sblets castery
They BastitiBented tome man angu
I lant taot suncedrthet prourpli
And Biecon eels acecount tre Car
This tain Datertlals comegel yan
Is the Woilg  ate costort thab f
Thin Incin is Inlasar cumand mot
```

Figure 10: Snippets from **WGAN** (left) and **5-SW-WGAN** (right) on the **One Billion Word Benchmark**, taken at the 700-*th* and 2500-*th* iteration (top and bottom row, respectively).



(a) GAN training [8] (b) SGAN training

Figure 11: Graphical representations [8] of the information flow structures of GAN and SGAN training. See § 3.

select among the elements of $\mathcal{D}$ and $\mathcal{G}$, respectively. Hence, $\mathcal{A} = (A_g, A_d)$ have a finite set of N actions.

Such "top level players" in SGAN assign uniform distribution over their actions, more precisely both G and D sample from the elements of $\mathcal{D}$ and $\mathcal{G}$ respectively, with uniform probability. To connect to classical training, let us assume that G and D fix their choice to one element of $\mathcal{D}$ and $\mathcal{G}$ respectively, *i.e.* with probability one they sample from a single generator/discriminator. The trained networks G_0 and G_i, as well as D_0 and D_j, with i and j being the selected choice of G and D respectively, are identical in expectation. Finally, rather than predefining the uniform sampling in SGAN, incorporating estimations of the actions' pay-off $u = (u_g, u_d)$ could prove useful for training (G_0, D_0).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2

[2] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777. 2017. 1

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1

[4] S. Ioffe and C. Szegedy. 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, pages 448–456. JMLR.org, 2015. 1

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 1

[6] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013. 1

[7] A. Paszke, S. Gross, S. Chintala, and G. Chanan. PyTorch. `https://github.com/pytorch/pytorch`, 2017. 1

[8] D. Pfau and O. Vinyals. Connecting generative adversarial networks and actor-critic methods. *CoRR*, abs/1610.01945, 2016. 2, 4, 5

[9] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 1

[10] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27(3):832–837, 09 1956. 1

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29*, pages 2234–2242, 2016. 2

[12] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 1, 2

[13] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf. AdaGAN: Boosting generative models. In *Advances in Neural Information Processing Systems 30*. 2017. 1