A deep introspection on Generative Adversarial Networks

Riccardo Lincetto, Guglielmo Camporese
Department of Information Engineering, University of Padova – Via Gradenigo, 6/b, 35131 Padova, Italy
emails: riccardo.lincetto, guglielmo.camporese @studenti.unipd.it

***Abstract*—GANs, namely Generative Adversarial Networks, are a hot topic nowadays. These models have the ability of generating good quality data, learning their distributions on a training set. Image generation in particular benefited from this framework, thanks also to the ease of assessment of the results obtained. Here we review the analysis of GAN models, recalling the necessary results from game theory, and propose a parametrization of the problem, which is then assessed analytically and by means of simulation, on image generation with MNIST dataset.**

I. INTRODUCTION

Since its rise, deep learning had a great impact on discriminative models. Generative models instead were not affected by this innovation at first, but this trend changed with the introduction of Generative Adversarial Networks (GAN), a powerful framework first introduced in [1]. Since then, GAN gained more and more momentum because of the ability of training *deep generative models*, avoiding some of the difficulties encountered in other frameworks [2].

GAN is a sub-class of generative models where a probability density function (pdf) is implicitly defined; since GAN makes use of, generally two, *neural networks*, the pdf is induced by their architecture and parameters design: given a training set of sample data, distributed according to an unknown pdf p_{data}, the purpose of GAN is indeed to generate samples according to a distribution p_g, that mimics p_{data}, without explicitly defining it. As suggested by the name, this is achieved by putting in competition two entities: a generator and a discriminator. The task of the generator (G) is to generate data that can be regarded as true by the discriminator, while the discriminator (D) has the purpose of correctly distinguishing real from fake data. The classical real-life analogy with this process involves counterfeiters trying to produce fake currency and the police trying to detect it. A graphical representation of the process is depicted in fig.1. This kind of interaction between the two entities can naturally be modeled with a game theoretical approach, where each player has its own strategies and payoffs, but in this paper we will rather talk about costs, as will be discussed in II. The major drawback of this framework anyway, is that the training of the model requires to compute the Nash Equilibrium (NE) of the game involving the two entities, which is not as simple as optimizing an objective function: without the guarantee of a NE, results obtained by the model could be different fom the ones desired.

In this paper we review some of the literature and explain our need to go back to the origins of GAN, implementing our own version of the code and simulating different scenarios, where discriminator is passed different fake-to-true ratios of images.

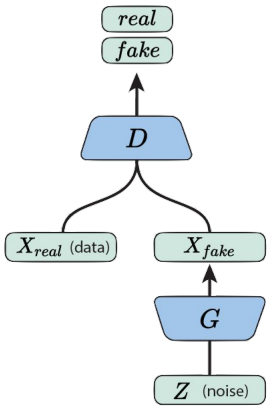

Figure 1: a block-diagram representing generative adversarial networks. A generator network (G) creates samples in data space from noise. A discriminator network then compares data, trying to distinguish true from fake samples.

The remainder of the paper is organized as follows: a brief overview of the literature is presented in II; a description of our work is then presented in III; the obtained results are presented in IV; finally we discuss our conlcusions in V.

II. RELATED WORK

Since their appearance in literature, GANs have been successfully applied to problems of image generation, editing and semi-supervised learning [5] [6]. Results obtained were so promising that the new framework captured the interest of many researchers, leading to a proliferation of various flavors of GAN, each claiming to have better performances on a specific domain. It's difficult anyway to understand how to compare different GAN models, because of the lack of a consistent metric and the different architectures networks can be designed with, which for each project are related to the corresponding computational budget. A tentative to define some guidelines to avoid these problems, together with a fair and comprehensive comparison of state-of-the-art GANs, is discussed in [3]: what emerges here is that the computational budget plays a major role, allowing bad algorithms to outperform good ones if given enough time; plus, despite the many claims of superiority, there's no empirical evidence that those algorithms are better

across all datasets, in fact in most of the cases the original model outperforms the others. We thus report here a formalization of the original problem [1], modeling it with a game theory approach in a more precise way, as done in [4].

Deep Convolutional GANs exploit as players (G and D) the learning capability of neural networks. The two anyway have different purposes, so in general they are designed with different architectures (e.g. as in fig.**??** and). As suggested in [1], the generator's distribution p_g over data x is learnt defining a prior on input noise variables $p_z(z)$ and representing the mapping to the data space as $G(z;\theta_g)$, where θ_g stands for the weights of the network. Similarly for the discriminator, a mapping $D(x,\theta_d)$ can be defined from the data space to a scalar, also referred to as $D(x)$, representing the probability that the input belongs to the true distribution p_{data}, i.e. to the training set. In the formalization of the game then, pure strategies are defined by the sets of possible θ_g and θ_d, while the utilities functions are the opposite of the loss functions that the networks have to minimize.

For classification tasks, cross-entropy loss function is universally accepted as the best choice, giving good results in terms of learning speed: this is what is usually selected for D in GAN. The simplest design of GAN uses as loss function for G the opposite of D's, defining thus a zero-sum game (MM-GAN). Ideally, the number of strategies for each player would be infinite but, as pointed out in [4], when using floating point numbers it becomes finite, albeit very large: the game is then finite, implying the existence of a NE, at least in mixed strategies, which however is hard to compute. It has to be kept into account also the fact that the optimization of a neural network can lead to a Local NE (LNE) because the problem is non-convex. We can then solve the equilibrium problem by computing the minimax of the payoffs. Loss functions are defined as:

$$L^{(D)} = -\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$
$$L^{(G)} = -L^{(D)},$$

where, the expectations are computed on mini-batches of data. For the very definition of cross-entropy loss function, $L^{(G)}$ can be simplified because all the samples are generated according to the same distribution:

$$L^{(G)} = \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))].$$

The value of the game is defined as $V(G,D) = -L^{(D)}$. Fixing a generator $\bar{G}$, the optimal discriminator can be computed maximizing D's utility, which corresponds to $V(\bar{G},D)$:

$$D^*(x) = arg\max_D\{V(D,\bar{G})\}.$$

Assuming that the model has infinite capacity in representing pdfs, we can study the convergence to a NE point in the space of pdf:

$$V(\bar{G},D) =$$
$$\int_x p_{data(x)}\log(D(x))dx + \int_z p_z(z)\log(1-D(\bar{G}(z)))dz =$$
$$= \int_x \Big[p_{data(x)}\log(D(x)) + p_g(x)\log(1-D(x))\Big]dx =$$

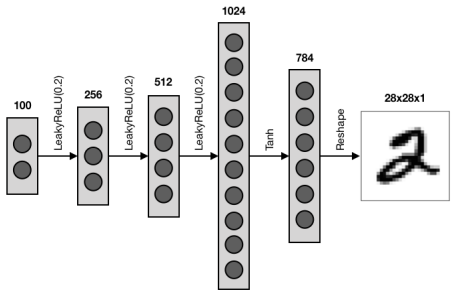

Figure 2: a high-level representation of G, the generator network, with 4 hidden layers. Each circle represents a perceptron unit in the network. A random vector of dimensionality 2, i.e. input noise, is transformed into a 28x28 grayscale image. The architecture presented is also used in our implementation of the model.

$$= \int_x L(x, D(x), \dot{D}(x))dx.$$

$D^*(x)$ must satisfy the Euler-Lagrange equation:

$$\frac{\partial L}{\partial D} = \frac{dL}{dx}\frac{\partial L}{\partial \dot{D}}$$

and since $\partial L/\partial\dot{D} = 0$ we get:

$$\frac{\partial L}{\partial D} = \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1-D(x)} = 0$$

that implies:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}.$$

Minimizing then over $p_g(x)$, a point of minimum is found for $p_g = p_{data}$, where $V(G^*,D^*) = -log(4)$.

Cross-entropy loss functions are particularly effective for discrimination tasks, but they're not for generation ones because, in the beginning of the training phase, when G is poor and D is able to correctly recognize generated samples, $log(1-D(G(z)))$ *saturates* to zero, thus resulting in a poor gradient. The learning in that case is too slow, but this problem can be avoided changing the loss function for G: instead of minimizing $log(1-D(G(z)))$, we can maximize $log(D(G(z)))$. This is formally defined as a Non-Saturating GAN (NS-GAN), where the losses to be minimized are:

$$L^{(D)} = -\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$
$$L^{(G)} = -log(D(G(z))).$$

This new game isn't zero-sum any more, but the same equilibrium point of the dynamics can be found as before, thus providing the same theoretical results.

III. EXPERIMENTAL SETTING

For our project, we implemented a version of NS-GAN which can be found at [**?**]. The training of the two neural networks, depicted in FIG.X, occurs after mini-batches of data are passed to them: for the discriminator network D, in a mini-batch there are usually the same number of true and of

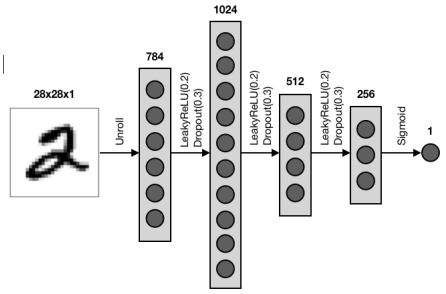

Figure 3: a high-level representation of D, the discriminator network, with 4 hidden layers. Each circle represents a perceptron unit in the network. 28x28 grayscale images are transformed into a scalar in range $[0, 1]$, representing the probability of the image being true. The architecture presented is also used in our implementation of the model.

fake images. We instead carried out the training with different configurations, where D is passed different true-to-fake data ratios: this is done in practice defining a parameter α that represents the portion of true data with respect to the size of the mini-batch. This results in a parametrisation of the loss functions as follows:

$$\begin{aligned} L^{(D)} = -\alpha \cdot \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] \\ -(1-\alpha) \cdot \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \\ L^{(G)} = ?\,? \end{aligned}$$

Performing the same analysis as before, we found that the optimal discriminator should have the form:

$$D^*_\alpha(x) = \frac{\alpha \cdot p_{data}(x)}{\alpha \cdot p_{data}(x) + (1-\alpha) \cdot p_g(x)}$$

and that the loss of the generator is minimized, again, when

$$p_g = p_{data},$$

from which the optimal D can be rewritten as

$$D^*_\alpha(x) = \alpha.$$

In support of these results, it can also be noticed that when setting $\alpha = 0.5$, i.e. when there are the same amount of true and fake images in a mini-batch, results are consistent with what found in II.

To evaluate the effects of this parametrisation, we preliminarily trained the NS-GAN models on two different 2D distributions, defined ad-hoc and affected by noise: first on a line described by a polynomial function and then on a circle. The values of α that we tested are $\alpha = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Distribution parameters in these cases are irrelevant because they would influence only the scale of the generated points and, anyway, the true generating distribution couldn't be compared to the one defined by G which is inferred implicitly. Once evaluated the two artificial distribution, we tested our code also on the MNIST dataset on the same values of α. All results are reported in IV.

IV. RESULTS

We first report here the results obtained on the two 2D datasets. In FIG.X the losses for both D and G are plotted, as functions of the epochs. It can be immediately noticed that for both data distributions, the highest loss for the discriminator is obtained with $\alpha = 0.5$ which foretells the training of a good generator (notice that for this α, as expected, $L^{(D)}$ converges to $\ln(2) = 0.693$). Anyway also with $\alpha = \{0.3, 0.7\}$ good results are achieved in this sense. From the generator's losses it can be noticed that, the lower α, the faster the loss convergence to the final value. An insight on how well are points generated is given in FIG.X, where for different values of α we can see how points are generated after 30000 training epochs: these results confirm what expected, that is better performances from more balanced number of true images per mini-batch.

When applying the model to the MNIST dataset, similar results are obtained FIG.X, with losses for the discriminator approaching this time 0.5 for $\alpha = \{0.3, 0.5, 0.7\}$. Also in this data space, as pointed out in the figure, when mini-batches are more balanced the D loss is higher, indicating better generators. The evolution of how numbers are generated is depicted in FIG.X, where for each α, every 20 and up to 100 epochs, a grid with 36 outputs are displayed. Also here for lower values of α, the generator outputs acceptable images much faster, as observed for the 2D case.

V. CONCLUSIONS

In this report we presented a review of GAN framework, as presented in the original paper [1], with a game theoretical approach, going through the resolution method to find a Nash equilibrium. We then proposed a parametrization of the loss function by varying the percentage of true images used for the training in each mini-batch, adapting the analysis previously presented to this new case. Then we tried to apply our own implementation of the model to three different datasets, two of which are generated ad-hoc and used as toy examples, while the last one is the MNIST dataset, well known in literature for this kind of tasks. Results obtained show that when mini-batches are more balanced, the generator model performs better, but also trainings with less true samples can be performed with good results, which opens the door to the use of GANs even for slightly smaller datasets.

Another possible way to organise mini-batches, that could be investigated in the future, is to introduce randomness in the percentage of true samples per mini-batch: α for example could stand for the probability of passing a true sample. The behaviour in this case should be similar, on average, to the one with α indicating a deterministic ratio.

REFERENCES

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[3] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. *arXiv*, 2017.

[4] F. A. Oliehoek, R. Savani, J. Gallego-Posada, E. van der Pol, E. D. de Jong, and R. Gross. GANGs: Generative Adversarial Network Games. *ArXiv e-prints*, December 2017.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[6] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242, 2016.