Towards Theoretically-Grounded GAN Training

Mohammed Ehab
ehab02@mit.edu

12/15/2023

Abstract

Since their invention in 2014, Generative Adversaral Networks (GANs) have been the most successful solution for generative modeling, only contested recently by diffusion models. However, they've been notoriously difficult to train, in good part due to training practices that are not supported by theory. The goal of this project is to work towards a more rigorous approach to GAN training.

1 Introduction

GANs have been one of the leading models in generative machine learning. Their employment of game theory to solve a problem that's not game-theoretical in nature is brilliant and has been called by Yann LeCun "the most interesting idea in the last 10 years in ML." But that use of game theory is exactly why their training is difficult: researchers have been using heuristics in their training that are not grounded in theory and lead to frustrating training.

In this report, we will discuss a few of these heuristics, with a focus on the training dynamics. We will start by proposing better training dynamics that have theoretical justification. We will then see why other heuristics used in the GAN literature prevent the dynamics from converging to something meaningful. We will finish by proposing solutions to be explored in future work.

The current code for the project can be found here.

2 GANs a la Goodfellow et. al.

GANs were first proposed as a solution for generative modeling in [GPAM+14]. In the GAN framework, there's a generator network G and a discriminator network D. The generator tries to take noise as input and turn it into data, while the discriminator takes an image and says how likely for it to be coming from the data distribution versus from G. They jointly train to solve the following problem:-

$$\min_{\theta_G} \max_{\theta_D} E_{x\sim p_{data}}[\log(D(x))] + E_{z\sim p_z}[\log(1 - D(G(z)))]$$

where p_z is some latent distribution, typically IID noise. This game has a unique Nash equilibrium where $G(z)$ has the same distribution as p_{data}, and D just outputs $\frac{1}{2}$ (in the limit where the networks can express any function).

The training procedure the authors proposed is the following: for k steps, perform stochastic gradient ascent on D's parameters; then for 1 step, perform stochastic gradient descent on G's parameters.

One reason GAN training is notoriously difficult is that while SGD for minimization problems is guaranteed to converge to a local optimum, these dynamics have very little theoretical guarantees. In fact, they've been shown to easily cycle or converge to spurious limit points [FVGP19]. Hence, we will investigate training GANs with stronger algorithms.

3 STON'R Algorithm

STON'R was proposed in [DGSZ22] to offer convergent dynamics for the following problem: find a local Nash equilibrium for

$$\min_{\theta \in [0,1]^n} \max_{\omega \in [0,1]^m} f(\theta, \omega)$$

where a local Nash equilibrium is a point (θ^*, ω^*) such that θ^* can't locally change to decrease f, and ω^* can't locally change to increase it, akin to a local optimum in standard optimization.

Here's an informal high-level overview of STON'R: we will be thinking of (θ, ω) as one big vector. We'll call a coordinate i satisfied if it can't be locally changed to decrease the function if it's one of the first n coordinates and can't be locally changed to increase the function otherwise (i.e. can't be locally changed to increase the respective player's utility). The goal of the algorithm will be to satisfy the coordinates one-by-one. Suppose that at the current iteration, the first i coordinates are satisfied. We will try to satisfy coordinate $i+1$ while keeping the first i coordinates satisfied. To do that, we'll run the following continuous-time dynamics: let $V_i(x) = (-1)^{[i \leq n]} \frac{\partial f}{\partial x_i}$; if we're currently at a point z, we will move infinitesimally in a unit direction d that satisfies $(\nabla V_j(z))^T \cdot d = 0$ for all $j \leq i$. This maintains the condition that the first i coordinates are satisfied. We can also set $d_j = 0$ for $j > i+1$ to keep them unchanged. This gives a set of equations that specifies d (up to direction, which should also be chosen carefully). Running the dynamics will terminate with one of 2 events:-
1. (Good event) coordinate $i+1$ is satisfied.
2. (Bad event) we hit the boundary.
In the good event, we move on to the next coordinate. In the bad event, we backtrack to the previous coordinate and try again from there. There are a few more details in the algorithm, but they won't be important for our discussion.

The unusual thing about STON'R is that it's unclear if it will even terminate. There's no clear monovariant, and even the number of satisfied coordinates can decrease as the algorithm runs. Instead, in its heart lies a topological fixed point argument for convergence. This already makes it quite different from the standard machine learning setting where a potential function decreases throughout gradient descent.

There are a couple reasons why implementing STON'R is not suitable for GANs. For starters, no neural network ever is trained neuron-by-neuron. The neurons are supposed to work together, and from a software engineering perspective, updating all of them simultaneously is much more efficient. More importantly, the problem STON'R solves is constrained while our problem isn't, and there's no reason to artificially constrain it. In fact, the existence of a boundary complicates STON'R and is the reason we need a fixed-point argument. We will show that we can adapt the good ideas from STON'R to our setting while dropping the complications.

4 Proposed Algorithm

Recall the optimization problem we're trying to solve:-

$$\min_{\theta_G} \max_{\theta_D} L(\theta_G, \theta_D)$$

where $L(\theta_G, \theta_D) = E_{x \sim p_{data}}[\log(D(x))] + E_{z \sim p_z}[\log(1 - D(G(z)))$. Let the best-response manifold $\mathcal{M}$ be the set of points (θ_G, θ_D) satisfying the following:-

$$\theta_D \in \operatorname*{argmax}_{\theta_D} L(\theta_G, \theta_D)$$

Then, our optimization problem becomes:-

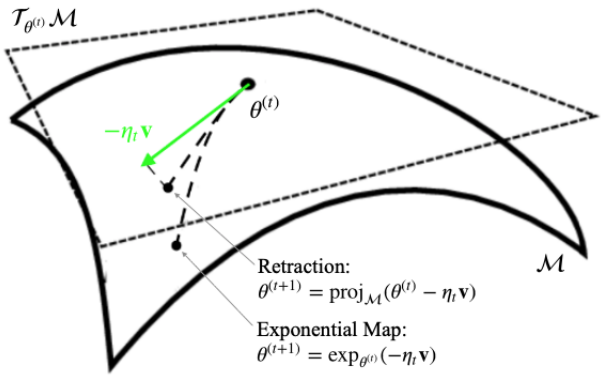
$$\min_{(\theta_G, \theta_D) \in \mathcal{M}} L(\theta_G, \theta_D)$$

which replaces a min-max with a normal minimization problem over a manifold!

The issue is that $\mathcal{M}$ is hard to get a handle on, but if we only care about local Nash equilibria, we can restrict our attention to θ_D that are local best-responses. Then, $\mathcal{M}$ can be defined implicitly by the equation $\nabla_{\theta_D} L(\theta_G, \theta_D) = 0$. We can then appeal to tools for optimization on manifolds to find a local Nash equilibrium.

5 Optimization on Manifolds

It turns out that we can run SGD to optimize on manifolds [Bon13]. The algorithm proceeds as follows: first, compute the (stochastic) gradient in $\mathbb{R}^n$. Then, project it onto the tangent space of the manifold. Finally, take a step and retract the point back onto the manifold.



To retract back onto the manifold, we can train the discriminator until optimality with normal SGD. Technically speaking, this algorithm only has guarantees if $\mathcal{M}$ is Riemannian and if the retraction used obeys some technical conditions. I deprioritized proving these results, and they're admittedly still on my to-do list.

6 Comparison with GDA and STON'R

In GDA, the generator and discriminator compete to optimize their losses. The generator can exploit weaknesses in the discriminator to improve its loss, not by actually producing better data, but by producing samples the discriminator can't distinguish well. In this algorithm, the generator and discriminator cooperate. The generator takes a gradient step knowing that the

discriminator would also improve in response, and the component of the gradient where it exploits the discriminator is projected away.

This algorithm is strongly tied to STON'R in the following way: the generator moves its parameters while keeping the discriminator satisfied in the same way STON'R keeps coordinates up to i satisfied. The dynamics we get from optimization over manifolds are the same as the ones from STON'R. This algorithm, however, is more on the macro level of networks instead of parameters. It doesn't have to worry about hitting the boundary, and in its heart, there is indeed a potential function that decreases.

7 Implementation Details and Initial Results

An important implementation detail lies in the projection step. Projecting naively would take $O(n^3)$ where n is the number of parameters in a neural network, which would take a massive amount of time. Instead, we use an iterative procedure: projection on the column space of a matrix A can be written as the following optimization problem:-

$$\min_x ||Ax - b||^2$$

It's not hard to show that this function is convex, and so we can minimize it with yet another gradient descent. The key idea is to warm-start using the result from the last iteration, since the projection should change continuously between steps.

The matrix we need to project onto is $\frac{\partial^2 L}{\partial \theta_G \partial \theta_D}$. The gradients from the iterative procedure can be computed very efficiently in PyTorch with the built-in Hessian-vector product functionality.

I started testing the algorithm on the MNIST dataset, and to my surprise, in the first run of the algorithm, the loss stayed steady. After some debugging, I discovered that the gradient w.r.t θ_G is almost 0 from the very beginning! This is surprising given than G still has all the learning to do.

It turns out that this is a well-documented issue with GANs that was even pointed out in the Goodfellow et. al. paper. When the discriminator is much better than the generator, it doesn't give useful gradients. Notice that this is exactly our setting, with the discriminator being optimal while the generator has random parameters.

The solution proposed in that paper was to replace the generator loss with $-\log(D(G(z)))$. Later, it was shown in [AB17] that the gradient indeed ought to vanish, and further that the new loss is not a great choice. In fact, it encourages the distributions to be different and encourages mode collapse. The same authors later (together with Soumith Chintala) proposed an alternative solution discussed in the next section.

8 Wasserstein GANs

Wasserstein GANs (WGANs) were proposed in [ACB17] to solve the issues above. Recall the Wasserstein distance:-

$$W_1(X, Y) = \inf_{\gamma \in \Gamma(X,Y)} E_{(x,y)\sim\gamma}|x - y|$$

Suppose we train the generator to minimize $W_1(G(z), p_{data})$. How can we design a discriminator for that?

By Kantorovich-Rubinstein duality:-

$$W_1(X, Y) = \sup_{||f||_L \leq 1} E_{x \sim X}[f(x)] - E_{y \sim Y}[f(y)]$$

where the supremum is over 1-Lipschitz functions. The function f is called the critic and will play the role of the discriminator. We will parameterize it as a neural network, allowing us to write the following optimization problem:-

$$\min_{\theta_G} \max_{\theta_f} E_{x \sim p_{data}}[f(x)] - E_{z \sim p_z}[f(G(z))]$$

The main caveat in this optimization problem is enforcing the Lipschitz constraint over f. The authors proposed limiting the weights of the network to come from $[-c, c]$ for this, and they enforced that by weight clipping. They admitted this is a terrible way to constrain the weights and encouraged interested researchers to find a better way, and it's worthy to note that in theory, STON'R would be a better way.

9 Wasserstein GANs with Gradient Penalties

A better approach for training WGANs is shown in [GAA+17]. In this paper, they show that the optimal critic f^* has gradient norm 1 almost everywhere, so they enforce a constraint on the gradient to have norm 1 instead of constraining the weights. They do so in a "soft" way by penalizing the gradient being far away from 1, modifying the discriminator loss to be the following:-

$$L_D = E_{z \sim p_z}[f(G(z))] - E_{x \sim p_{data}}[f(x)] + \lambda E_{x \sim \tilde{p}}[(||\nabla f(x)|| - 1)^2]$$

where the distribution $\tilde{p}$ is obtained as follows: let $z \sim p_z$, $x \sim p_{data}$, and $t \sim Unif([0, 1])$, then $\tilde{p} \sim tG(p_z) + (1 - t)x$. In words, they enforce the gradient have norm 1 on lines connecting real and fake images.

Since the generator and discriminator losses don't add up to 0 anymore, this is no longer a min-max problem. But that's okay, since our algorithm can solve general games and never used the fact that the game is zero-sum with the other loss function.

10 Second Round of Experiments

After incorporating WGAN-GPs into the code, the vanishing gradient issue was indeed fixed. However, the loss still didn't seem to be decreasing. The issue I conjectued to be causing this is the following: in standard ML, the loss is usually computed using a random minibatch of the data. But since the best-response manifold is defined in terms of that loss, the manifold was changing every iteration.

To fix this, I changed the loss computation to be over the entire data. I also fixed the randomness z used to generate the points where the gradient penalty is being evaluated on. The use of the entire dataset incurred a massive computational cost, and it took a few days to get around 30 epochs in.

Unfortunately, I still didn't get results after that, and debugging something that takes days to run is difficult. There are also plenty of moving component that could be going wrong. I decided then it's best to build the code up step by step from the simplest possible task: the underlying data distribution is just another Gaussian. To my surprise, even in the simplest case where it's a 1D standard Gaussian, the algorithm doesn't manage to generate that! I then wrote a theoretical analysis of what happens and found out that this is a fundamental issue with WGAN-GPs.

11 Case Study of Generating Gaussians

We'll give theoretical justification of why the algorithm didn't generate data from a Gaussian even in this simple case. Suppose the generator is just parameterized by one scaler θ_G such that $G(z) = z + \theta_G$, and suppose the discriminator is parameterized by one scaler θ_D such that $D(x) = \theta_D x$. Let's compute the best response of the discriminator:-

$$L_D(\theta_G, \theta_D) = \lambda(|\theta_D| - 1)^2 + \theta_D\theta_G$$

$$\nabla_{\theta_D} L_D(\theta_G, \theta_D) = 2\lambda(|\theta_D| - 1)sgn(\theta_D) + \theta_G = 0 \Rightarrow |\theta_D| = -\frac{\theta_G}{2\lambda sgn(\theta_D)} + 1$$

For $\theta_G < 2\lambda$ we have 2 solutions: $\theta_D = -\frac{\theta_G}{2\lambda} \pm 1$. This is unfortunate because it means that in the limit as $\lambda \to \infty$, $+1$ and -1 are both local best responses regardless of θ_G. The correct best response in WGANs without gradient penalties is $sgn(\theta_G)$ (and can be anything for 0), which would give the generator useful information. Instead, we have 2 different local best-response manifolds. Suppose we initially land on the $+1$ manifold. Computing the minimum yields:-

$$L_G(\theta_G, \theta_D) = -\theta_G\theta_D = \frac{\theta_G^2}{2\lambda} - \theta_G \Rightarrow \theta_G = \lambda$$

So $(\lambda, \frac{1}{2})$ is a local Nash equilibrium! Alas, WGAN-GPs allow for meaningless Nash equilibria that heavily depend on the hyperparameters, even in this simple case. And to make matters worse, by being stuck on one of the manifolds, the algorithm is unable to output the interesting local Nash equilibrium.

12 Better Gradient Penalties

The $\lambda(||\nabla f|| - 1)^2$ choice for the gradient penalty is quite arbitrary, and in fact doesn't quite achieve what we want to achieve. The point of enforcing the Lipschitz constant is that without it, for any critic f, we can multiply f by a constant c so that the WGAN loss, $L = E_{x\sim p_{data}}[f(x)] - E_{z\sim p_z}[f(G(z))]$, becomes cL. To prevent that from happening, we need a penalty term that scales linearly with c for $c \geq 1$ and doesn't penalize at all for $c \leq 1$, so the correct penalty to use is $\lambda ReLU(\nabla f - 1)$. Think of this in the following way: as long as $\nabla f \leq 1$, we incur no penalty, and only the actual loss contributes to L_D. Once we get to the boundary, the penalty term take over and prohibits ∇f from getting out. For any $\lambda \geq W_1(p_{data}, G(p_z))$, the optimal critic will indeed be the WGAN optimal critic, which is not true for WGAN-GP. Of course in practice, bigger λ means harder optimization.

With the new gradient penalty, I was able to get Gaussians to work, although I saw some oscillation around the optimal mean. The reason is that the discriminator wasn't being fully trained to optimality between iterations, so it took a few iterations before it started pushing the generator back towards the mean.

13 Future Work

There's clearly more investigation to be done, but at some point, I had to stop working and write the report, leaving this investigation for future work. Here are a few ideas we can explore from here:-

13.1 Continuing the Work on the ReLU Penalty

This is the obvious continuation point of the project from here. That being said, even with Gaussians, the training is a little rough. The algorithm has a lot of choices and hyperparameters, most importantly λ, multiple learning rates, when to stop the gradient descent used to project, and when to stop training the discriminator between iterations. For the sake of time, I set all of these to be a fixed number of iterations, but this is far from ideal for working with harder datasets. Hopefully with more relaxed work on the project, I'll look into better practices.

13.2 Function Approximator for Lipschitz-1 Functions

I have this idea less figured out but will include it for completeness. The difficulty with WGANs is that they use neural networks as general function approximators and then try to enforce a Lipschitz constraint on them. This makes sense with the success of neural networks as general approximators, but in this niche application, what if we seek a family of function approximators for Lipschitz-1 functions to begin with?

13.3 Better GANs

I also thought about different fundamental ideas for building GANs. Recently, I became aware of the Fourier-based metric between distributions defined as follows:-

$$d_2(P, Q) = \sup_{k \in \mathbb{R}^d \setminus \{0\}} \frac{|\phi_P(k) - \phi_Q(k)|}{||k||^2}$$

where $\phi_P(k)$ is the characteristic function of P. It was shown in [ACG+20] that this metric is equivalent to the Wasserstein-2 distance.

We can use the same high-level idea behind WGANs but with this metric instead of the Wasserstein metric. The critic here will be the vector k in the supremum. This idea has 2 major advantages over WGANs: the critic is just a vector instead of a neural network, and we don't have to enforce a Lipschitz constraint anymore. And furthermore, because the 2 metrics are equivalent, they should induce the same topology, so we should have the same convergence properties as WGANs!

Feedback and advice on the future work ideas would be greatly appreciated.

14 Discussion

I think it was amazing working on this project, but it's clear to me now that the original goal of perfect practical GAN training was too ambitious. Neither the GANs we currently have nor min-max optimization theory are developed enough to do this smoothly. On the GAN side, it's still open to develop a GAN that doesn't suffer from vanishing gradients but doesn't require an intractable optimization to even compute the best response. On the min-max optimization side, algorithms don't have nearly strong enough guarantees compared to SGD. SGD is able to converge with only an unbiased estimator of the gradient, allowing efficient training with minibatches. I doubt the same is true for our algorithm or STON'R, and developing something with that guarantee in mind is crucial for ML applications.

That being said, this is more than understandable. Computing local min-max equilibria in general is PPAD-hard [DSZ20], at least in the constrained regime. However, I do believe the generative modeling task has a special structure that can be exploited to circumvent that; we just haven't perfectly pinned it down yet.

References

[AB17] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks, January 2017. arXiv:1701.04862 [cs, stat].

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, December 2017. arXiv:1701.07875 [cs, stat].

[ACG+20] Gennaro Auricchio, Andrea Codegoni, Stefano Gualandi, Giuseppe Toscani, and Marco Veneroni. The Equivalence of Fourier-based and Wasserstein Metrics on Imaging Problems, May 2020. arXiv:2005.06530 [math-ph, stat].

[Bon13] Silvere Bonnabel. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, September 2013. arXiv:1111.5280 [cs, math, stat].

[DGSZ22] Constantinos Daskalakis, Noah Golowich, Stratis Skoulakis, and Manolis Zampetakis. STay-ON-the-Ridge: Guaranteed Convergence to Local Minimax Equilibrium in Nonconvex-Nonconcave Games, October 2022. arXiv:2210.09769 [cs, math].

[DSZ20] Constantinos Daskalakis, Stratis Skoulakis, and Manolis Zampetakis. The Complexity of Constrained Min-Max Optimization, September 2020. arXiv:2009.09623 [cs, math].

[FVGP19] Lampros Flokas, Emmanouil-Vasileios Vlatakis-Gkaragkounis, and Georgios Piliouras. Poincar\'e Recurrence, Cycles and Spurious Equilibria in Gradient-Descent-Ascent for Non-Convex Non-Concave Zero-Sum Games, October 2019. arXiv:1910.13010 [cs, math, stat].

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs, December 2017. arXiv:1704.00028 [cs, stat].

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. arXiv:1406.2661 [cs, stat].