Modified GAN training by using tools from game theory

Vaishnav K.V.
MTech AI
`vaishnavk@iisc.ac.in`

Abstract

Even though GANs can accurately model complicated distributions, they are known to be difficult to train due to instabilities caused by a difficult minimax optimisation problem. In the work named "An Online Learning Approach to GAN" from [1] , the author views the problem of training GANs as finding a mixed strategy in a zero-sum game.This paper will be the center of our discussion and other papers are useful in leading or motivating the problem ,the above paper trying to solve. The author proposes a novel training method named Chekhov GAN based on ideas from online learning.The paper has two main contributions, one in theory and other in practice. On the theory side, the author shows that the algorithm proposed provably converges to equilibrium for semi-shallow GAN architectures(a toy GAN architecture). On the practical side, the paper proposes an efficient heuristic guided by the theoretical results, which can be applied to commonly used deep GAN architectures.

1 Context and Motivation

1.1 Basic GAN Framework

1.1.1 A Brief

I will start by setting a stage first.Goodfellow's paper [2] on Generative Adversarial Networks (GANs) proposed a new deep learning architecture that involves two neural networks, a generator and a discriminator, which learn to generate realistic data from a random noise input. The generator tries to produce data that is indistinguishable from real data, while the discriminator tries to correctly classify the generated data as fake. Through this adversarial process, both networks improve their performance until the generator is able to produce realistic synthetic data. GANs have shown impressive results in various applications.In short,The generator network takes in a random noise input and tries to generate realistic data that can fool the discriminator network. The discriminator network, on the other hand, is trained to recognize the difference between real and synthetic data.

1.1.2 Formalising

Now Let's formalise the same. Let us denote the data distribution by $p_{\text{data}}(\mathbf{x})$ and the model distribution by $p_{\mathbf{u}}(\mathbf{x})$. A probabilistic discriminator is denoted by $h_{\mathbf{v}} : \mathbf{x} \to [0; 1]$ and a generator by $G_{\mathbf{u}} : \mathbf{z} \to \mathbf{x}$. The GAN objective is:

$$\min_{\mathbf{u}} \max_{\mathbf{v}} M(\mathbf{u}, \mathbf{v}) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log h_{\mathbf{v}}(\mathbf{x}) + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} \log(1 - h_{\mathbf{v}}(G_{\mathbf{u}}(\mathbf{z}))) \ . \tag{1}$$

Each of the two players (generator/discriminator) tries to optimize their own objective, which is exactly balanced by the loss of the other player, thus yielding a two-player zero-sum minimax game.

1.1.3 Comparing GAN With VAE

The fundamental difference between VAE (Variational Autoencoder)[3] and GAN (Generative Adversarial Network) lies in their architecture and training approach. The main difference between VAEs and GANs is that VAEs are based on the reconstruction and compression of input data, while GANs are based on a two-player game between the generator and the discriminator networks. VAEs are typically better suited for modelling continuous, structured data, while GANs are better suited for generating complex, realistic data like images or audio. Note that we are not solving any optimisation problem but instead finding a Nash equilibrium.

1.1.4 Different Perspectives

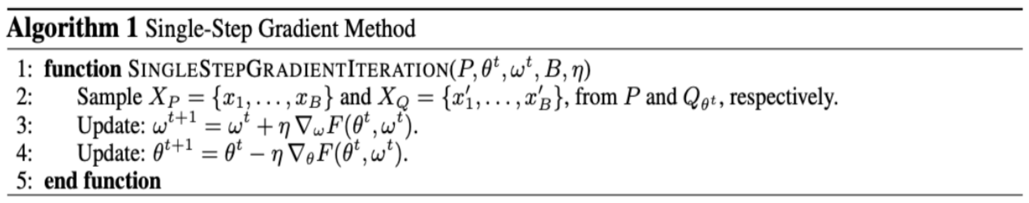
One way to look at the GAN objective is as a minimisation of f-divergence [4] for a particular instantiation for f, which is more of a statistical perspective. The paper introduces the concept of f-GAN. f-GAN can be seen as a generalization of Generative Adversarial Networks (GANs), which are a popular class of generative models. Unlike vanilla GAN, f-GAN introduces the use of f-divergences, a family of distance measures between probability distributions, to train the generator network. The f-GAN algorithm involves training a discriminator network to distinguish between real and generated samples, while simultaneously training a generator network to produce samples that minimize a particular f-divergence between the true data distribution and the generated distribution. By using f-divergences, f-GAN can accommodate a wider range of distance measures than GANs, which are limited to using the Jensen-Shannon divergence. Additionally, f-GAN provides a more principled and unified framework for training generative models. Overall, f-GAN can be seen as a generalization of GANs that provides a more flexible and principled approach to training generative models.

Another viewpoint is from a game theory side.GANs can be viewed as a two-player zero-sum game from a game theory perspective. In a two-player zero-sum game, the payoff of one player is the negative of the payoff of the other player. In the case of GANs, the two players are the generator and discriminator networks. The discriminator network aims to maximize its payoff by correctly classifying samples as either real or fake, while the generator network aims to minimize the discriminator's payoff by generating realistic samples that are difficult to classify as fake. The payoff of the discriminator network is the negative of the generator network's payoff. In other words, if the generator produces samples that are difficult to classify as fake, the discriminator's payoff decreases while the generator's payoff increases. On the other hand, if the generator produces low-quality samples that are easy to classify as fake, the discriminator's payoff increases while the generator's payoff decreases. The iterative training process of GANs can be seen as a repeated two-player zero-sum game, where the generator and discriminator networks compete against each other to improve their payoffs. The game ends when both networks have to converge to a Nash equilibrium, where the generator produces samples that are indistinguishable from real ones, and the discriminator cannot distinguish between real and generated samples.

Overall, viewing GANs as a two-player zero-sum game provides a useful framework for analyzing their training dynamics and understanding how the generator and discriminator networks interact with each other during training. Here in the discussion, we will focus on this perspective.

1.2 The Training Procedure For Vanilla GAN

The optimal solution here is a Pure Strategy Nash Equilibrium (PSNE); we need to compute that equilibrium point. Goodfellow's approach is -**Alternative gradient descent**. The generator and discriminator are trained alternatively in a series of iterations. During each iteration, the generator and discriminator are updated with different objectives. .We will update the parameters of both blocks alternatively. The idea is to do gradient descent and ascent in an alternative manner. We could use SGD to update. I have given a simple version of the same below.



Algorithm 1 Single-Step Gradient Method

```
1: function SINGLESTEPGRADIENTITERATION($P, \theta^t, \omega^t, B, \eta$)
2:     Sample $X_P = \{x_1, \ldots, x_B\}$ and $X_Q = \{x'_1, \ldots, x'_B\}$, from $P$ and $Q_{\theta^t}$, respectively.
3:     Update: $\omega^{t+1} = \omega^t + \eta \nabla_\omega F(\theta^t, \omega^t)$.
4:     Update: $\theta^{t+1} = \theta^t - \eta \nabla_\theta F(\theta^t, \omega^t)$.
5: end function
```

Figure 1: Alternative gradient descent

1.3 Problems With The Standard GAN(Alternating Gradient Descent):

Salimans et al. [5] claimed that even for small games, the approach could fail. The author also claims that GAN training is difficult and unstable, even when working with relatively simple and small datasets. The paper discusses several common challenges in GAN training, such as mode collapse, where the generator produces only a limited range of outputs, and vanishing gradients, where the gradients used for updating the generator and discriminator parameters become too small to be effective. These issues can lead to GAN training failing to converge or producing low-quality output.

The authors noted that even in scenarios where the training data is relatively simple and the network architecture is straightforward, these challenges can still arise. This underscores the need for careful tuning of hyperparameters, such as the learning rate and batch size, and the use of advanced training techniques, such as regularization and batch normalization, to improve the stability and convergence of GAN training. Another paper named "Unrolled Generative Adversarial Networks" from Metz et.al[6] discusses several problems with the vanilla GAN training method, which inspired the development of their algorithm. These problems include:

1. Mode collapse: In vanilla GANs, the generator can learn to produce a limited range of outputs, resulting in mode collapse. This occurs when the generator produces the same or similar outputs for many different inputs, leading to a lack of diversity in the generated data.

2. Oscillations and instability: Vanilla GAN training can be unstable and suffer from oscillations in which the generator and discriminator repeatedly outperform each other, leading to poor convergence and low-quality output.

3. Slow convergence: Vanilla GAN training can take a long time to converge, especially when working with large and complex datasets.

4. Difficulty in tuning hyperparameters: The success of GAN training depends heavily on the careful tuning of hyperparameters, including learning rates, batch sizes, and regularization parameters. Finding the optimal values for these hyperparameters can be time-consuming and challenging.

Here is a problem, and a lot of papers have come up with a variety of solutions. The paper proposes one such.

2 Proposal

2.1 Background

2.1.1 Two-Player Zero-Sum Game

Consider two players $\mathcal{P}$,$\mathcal{P}_2$, which may choose pure decisions among continuous sets $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively. A zero-sum game is defined by a function $M : \mathcal{K}_1 \times \mathcal{K}_2 \mapsto \mathbb{R}$ which sets the utilities of the players. Concretely, upon choosing a pure strategy $(\mathbf{u}, \mathbf{v}) \in \mathcal{K}_1 \times \mathcal{K}_2$ the utility of $\mathcal{P}_1$ is $-M(\mathbf{u}, \mathbf{v})$, while the utility of $\mathcal{P}_2$ is $M(\mathbf{u}, \mathbf{v})$. The goal of either $\mathcal{P}_1$/$\mathcal{P}_2$ is to maximize their worst-case utilities; thus,

$$\min_{\mathbf{u}\in\mathcal{K}_1} \max_{\mathbf{v}\in\mathcal{K}_2} M(\mathbf{u}, \mathbf{v}) \quad \textbf{(Goal of } \mathcal{P}_1\textbf{)}, \qquad \& \qquad \max_{\mathbf{v}\in\mathcal{K}_2} \min_{\mathbf{u}\in\mathcal{K}_1} M(\mathbf{u}, \mathbf{v}) \quad \textbf{(Goal of } \mathcal{P}_2\textbf{)} \tag{2}$$

2.1.2 Pure Strategy Nash Equilibrium

This definition of a game makes sense if there exists a point $(\mathbf{u}^*, \mathbf{v}^*)$, such that neither $\mathcal{P}_1$ nor $\mathcal{P}_2$ may increase their utility by unilateral deviation. Such a point $(\mathbf{u}^*, \mathbf{v}^*)$ is called a *Pure Nash Equilibrium*, which is formally defined as a point which satisfies the following conditions:

$$M(\mathbf{u}^*, \mathbf{v}^*) \leq \min_{\mathbf{u}\in\mathcal{K}_1} M(\mathbf{u}, \mathbf{v}^*), \;\; \& \;\; M(\mathbf{u}^*, \mathbf{v}^*) \geq \max_{\mathbf{v}\in\mathcal{K}_2} M(\mathbf{u}^*, \mathbf{v}) \, .$$

2.1.3 Mixed Strategy Nash Equilibrium

While a pure Nash equilibrium does not always exist, the pioneering work of Nash [**?**] established that there always exists a *Mixed Nash Equilibrium* (MNE or simply equilibrium), i.e., there always exist two distributions $\mathcal{D}_1, \mathcal{D}_2$ such that,

$$\mathbb{E}_{(\mathbf{u},\mathbf{v})\sim\mathcal{D}_1\times\mathcal{D}_2}[M(\mathbf{u}, \mathbf{v})] \leq \min_{\mathbf{u}\in\mathcal{K}_1} \mathbb{E}_{\mathbf{v}\sim\mathcal{D}_2}[M(\mathbf{u}, \mathbf{v})], \; \& \; \mathbb{E}_{(\mathbf{u},\mathbf{v})\sim\mathcal{D}_1\times\mathcal{D}_2}[M(\mathbf{u}, \mathbf{v})] \geq \max_{\mathbf{v}\in\mathcal{K}_2} \mathbb{E}_{\mathbf{u}\sim\mathcal{D}_1}[M(\mathbf{u}, \mathbf{v})] \, .$$

Finding an exact MNE might be computationally hard, and we are usually satisfied with finding an approximate MNE. This is defined below,

2.1.4 Epsilon-MSNE

Let $\varepsilon > 0$. Two distributions $\mathcal{D}_1, \mathcal{D}_2$ are called ε-MNE if the following holds,

$$\mathbb{E}_{(\mathbf{u},\mathbf{v})\sim\mathcal{D}_1\times\mathcal{D}_2}[M(\mathbf{u}, \mathbf{v})] \leq \min_{\mathbf{u}\in\mathcal{K}_1} \mathbb{E}_{\mathbf{v}\sim\mathcal{D}_2}[M(\mathbf{u}, \mathbf{v})] + \varepsilon,$$
$$\mathbb{E}_{(\mathbf{u},\mathbf{v})\sim\mathcal{D}_1\times\mathcal{D}_2}[M(\mathbf{u}, \mathbf{v})] \geq \max_{\mathbf{v}\in\mathcal{K}_2} \mathbb{E}_{\mathbf{u}\sim\mathcal{D}_1}[M(\mathbf{u}, \mathbf{v})] - \varepsilon \, .$$

2.1.5 Nash Existence Theorem

The Nash existence theorem is a fundamental result in game theory that states that every finite strategic game (i.e., a game in which players have a finite number of pure strategies) has at least one mixed-strategy Nash equilibrium. But Nash's theorem doesn't say anything about how to compute the same.[7]

2.1.6 FTRL Algorithm

The Follow-the-Regularized-Leader (FTRL) algorithm is a popular online learning algorithm that has been widely studied in the machine learning literature. But to start with, I will give a non-technical brief on the online learning framework. Online learning, also known as incremental learning, is a machine learning paradigm in which the model is trained on a stream of data that arrives continuously over time. In online learning, the model updates its parameters continuously as new data becomes available rather than being trained on a fixed dataset all at once. In short, Online learning is a sequential decision-making framework in which a player aims at minimizing a cumulative loss function revealed to her sequentially. The source of the loss functions may be arbitrary or even adversarial, and the player seeks to provide worst-case guarantees on her performance.

Formally, this framework can be described as a repeated game of T rounds between a player $\mathcal{P}_1$ and an adversary $\mathcal{P}_2$. At each round $t \in [T]$:

1. $\mathcal{P}_1$ chooses a point $\mathbf{u}_t \in \mathcal{K}$ according to some algorithm $\mathcal{A}$,
2. $\mathcal{P}_2$ chooses a loss function $f_t \in \mathcal{F}$,
3. $\mathcal{P}_1$ suffers a loss $f_t(\mathbf{u}_t)$, and the loss function $f_t(\cdot)$ is revealed to her.

The adversary is usually limited to choosing losses from a structured class of objectives $\mathcal{F}$, most commonly linear/convex losses. Also, the decision set $\mathcal{K}$ is often assumed to be convex. The performance of the player's strategy is measured by the *regret*, defined as,

$$\text{Regret}_T^{\mathcal{A}}(f_1, \ldots, f_T) = \sum_{t=1}^{T} f_t(\mathbf{u}_t) - \min_{\mathbf{u}^* \in \mathcal{K}} \sum_{t=1}^{T} f_t(\mathbf{u}^*) \ . \tag{3}$$

Thus, the regret measures the cumulative loss of the player compared to the loss of the *best-fixed decision in hindsight*. A player aims at minimizing her regret, and we are interested in *no-regret* strategies for which players ensure an $o(T)$ regret for any loss sequence .A regret which depends linearly on T is ensured by any strategy and is, therefore, trivial.

While there are several no-regret strategies, an important class of those belongs to **Follow-the-Regularized-Leader (FTRL)** algorithm where

$$\mathbf{u}_t = \arg\min_{\mathbf{u} \in \mathcal{K}} \sum_{\tau=1}^{t-1} f_\tau(\mathbf{u}) + \eta_t^{-1} R(\mathbf{u}) \qquad \textbf{(FTRL)} \tag{4}$$

FTRL takes the accumulated loss observed up to time t and then chooses the point in $\mathcal{K}$ that minimizes the accumulated loss plus a regularization term $\eta_t^{-1} R(\mathbf{u})$. The regularization term prevents the player from abruptly changing her decisions between consecutive rounds. This property is often crucial to obtaining no-regret guarantees. Note that FTRL is not always guaranteed to yield no regret and is mainly known to provide such guarantees in the setting where losses are linear/convex [8].

2.1.7 Sub-linear Regret For FTRL Algorithm

In the case of FTRL with convex loss, it has been shown that the algorithm can achieve sublinear regret under certain conditions on the convex loss function and the sequence of data presented to the algorithm. In particular, the regret is bounded by O(sqrt(T*log(T))), which means that the regret grows slower than linearly with the number of time steps T, but faster than logarithmically. Sublinear regret means that the regret grows slower than linearly with the number of time steps (or iterations) of the algorithm. Specifically, if the regret grows no faster than O(sqrt(T)) for a sequence of T time steps, then the algorithm is said to have sublinear regret. This result is significant because it shows that the FTRL algorithm can achieve good performance even when presented with large amounts of data over long periods of time. It's a very important result in the online learning literature.

Note :

- In online learning literature, algorithms that achieve sublinear regrets are called a no-regret algorithms.
- If we have an algorithm which can compute an approximate MSNE (i.e. eps-MSNE) using the FTRL-like approach, it would be great. This is exactly what Freund and Schapire's work proposes.

2.1.8 No-regret Algorithm For Approximate MSNE

In zero-sum games, no-regret algorithms may be used to find an approximate MNE. Unfortunately, computationally tractable no-regret algorithms do not always exist. An exception is the setting when

M is convex-concave. In this case, the players may invoke the powerful no-regret methods from online convex optimization to (approximately) solve the game. This seminal idea was introduced in [9], where it was demonstrated how to invoke no-regret algorithms during T rounds to obtain an approximation guarantee of $\varepsilon = O(1/\sqrt{T})$ in zero-sum matrix games. This was later improved by [10], demonstrating a guarantee of $\varepsilon = O(\log T/T)$. The result that we are about to present builds on the scheme of [9].

2.1.9 Classifying GAN Based On Architecture

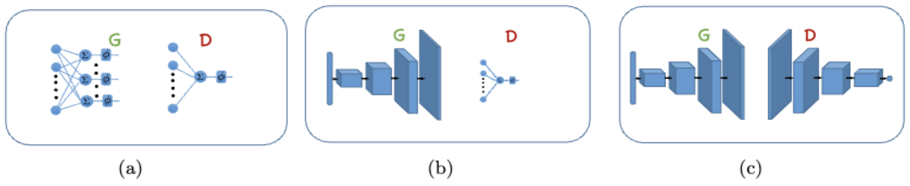


Figure 2: Three types of GAN architectures. (a) shallow. (b) semi-shallow (c) Right: Deep.

We could classify the GAN architectures into 3 classes. They are shallow,semi-shallow and deep. It is based on the extent of parametrisation we opt for.The most elementary architecture that one might consider is a shallow one, e.g. a GAN architecture which consists of a single-layer network as a discriminator and a generator with one hidden layer. However, one typically requires a powerful generator that can model complex data distribution. This leads us to consider a semi-shallow architecture where the generator is any arbitrary network. Deep means both generator as well as the discriminator are arbitrarily powerful networks.

2.2 Theory Side:

The author claims that a semi-shallow GAN setting with some additional conditions will induce an interesting game structure which we denote as semi-concave.Formally a game, M, is semi-concave if for any fixed $\mathbf{u}_0 \in \mathcal{K}_1$ the function $g(\mathbf{v}) := M(\mathbf{u}_0, \mathbf{v})$ is concave in $\mathbf{v}$. Here the additional condition puts a restriction on the activation function for the Discriminator. The following is a proposition:

Proposition 1. *Consider the GAN objective in Eq.* (1) *and assume that the adversary is a single-layer with a sigmoid activation function, meaning* $h_{\mathbf{v}}(\mathbf{x}) = 1/(1 + \exp(-\mathbf{v}^\top \mathbf{x}))$, *where* $\mathbf{v} \in \mathbb{R}^n$. *Then the GAN objective is concave in* $\mathbf{v}$.

For this interesting game structure (i.e. semi-concave), the author proposes the following algorithm:

This algorithm can be seen as an instantiation of the scheme derived by Schaphire, with specific choices of the online algorithms A1 and A2 used by the players. Note that both A1 and A2 are two different instances of the FTRL algorithm, which I have discussed earlier. First, I will give a brief on the flow of the algorithm then we could discuss a theorem which gives theoretical guarantees associated with it.

2.2.1 Flow Of The Algorithm

First, note that each player calculates a sequence of T points based on an online algorithm $\mathcal{A}_1/\mathcal{A}_2$. Interestingly, the sequence of (loss/reward) functions given to the online algorithm is based on the game objective M, and also on the decisions made by the other player. So the decision made by each player (through FTRL) is a function of the decisions already made by the other player. The generator and discriminator correspondingly it is the maximisation and minimisation problem resp. Accordingly, the definition of regret will change.To illustrate, consider the sequence of losses that player $\mathcal{P}_1$ receives, denoted as $f_t(\mathbf{u}) := M(\mathbf{u}, \mathbf{v}_t)t \in [T]$. Upon completion of T rounds, two

```
Algorithm 1 CHEKHOV GAN
  Input: #steps $T$, Game objective $M(\cdot,\cdot)$
  for $t = 1 \dots T$ do
    Calculate:

      (Alg. $\mathcal{A}_1$)  $\mathbf{u}_t \leftarrow \text{FTRL}_1(f_0,\dots,f_{t-1})$  &  (Alg. $\mathcal{A}_2$)  $\mathbf{v}_t \leftarrow \text{FTRL}_2(g_0,\dots,g_{t-1})$

    Update:    $f_t(\cdot) = M(\cdot,\mathbf{v}_t)$  &  $g_t(\cdot) = M(\mathbf{u}_t,\cdot)$
  end for
  Output mixed strategies: $\mathcal{D}_1 \sim \text{Uni}\{\mathbf{u}_1,\dots,\mathbf{u}_T\}$, $\mathcal{D}_2 \sim \text{Uni}\{\mathbf{v}_1,\dots,\mathbf{v}_T\}$.
```

Figure 3: Proposed Algorithm for GAN training

mixed strategies denoted as $\mathcal{D}_1$ and $\mathcal{D}_2$ are obtained. These strategies correspond to the uniform distributions over the sets of online decisions, namely $\mathbf{u}_t t \in [T]$ and $\mathbf{v}_{t t\in[T]}$, respectively. Note that the first decision points $\mathbf{u}_1, \mathbf{v}_1$ are set by $\mathcal{A}_1, \mathcal{A}_2$ before encountering any (loss/reward) function, and the dummy functions $f_0(\mathbf{u}) = 0, g_0(\mathbf{v}) = 0$. This is defined in order to not through any error as per the algorithm. But from an implementation point of view, this is equivalent to starting the execution from $f_1(\mathbf{u}), g_1(\mathbf{v})$

The author proposes the following theorem, which provides guarantees for semi-concave games:

Theorem 1. *Let $\mathcal{K}_2$ be a convex set. Also, let M be a semi-concave zero-sum game, and assume M is L-Lipschitz continuous. Then upon invoking Alg.* **??** *for T steps, using the FTRL versions $\mathcal{A}_1, \mathcal{A}_2$, appearing below, it outputs mixed strategies $(\mathcal{D}_1, \mathcal{D}_2)$ that are ε-MNE, where $\varepsilon = O(1/\sqrt{T})$.*

$$(\mathcal{A}_1) \quad \mathbf{u}_t \leftarrow \operatorname*{argmin}_{\mathbf{u}\in\mathcal{K}_1} \sum_{\tau=0}^{t-1} f_\tau(\mathbf{u}) \qquad \& \qquad (\mathcal{A}_2) \quad \mathbf{v}_t \leftarrow \operatorname*{argmax}_{\mathbf{v}\in\mathcal{K}_2} \sum_{\tau=0}^{t-1} \nabla g_\tau(\mathbf{v}_\tau)^\top \mathbf{v} - \frac{\sqrt{T}}{2\eta_0}\|\mathbf{v}\|^2$$

It is worth noting that the algorithm $\mathcal{A}_1$ corresponds to the follow-the-leader scheme, which is equivalent to FTRL without any regularization. Similarly, the discriminator $\mathcal{A}_2$ also employs the FTRL scheme. It is crucial to observe that as the number of iterations T increases, the accuracy of the approximation ϵ improves. Consequently, for sufficiently large T, it becomes possible to obtain an approximation that is arbitrarily good.

2.2.2 Proof Sketch

The proof relies on a theorem by Freund (1999) that shows if two strategies, $\mathcal{A}_1$ and $\mathcal{A}_2$, ensure no regret, then they converge to an approximate MNE. The game being concave with respect to $\mathcal{P}_2$, the FTRL version of $\mathcal{A}_2$ used in Theorem 1 is a no-regret strategy. However, showing that $\mathcal{A}_1$ is also a no-regret strategy is challenging, especially for semi-concave games that lack special structure. Previous work has shown that there is no efficient no-regret algorithm $\mathcal{A}_1$ in the general case where the loss sequence is arbitrary. However, the loss sequence, in this case, follows a special sequence based on the discriminator's choices. The discriminator decision sequence has a property that stabilizes the loss sequence, enabling us to establish no-regret for $\mathcal{A}_1$.

2.3 Practical Side:

2.3.1 Assumptions

Some assumptions/restrictions are taken while proposing the practical version of the algorithm. Guided by the theoretical results, got some heuristics which we apply to commonly used deep GAN architectures. The following assumptions/restrictions are taken while proposing the practical version of the algorithm.

- Use FTRL for both players.
- Take the gradient step each time instead of finding the argmin or argmax
- Use a Queue of finite length.
- The latest generator is used for generating new samples.

2.3.2 The Modified Algorithm (Implementable Version)

```
Algorithm 2 Practical CHEKHOV GAN
  Input: #steps $T$, Game objective $M(\cdot,\cdot)$, number of past states $K$, spacing $m$
  Initialize: Set loss/reward $f_0(\cdot) = 0, g_0(\cdot) = 0$, initialize queues $\mathcal{Q}_1$.insert($f_0$), $\mathcal{Q}_2$.insert($g_0$)

  for $t = 1 \dots T$ do
    Update generator and discriminator based on a minibatch of noise samples and data samples:

    $\mathbf{u}_{t+1} \leftarrow \mathbf{u}_t - \nabla_{\mathbf{u}_t}\left(\frac{1}{|\mathcal{Q}_1|}\sum_{f\in\mathcal{Q}_1} f(\mathbf{u}) + \frac{C}{\sqrt{t}}\|\mathbf{u}\|^2\right)$ & $\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t - \nabla_{\mathbf{v}_t}\left(\frac{1}{|\mathcal{Q}_2|}\sum_{g\in\mathcal{Q}_2} g(\mathbf{v}) - \frac{C}{\sqrt{t}}\|\mathbf{v}\|^2\right)$

    Calculate: $f_t(\cdot) = M(\cdot,\mathbf{v}_t)$ & $g_t(\cdot) = M(\mathbf{u}_t,\cdot)$
    Update $\mathcal{Q}_1$ and $\mathcal{Q}_2$ (see main text or Algorithm 3 in the appendix)
  end for
```

Figure 4: An implementable version of the proposed algorithm

2.3.3 Algorithm For Updating The Queue :

- This is the A3 that the proposed algorithm refers to. And A1 and A2 refer to the proposed and practical versions of the algorithm, respectively.
- **There is an error in the specified algorithm**. But from implementation, I came to the conclusion the way in which the queue is updated doesn't have a significant impact, at least for the dataset that I have tested with (CelebA and a mixture of Gaussian)

```
Algorithm 3 Update queue for Algorithm $\mathcal{A}_1$ and $\mathcal{A}_2$
  Input: Current step $t$, $m > 0$
  if ($t \mod m == 0$ and $|\mathcal{Q}|) == K$) then
    $\mathcal{Q}$.remove_last()
    $\mathcal{Q}$.insert($f_t$)
    $m = m + inc$
  else
    $\mathcal{Q}$.replace_first($f_t$)
  end if
```

Figure 5: Algorithm for Queue updation

3 Implementation

In the implementation, the traditional initialization step has been replaced with selecting the optimal parameter for the corresponding regularizer. Additionally, there are two separate queues maintained - one for the generator and the other for the discriminator. For the implementation of the system, the architecture of DC-GAN [11] is utilized. An important takeaway is: **The algorithm can be seen as an extension of the standard GAN training procedure.** If I instantiate the queue length as one and remove the initialisation step, we could obtain back the Vanilla training procedure.

3.1 Experiments

The authors are comparing the performance of the proposed GAN algorithm with respect to the standard GAN's performance with the help of MNIST, CelebA, and the multi-modal Gaussian dataset. The comparison is with respect to Stability during the training, Sample diversity and Mode collapse. This is the configuration that I have used for the training: batch-size = 128 image-size = 64,64 number of channels = 3 latent dimension = 100 epochs=20 lr (the parameter in Adam optimiser)= 0.0002 K=10 C=0.01. For the Gaussian dataset mixture, use a latent space dimension as 2. I have implemented GAN training using CelebA dataset and generated 100 images using the trained generator. The noise follows Normal distribution. I have given the images generated using standard GAN and the proposed approach. For GAN, the DC GAN architecture is used.

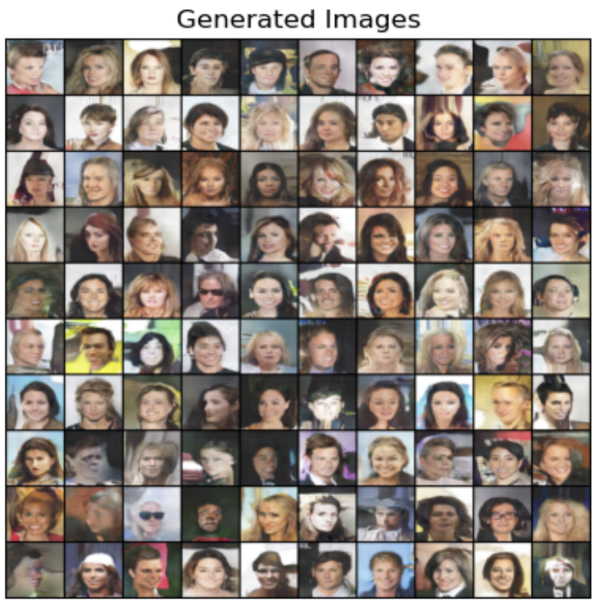


Figure 6: 10*10 Image generated with vanilla GAN trained on CelebA dataset

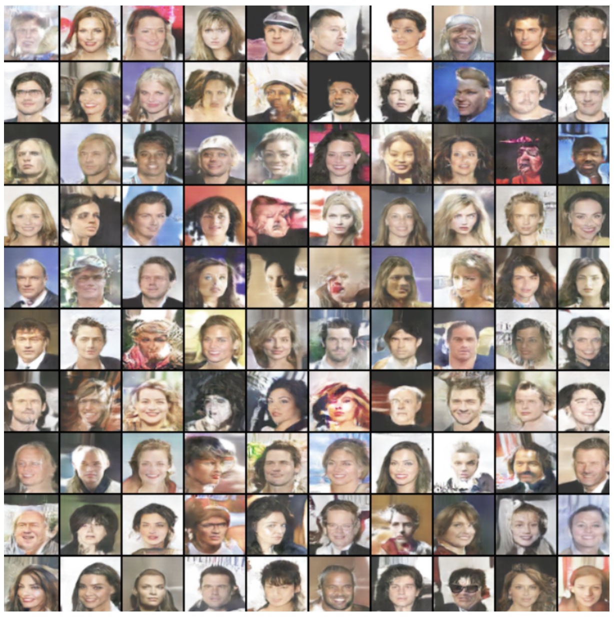


Figure 7: 10*10 Image generated with Chekhov GAN trained on CelebA dataset

Note: With the above samples, qualitative comments on Stability during the training, Sample diversity can be made, but not sure about modal collapse.

To analyse the improvement in the modal collapse aspect, I have used multi-modal Gaussian. The heat map of the dataset is given in the figure. Symmetrical architectures are used for the Generator as well as for the Discriminator.

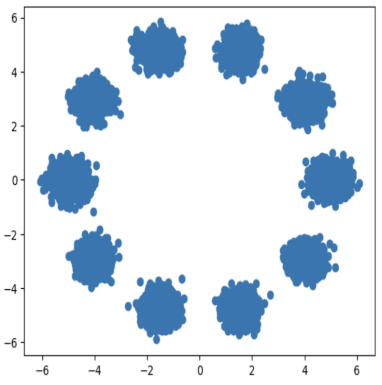


Figure 8: Heatmap for the training data

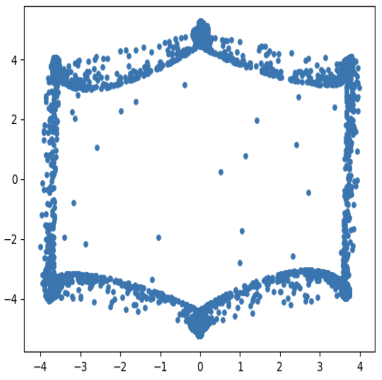


Figure 9: Heat map for the generated samples with vanilla GAN

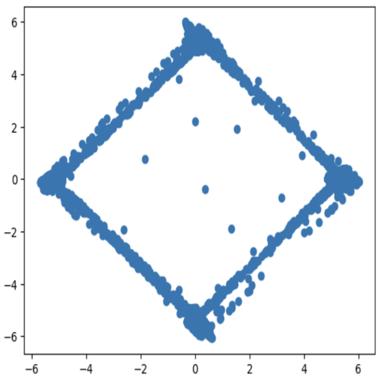


Figure 10: Heat map for the generated samples with Chekhov GAN

4 Learning/Takeaways

While implementing the paper came a set of challenges and, thereby, lessons to learn. There were no implementations available as the reference. So I have decided to build on top of standard GAN architectures, and the code is made available on "paper with code" as well as in the following GitHub repo: For more information, visit `https://github.com/vaishn99/modified-GAN-training`. For the same problem, multiple solutions(Papers) were available. But I couldn't get if there are any connections across the solutions, whether one implies the other. There is a mistake in the queue updation algorithm given in the paper. But the organisation of the paper is really good, and it is appreciable. Here multiple loss terms contribute to the final loss. We need to see how PyTorch compute the gradient with respect to all these terms for a single update. There are some nuances we need to consider[12], which happen because of the inherent design of the Pytorch framework. The most important lesson is the following: Building up theory-to-back concepts in deep learning is hard. Each time we update the parameters, the entire structure changes, which makes things challenging.

5 Conclusion

It is known that training a GAN is hard and can lead to other problems like mode collapse, instability etc. Hence there is a problem. Multiple solution concepts have been proposed, and a paper named "An Online Learning Approach to GAN" uses tools from game theory to improve the training procedure. In short, the idea is as follows: Instead of finding a PSNE, we will find the MSNE.To find the MSNE, we will use the FTRL algorithm from the online learning literature. This approach is interesting because we have converted a problem of finding the Nash equilibrium to solving an optimisation problem. But from the implementation section, as I have mentioned, this training approach can be seen as a direct extension of the vanilla GAN training method. In short, we went through the game theoretical way and came up with an algorithm to find an approximate MSNE. Then came up with a practical version of the proposed algorithm. Surprisingly, the practical version of the algorithm is a natural extension of the vanilla GAN training algorithm. This is quite interesting but surprising too.

References

[1] Paulina Grnarova, Kfir Y. Levy, Aurelien Lucchi, Thomas Hofmann, and Andreas Krause. An online learning approach to generative adversarial networks, 2017.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[4] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization, 2016.

[5] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[6] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2016.

[7] Constantinos Daskalakis. On learning algorithms for nash equilibria. `https://people.csail.mit.edu/costis/learning.pdf`.

[8] Elad Hazan. Introduction to online convex optimization, 2021.

[9] Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

[10] Constantinos Daskalakis, Alan Deckelbaum, and Anthony Kim. Near-optimal no-regret algorithms for zero-sum games. volume 92, pages 235–254, 10 2011.

[11] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[12] apaszke. How to combine multiple criterions to a loss function? `https://discuss.pytorch.org/t/how-to-combine-multiple-criterions-to-a-loss-function/348`.