ESTR3114

CHAN Yuk Kit
1155193356

HUI Man Hei
1155194718

Abstract

Generative Adversarial Networks (GANs) have emerged as a type powerful unsupervised learning framework for generative modeling, being able to learn and generate complex images, audio and other data. However, they are also known for being notoriously difficult to train due to its adversarial nature between the generator and discriminator, which leads to high instability. This paper explores various optimization strategies for GANs, first investigating the effectiveness of vanilla GANs and Wasserstein GANs, along with three basic optimization algorithms: Stochastic Gradient Descent, RMSProp, and Adam. We also study alternative models, namely Consensus and Optimistic Optimizations. Rigorous experiments are carried out to evaluate the performance difference between different algorithms, highlighting their advantages and the importance of an appropriate optimization method. This report contributes to the understanding of the optimization in GANs, providing useful insights in the training process of adversarial generative models.

1 Introduction

Generative Adversarial Networks (GANs) can be conceptualized as addressing a statistical inference problem, where the objective is to approximate the original data distribution using real-world examples. In the image generation problem, this involves producing new images that are statistically similar to a given dataset of real images. GAN is one of the most famous networks in the generation field, known for its incredible ability to learn complex data distributions [1].

Despite its promising potentials, the training of GANs have remained challenging due to its unsupervised and adversarial nature, which leads to problems like mode collapse and vanishing gradients, making it challenging for the model to achieve convergence [1]. One common alternative, Wasserstein GAN (WGAN), is introduced to improve upon the common problems in GANs by providing a more meaningful learning curve [2].

In this paper, we will study three different optimizing algorithms: Stochastic Gradient Descent (SGD), Root Mean Squared Propagation (RMSProp) and Adaptive Moment Estimation (Adam). We also investigate two alternative algorithm for the framework: Consensus and Optimistic, and carry out experiments to compare the performance of different optimization strategies with MNIST [3] and CelabA [4] dataset. We hope this can provide insight on the optimization of GANs.

2 Basics of GAN

Goodfellow et al. propose the idea of GAN, an unsupervised adversarial generative network framework [5]. The generative model and the discriminative model are in a zero-sum competition. The generator tries to create samples that mimic the real data, such as counterfeiters trying to produce fake currency. Meanwhile, the discriminator model seeks to distinguish these fakes samples from genuine data, similar to how the police detect counterfeit bills [5]. The two models keep on competing and improve their generation/discrimination overtime. The network is considered successful if the generator can produce indistinguishable counterfeits.

2.1 Numeric of GAN

The GAN problem can be expressed as a min-max (minimax) problem with the following value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(D(x))] + \mathbb{E}_{z \sim p_{\mathbf{z}}(z)}[\log(1 - D(G(z)))]$$

The generator $G(z)$ generates data with a random noise input z over the distribution $p_z(z)$; while the discriminator $D(x)$ returns the probability of its input x being real, with $p_{\text{data}}(x)$ representing the real data distribution. The discriminator D tries to maximize the value of V, while the generator G attempts to minimize it. Commonly, gradient descent[1] is performed in attempt to achieve optimality.

3 Optimization in GAN

In this section, we will study various methods of optimizing a GAN. We will first explain the traditional and common algorithms for the optimization process: Stochastic Gradient Descent (SGD), Root Mean Squared Propagation (RMSProp) and Adaptive Momentum Estimation (Adam). Then we will analyze alternative approaches to the GAN problem, namely, Consensus and Optimistic Optimization, and Wasserstein GAN. We also discuss the challenges of a vanilla GAN and compare the different approaches to the standard GAN.

3.1 Traditional Algorithms

3.1.1 Stochastic Gradient Descent

In the original paper, Goodfellow et al. suggest using mini-batch SGD to be the training algorithm [5]. By using a small batch of sample for each update, basic mini-batch SGD increases the convergence speed and introduce noise to improve performance. For parameters θ_d of discriminator D and θ_g of generator G, the algorithm samples mini-batch for x and z, and updates the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right],$$

then samples another mini-batch z to update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

However, vanilla mini-batch descent does not guarantee a great performance as its gradient may be trapped in saddle points, leading to a vanished gradient [6]. It is also difficult to choose a proper learning rate, and different parameters may wish to have a different learning rate [7]. Different learning rules are required to improve these problem, as we will introduce below.

3.1.2 RMSProp

RMSProp, or Root Mean Squared Propagation, is an algorithm with adaptive learning rate proposed by Geoff Hinton is his lecture notes[2] [7]. The learning rate is calculated each update, by dividing the default learning rate η with the exponentially decaying root mean square of the gradients g with fraction β plus a smoothing term ϵ, which avoids division by zero (commonly set to 10^{-8}). The formulas are as follows:

$$E[g^2]_t = \beta_1 E[g^2]_{t-1} + (1 - \beta_1) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

[1] Note that descent is only done for generator G; gradient ascent is performed for the discriminator D as it aims to maximize function V.

[2] `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

Hinton suggests β to be 0.9, and Ruder suggests η to be 0.001 [7]. This adaptive method means that the learning rate will be adjusted inversely proportional to the gradient, so the update rate is lower when the gradient is high and vice versa, leading to a more stable training and improved convergence speed.

3.1.3 Adam

Before introducing Adaptive Momentum Estimation (Adam), we have to first understand a term called momentum[8]. The momentum term v_t is calculated by the sum of the current gradient g_t with a learning rate $-\eta$ plus a fraction β of the past momentum v_{t-1}. Carrying the exponentially decaying average of the past gradients, it is an analogy to the concept of momentum and potential energy in real life physics. It can be expressed with the following formula:

$$v_t = \beta v_{t-1} - \eta g_t$$

The parameters are then updated with the standard formula $\theta_{t+1} = \theta_t + v_t$. By carrying past gradients, the sum of vectors with different directions may lead to a smoother and quicker convergence. More importantly, the optimizer is able to maintain its velocity when approaching local minima and saddle points, which may allow it to escape these points to seek for the global optimum.

Adam [9] combines RMSProp and momentum, earning the benefits of both algorithms, and can be expressed via the following formulas, where the variables carry meaning similar to the previous cases:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

The first momentum m_t is calculated similar to momentum, an exponentially decaying average of past gradients; and the second momentum v_t is calculated similar to that in RMSProp, an exponentially decaying average of past squared gradients. The terms are then corrected with the following formulas to avoid bias towards zero, especially when the decay rates β_1 and β_2 are small (close to 1):

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

The parameters are then updated similar to that in RMSProp:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

The authors propose that $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [9]. It is obvious that Adam contains the advantages of both RMSProp and momentum, improving the common problems of vanilla SGD.

3.2 Consensus Optimization

Mescheder et al. demonstrated through convergence theory that an extremely small learning rate is necessary when the eigenvalue of the Jacobian of the updated gradient does not meet certain favorable conditions [10]. Here is the classical convergence he used to delve into future proof. Define $F : \Omega \to \Omega$ be a continuously differential function, for example $F(x) = x - \eta \nabla G(x)$ (A typical gradient descent with G as the lost function). Then, if there exists a $\bar{x}$ satisfying following:

1. $F(\bar{x}) = \bar{x}$, and
2. the absolute values of the eigenvalues of the Jacobian $F'(\bar{x})$ are all smaller than 1.

Then there will have a neighborhood region that U such that any $x \in U$ converge to $\bar{x}$.

Therefore, the eigenvalues of the Jacobian influence the converge result. Under some unfavorable conditions, including eigenvalues of updated gradient having large complex values and zero-real parts, extremely small learning rate η is needed to project the eigenvalues into the required unit ball. Under simple GAN setting, the problem tends to converge if[3]

$$\eta < \frac{1}{|\Re(\lambda)|} \frac{2}{1 + \left(\frac{\Im(\lambda)}{\Re(\lambda)}\right)^2}$$

[3]Although the definition $F(x) = x - \eta \nabla G(x)$ here is under a different setting compared to Mescheder's one, the negative direction and eigenvalues with positive real part will reach the same result as Mescheder's one.

It clearly shows that if $|\Im(\lambda)| >> |\Re(\lambda)|$ then $\frac{\Im(\lambda)}{\Re(\lambda)}$ would be extremely small. Detailed proposition and proof is in Appendix A. Therefore, Mescheder introduces a regularization term $L(w, \theta)$ to tackle this issue [10].

Algorithm 1: Consensus Algorithm

$\gamma \leftarrow$ regularization parameter
$\eta \leftarrow$ low learning rate
G , $D \leftarrow$ lost function for generator, reward function for discriminator
while not converged **do**
  $L(w, \theta) = \frac{1}{2}|| \begin{pmatrix} \nabla_w G(w, \theta) \\ \nabla_\theta D(w, \theta) \end{pmatrix} ||^2$
  $w_{t+1} \leftarrow w_t - \eta * \nabla_w G(w_t, \theta_t) + \eta * \gamma * \nabla_w L(w_t, \theta_t)$
  $\theta_{t+1} \leftarrow \theta_t + \eta * \nabla_\theta D(w_t, \theta_t) - \eta * \gamma * \nabla_\theta L(w_t, \theta_t)$

Referring to Algorithm 1, this regularization term will push the eigenvalues to native real part. Therefore, it could maximize the maximum η value. The regularization parameter will be choose in [0.1, 10].

3.3 Wasserstein GAN

Wasserstein GAN (WGAN) [2] is a variation of GAN that makes use of the Wasserstein distance, also called earth mover's distance. The Wasserstein distance represents the minimum cost of transporting "earth" or mass, to transform one data distribution into another. The Wasserstein distance of the real data distribution $\mathbb{P}_r$ and generated data distribution $\mathbb{P}_g$ can be represented as the lower bound (infimum) of any transportation plan:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right]$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ represents the set of all joint distributions $\gamma(x, y)$ with marginals $\mathbb{P}_r$ and $\mathbb{P}_g$.

The divergence used in the original GAN is called Jensen-Shanon (JS) divergence, which may not converge under some cases such as parallel lines [2]. Wasserstein distance however, do converge in the suggested cases. WGAN employs the Wasserstein distance to provide a continuous measure of the distance between the real and generated distributions, which facilitates smoother gradients and training [2]. The critic replaces the discriminator to provide an estimation of the Wasserstein distance between the real and generated samples. The loss function in Section 2.1 can be redefined as follows:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[C(x)] - \mathbb{E}_{z\sim p_{\mathbf{z}}(z)}[C(G(z))]$$

The authors also mention that the calculation of Wasserstein distance requires Lipschitz continuity [2], which ensures that small changes in input leads to controlled changes in the output, leading to stability to avoid exploding or vanishing gradients. They suggest weight clipping, which restrict the weights within a predefined range, to enforce this constraint, although admitting that it is a terrible way, where a parameter too large increases the difficulty to reach optimality, and a parameter too small can lead to vanishing gradient, but is still employed due to its simplicity and low cost [2].

3.4 Optimistic Optimization

Daskalakis et al. propose an alternative optimization method for solving GANs, or more specifically WGANs [11]. The proposed method, called Optimistic Mirror Descent (OMD), is a variant of gradient descent. By Section 3.1.1, we know that gradient descent is one of the method for solving the min-max optimization problem to train the GAN. To delve into the context of online learning, this descent algorithm can be adapted into a broader framework known as the no-regret learning algorithm and the gradient descent would be a special case for Follow-the-Regularized-Leader (FTRL) with a ℓ_2 regularizer. A detailed proof for the correlations between gradient descent and no-regret learning is in Appendix B.

Algorithm 2: Optimistic Algorithm

$\eta \leftarrow$ low learning rate
G , $D \leftarrow$ lost function for generator, reward function for critic
while not converged **do**
  $w_{t+1} \leftarrow w_t - (2\eta * \nabla_w G(w_t, \theta_t) - \eta * \nabla_w G(w_{t-1}, \theta_{t-1}))$
  $\theta_{t+1} \leftarrow w_t + 2\eta * \nabla_\theta D(w_t, \theta_t) - \eta * \nabla_\theta D((w_{t-1}, \theta_{t-1}))$

To understand the intuition of OMD, we can start from FTRL perspective. Since gradient descent is equivalent with the conjunction of FTRL and ℓ_2 regularizer,

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \left[\left(\sum_{i=1}^{t} \nabla_w^T w \right) + \frac{1}{2\eta} ||w||_2^2 \right]$$

It is noted that if the summation here include the gradient of next iteration, then the optimization here will lead to a constant term that arises only from the regularization. Therefore, it is done by adding a item called predictor M_{t+1}

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \left[\left(\sum_{i=1}^{t} \nabla_w^T w \right) + M_{t+1}^T w + \frac{1}{2\eta} ||w||_2^2 \right]$$

There are group of choices for choosing the predictor suggested by Daskalakis et al. and we choose to use the simple predictor, past gradient to test the ability of this algorithm.

4 Experiment

4.1 Setting

All experiments were conducted using Python 3.9 with TensorFlow 2.6 and CUDA 11.2 on a machine equipped with an NVIDIA GeForce GTX 1650. The models were implemented based on the Deep Convolution GAN (DCGAN) architecture, with generator and discriminator using both 4 convolution layers, while the datasets used are MNIST [3] and CelabA [4]. For DCGAN, we further experiment two set of models structure (with Batch normalization and without Batch normalization) to see whether these optimization algorithm could still perform in a poor condition. We repeat the experiment five times to ensure there is a fair evaluation[4] .

4.2 Results

We did three major sets of comparison between the optimization algorithms, including Vanilla GAN Optimization *versus* Wasserstein GAN Optimization,Vanilla GAN Optimization *versus* Consensus Optimization,Wasserstein GAN Optimization *versus* Optimistic Optimization. Both Vanilla GAN and Consensus Optimization will use the lost function stated in Section 2.1. While, Wasserstein GAN and Optimistic Optimization will use the the lost function stated in Section 3.3.

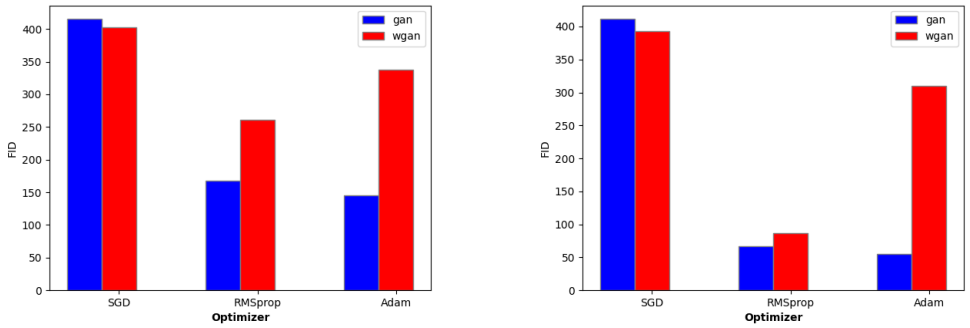
4.2.1 Vanilla vs Wasserstein

In comparison between Vanilla and Wasserstein in DCGAN with Batch Normalization (BN), we has done in total 12 types of experiment, including three types of optimizer explained in Section 3.1 and two learning rate chosen (1e-4, 1e-5)[5].

A complete sheet of generated samples is provided in Appendix C. It is important to emphasize that the samples were not selectively chosen or cherry-picked. The FID score evaluates test performance, with lower scores indicating better results. Based on Figure 1 (and Appendix C.1, Appendix C.2), both GAN and WGAN can be trained to generate images using RMSProp across various learning

[4]Github Link:`https://github.com/Mysterchan/estr3114_project/tree/main`

[5]Both Vanilla and Wasserstein, we have tired all three optimizer under two learning rate. So, In total $2 \times 3 \times 2 = 12$ types of experiment is conducted

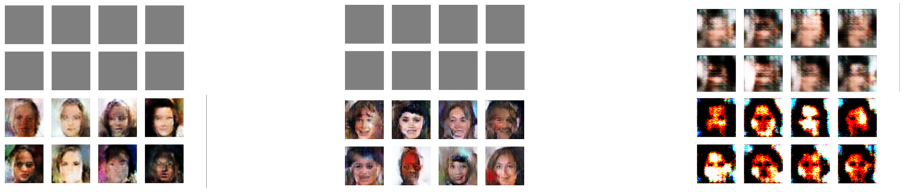


(a) Average FID Score under 1e-4 learning rate (b) Average FID Score under 1e-5 learning rate

Figure 1: The Comparison between Optimizer, Model and Learning rate on CelebA

rates. However, both models fail to generate plausible results when trained with SGD. While GANs can be successfully trained using Adam, WGANs are unable to replicate this result. This observation aligns with findings reported by Arjovsky et al. [2] regarding challenges in training WGANs.

In the experiments conducted without batch normalization (BN), we set the learning rate to 1e-5. As shown in Figure 2(a), we observed that WGANs are capable of generating images using RMSProp in the absence of batch normalization.



(a) GAN (Upper 2) and WGAN (Lower 2) when without BN (b) GAN (Upper 2) and Consensus (Lower 2) when without BN (c) WGAN (Upper 2) and Optimistic (Lower 2) under Adam

Figure 2: The Comparison of different algorithm

4.2.2 Vanilla vs Consensus

In the "no batch normalization" setting with a learning rate of 1e-5, we compared the performance of vanilla GAN and consensus GAN under three different optimizers: SGD, Adam, and RMSProp. Both models failed to generate meaningful images when trained with SGD and Adam. However, consensus optimization successfully enabled image generation when using RMSProp.

4.2.3 Wasserstein vs Optimistic

We followed the learning rate of 1e-4 as suggested by Daskalakis et al. [11], with batch normalization enabled. As stated, WGANs produced noisy and blurred images, while we did not replicate the exact results in optimistic GAN reported in the original paper (likely due to the difference in model size, which could not be resolved due to memory limitations), our findings nonetheless suggest that optimistic GANs are capable of generating higher-quality images compared to WGANs.

5 Conclusion

In this project, we explore several optimization algorithms like Consensus and Optimistic for GAN, and gain a broader understanding of how different variants of gradient descent, such as SGD, RMSProp and Adam, contribute to solving the min-max optimization problem in GANs. Through the completion of the experiments, we also discover how different algorithms perform under various scenarios, providing insights regarding the optimization of GANs.

References

[1] D. Saxena and J. Cao, "Generative adversarial networks (gans) challenges, solutions, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–42, 2021.

[2] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[3] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[4] Z. Liu, P. Luo, X. Wang, and X. Tang, "Large-scale celebfaces attributes (celeba) dataset," *Retrieved August*, vol. 15, no. 2018, p. 11, 2018.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[6] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *Advances in neural information processing systems*, vol. 27, 2014.

[7] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[8] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[9] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] L. Mescheder, S. Nowozin, and A. Geiger, "The numerics of gans," *Advances in neural information processing systems*, vol. 30, 2017.

[11] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng, "Training gans with optimism," *arXiv preprint arXiv:1711.00141*, 2017.

A Propositions and proof related to Consensus Optimization

Noted that the proof here is for gradient descent while [10] gives for gradient ascent only.

The value of the Jacobian $F'(x)$ would be equal to $I - \eta A$ where A is the jacobian of the gradient.

Lemma 1. *If λ is one of the eigenvalues of the A, $1 - \eta\lambda$ would be the a eigenvalue of the Jacobian $F'(x)$*

Proof. Suppose $A\vec{v} = \lambda\vec{v}$ where $\vec{v}$ is one of the corresponding eigenvectors. Then we have,

$$F'(x)\ \vec{v} = (I - \eta A)\vec{v} = \vec{v} - \eta A\vec{v} = \vec{v} - \eta\lambda\vec{v} = (1 - \eta\lambda)\vec{v}$$

Therefore, $(1 - \eta\lambda)$ would be the eigenvalue for matrix $F'(x)$. □

Lemma 2. *Assume that $A \in \mathbb{R}^{n\times n}$ only has eigenvalues with positive real-part and let $\eta > 0$. Then the eigenvalues of Jacobian $F'(x) = I - \eta A$ lie in the unit ball if and only if*

Proof. For $\lambda = a + b\,i$ with $a > 0$ we have eigenvalue $(1 - \eta\lambda)$ by lemma 1.

$$|1 - \eta\,\lambda|^2 = (1 - \eta\,a)^2 + \eta^2\,b^2 = 1 + \eta^2\,b^2 + \eta^2\,a^2 - 2\,\eta\,a.$$

Then, we have,

$$\eta\,(\eta\,b^2 + \eta\,a^2 - 2\,a) < 0$$

Since $\eta > 0$, the absolute value of the eigenvalue is smaller than 1 if and only if

$$\eta < \frac{2\,a}{b^2 + a^2} = \frac{2a^{-1}}{(b/a)^2 + 1}$$

By putting back $a = \Re(\lambda)$ and $b = \Im(\lambda)$, we have

$$\eta < \frac{1}{|\Re(\lambda)|} \frac{2}{1 + \left(\frac{\Im(\lambda)}{\Re(\lambda)}\right)^2}$$

□

B No-regret learning

The section aims to show Follow-the-Regularized-Leader (FTRL) with a ℓ_2 regularizer is equivalent to gradient descent.

For the FTRL framework with ℓ_2 regularizer at iteration t, we have the following,

$$x_t = \underset{x}{\operatorname{argmin}} \left[\left(\sum_{i=1}^{t-1} \ell_i \right) + \frac{1}{2\eta} ||x||_2^2 \right] \quad \text{where } \ell_i \text{ is the lost function at time } i$$

Consider the first-order approximation of ℓ_i, we have $\ell_i \approx \ell_i(x_i) + \nabla \ell_i(x_i)^T x$. Then the cumulative loss term becomes

$$\sum_{i=1}^{t-1} \ell_i \approx \left(\sum_{i=1}^{t-1} \ell_i \left(x_i\right) \right) + \boldsymbol{g}_{t-1}^T x \quad \text{where } \boldsymbol{g}_{t-1} = \sum_{i=1}^{t-1} \nabla \ell_i(x_i)$$

Therefore, the reformed x_t would be $\arg\min_x [(\sum_{i=1}^{t-1} \ell_i(x_i)) + \boldsymbol{g}_{t-1}^T x + \frac{1}{2\eta}||x||_2^2]$. Because we hope to find the minimum point of this item, we would apply differentiation on it.

$$\nabla_x \left(\left(\sum_{i=1}^{t-1} \ell_i \left(x_i\right) \right) + \boldsymbol{g}_{t-1}^T x + \frac{1}{2\eta} ||x||_2^2 \right) = \boldsymbol{g}_{t-1} + \frac{1}{\eta} x = 0$$

$$x_t = -\eta \boldsymbol{g}_{t-1} = x_{t-1} - \eta \nabla \ell_{t-1}(x_{t-1})$$

Hence, we get the gradient descent from FTRL with ℓ_2 regularizer.

C Sheets of samples

C.1 CelebA

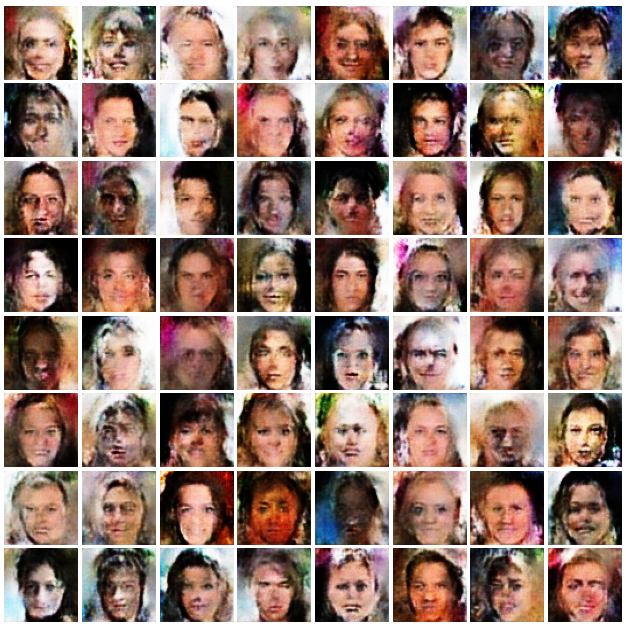


Figure 3: Samples from GAN using Adam Optimizer and Under 1e-5 rate in CelabA

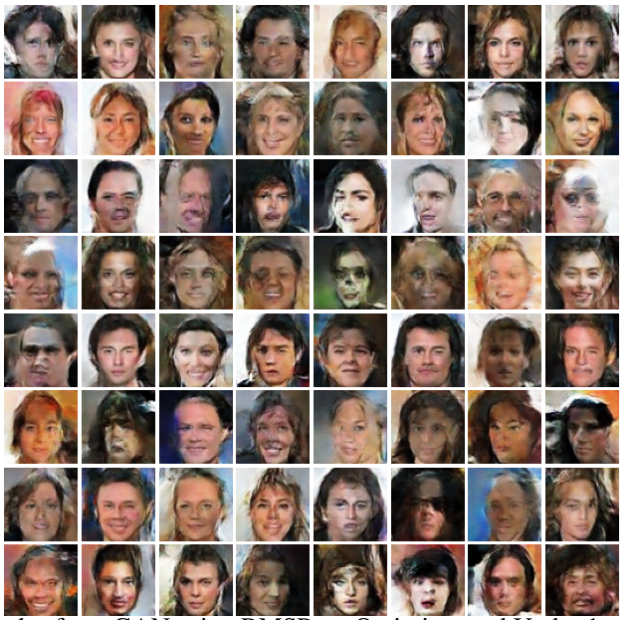


Figure 4: Samples from GAN using RMSProp Optimizer and Under 1e-5 rate in CelabA

Figure 5: Samples from WGAN using Adam Optimizer and Under 1e-5 rate in CelabA

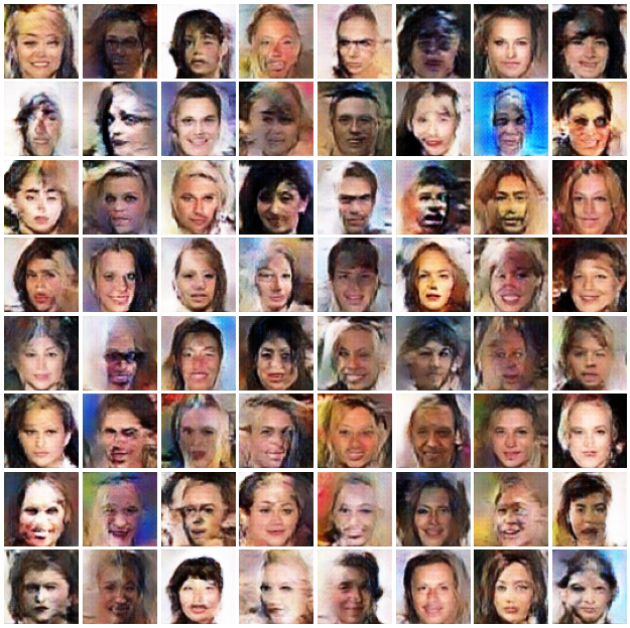


Figure 6: Samples from WGAN using RMSProp Optimizer and Under 1e-5 rate in CelabA

C.2 MNIST

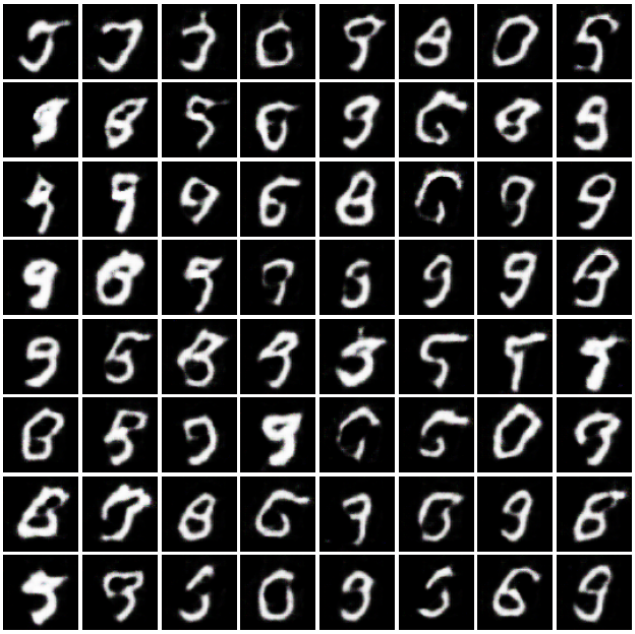


Figure 7: Samples from GAN using Adam Optimizer and Under 1e-5 rate in MNIST

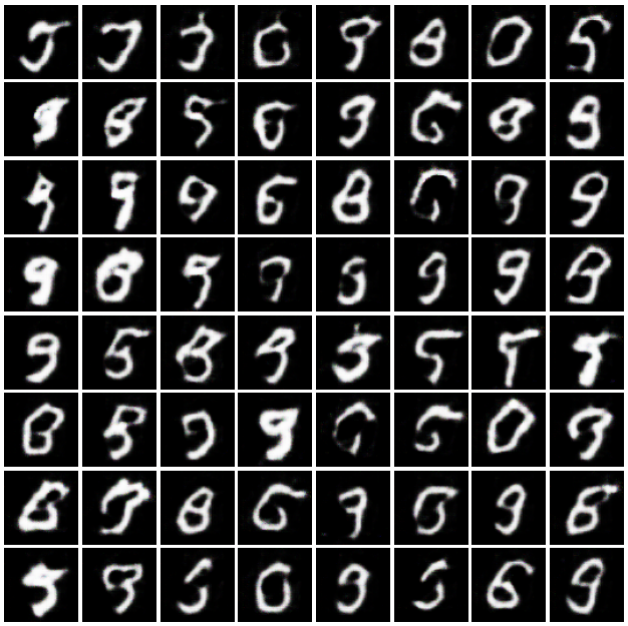


Figure 8: Samples from GAN using RMSProp Optimizer and Under 1e-5 rate in MNIST

Figure 9: Samples from WGAN using Adam Optimizer and Under 1e-5 rate in MNIST

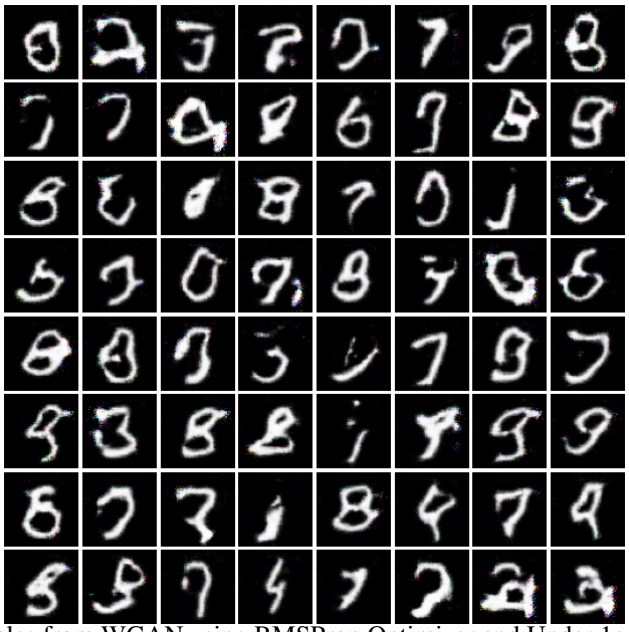


Figure 10: Samples from WGAN using RMSProp Optimizer and Under 1e-5 rate in MNIST