Optimizing Generative Adversarial Networks: The Impact of Landscape Modification on GAN Performance

By
Taiga Yamamoto

Supervisor:
Professor Michael CHOI

ST4288 Honours Project in Statistics
Department of Statistics and Data Science
National University of Singapore
2023/2024

Summary

Generative Adversarial Network (GAN) is one of the popular machine learning methods, particularly in generative models. The work proposes Landscape Modification (LM) in the context of GAN to understand if it can improve their performance. LM is a technique that modifies the loss landscape strategically so that algorithms can converge quickly, which has a high potential for the improvement of GAN training. This paper focuses on applying Landscape Modification to two GANs, Maximum Mean Discrepancy (MMD) GAN and Wasserstein GAN (WGAN), to investigate its behavior in these frameworks and how it accelerates learning.

This paper employs several optimizers to understand how the difference of each optimizer affects the behavior of LM. This study also examines the effects of LM varying the implementation methods of LM and combining it with different established modification methods, such as clipping for MMD GAN and gradient penalty for WGAN, to deepen the understanding of LM's behavior.

In conclusion, the thesis suggests that GAN with LM outperforms GAN without any techniques, unveiling its potential in the advancement of machine learning. However, its effectiveness notably depends on the optimizer's choice and where and how to apply LM to GANs. MMD GAN with LM outperforms the other established techniques on certain occasions, particularly when other modification techniques might not yield significant advancements. However, the effectiveness of LM varies across different experimental setups, and LM has many directions for the application of machine learning, necessitating further research to fully harness its potential in enhancing GAN performance.

Contents

1 Introduction

Generative Adversarial Networks (GANs) have been evolving rapidly since it was first introduced by Goodfellow et al. [1]. Various types of GANs have been developed to accelerate learning and achieve better generated images. Among them, this paper works with two types of GANs: Maximum Mean Discrepancy (MMD) GAN [2] and Wasserstein GAN (WGAN) [3]. MMD GAN replaces the objective function of GAN [1] with the Maximum Mean Discrepancy, and WGAN replaces it with Wasserstein distance, respectively.

This paper proposes the application of Landscape Modification (LM) [4] to strengthen the learning capability of the GANs. LM is a technique that strategically modifies the loss landscape, potentially improving machine learning performance and achieving stable learning and better convergence [5]. With the implementation of LM into MMD GAN and WGAN, this work aims to understand the behavior of LM in both GANs and see if it accelerates learning, potentially opening up new avenues for improved GAN learning, varying the parameters and its strategies for implementation in GANs.

Moreover, this paper examines the combinations of different modification techniques with LM. There are some established modification techniques that optimize the learning algorithms, such as MMD GAN with clipping [2] and WGAN with gradient penalty [3]. This study investigates the synergy of LM with these techniques to understand if the integration of modification techniques synergizes in order to understand the impact of LM on enhancing the performance of the GANs.

2 Landscape Modification

2.1 Landscape Modification

Landscape modification (LM) is a trick that modifies loss landscape in order to reach convergence faster. This technique was introduced in the context of simulated annealing, which was motivated by the process of annealing in metallurgy. The simulated annealing and stochastic gradient descent are similar [5], so LM can be applied to machine learning optimization. LM was introduced to transform the loss landscape so that algorithms converge to better global minima. The formula for gradient modification is as follows [5]:

$$\nabla H(x) = \frac{1}{f((U(x)-c)_+)+\varepsilon}\nabla U(x) \tag{1}$$

where $U(x)$ is a loss function, c is a running minimum, ε is a positive small constant, and $f : \mathbb{R}^d \leftarrow \mathbb{R}^+$ is a positive, twice differentiable and bounded function. When the current loss is smaller than c, the function f returns zero, and then, the gradient is divided by ε, which the default is 1. Therefore, the gradient is unchanged. On the other hand, when the current loss is larger than the running minimum, the gradient is divided by ε + $f(u(x)-c)$. The f returns a positive value; therefore, the gradient is divided by a value larger than 1. Then, the LM technique flattens the loss landscape.

This is the case of minimization. In GAN, the discriminator trains to maximize the loss. I suggest the landscape modification method tailored to the maximization problem.

$$\nabla H(x) = \frac{1}{f((c-U(x))_+)+\varepsilon}\nabla U(x) \tag{2}$$

where c is a running maximum. In the maximization problem, the landscape modification equation is applied so that when the current loss is lower than the running maximum of loss, the equation flattens the loss landscape so that algorithms can reach a better maximum effectively.

2.2 Illustration with $U(x)$

This section illustrates how Landscape Modification methods (Equation 1 and Equation 2) behave with the pre-determined constant c. I use the function $U(x)$ below in [5].

$$U(x) = sin(x) + sin(\frac{10x}{3}) \tag{3}$$

A minimization problem applies Equation 1 to $U(x)$ to modify the landscape. Here, I set $\varepsilon = -0.5$ and $f(x) = x$.

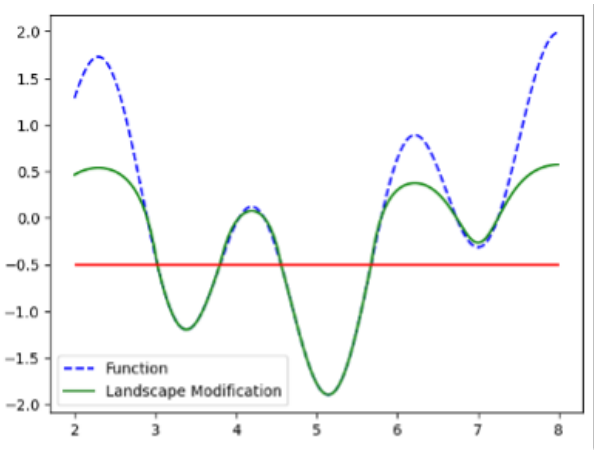


Figure 1: The Change of the Landscape with LM for Minimization

In the minimization problem, the loss landscape changes so that the modification makes it easier to ascend the hill by flattening the landscape below c while keeping the landscape below c as it is. The larger the loss function is, the larger the denominator, which makes the loss landscape vary for faster optimization, as in Figure 1. This modification helps the algorithm to converge to the minimum more quickly since the height of the hill the algorithm needs to climb is lowered.

A maximization problem applies Equation 2 to $U(x)$, modifying the loss landscape in the opposite manner. I set $c = 0.5$, and other parameters are the same as the minimization problem.

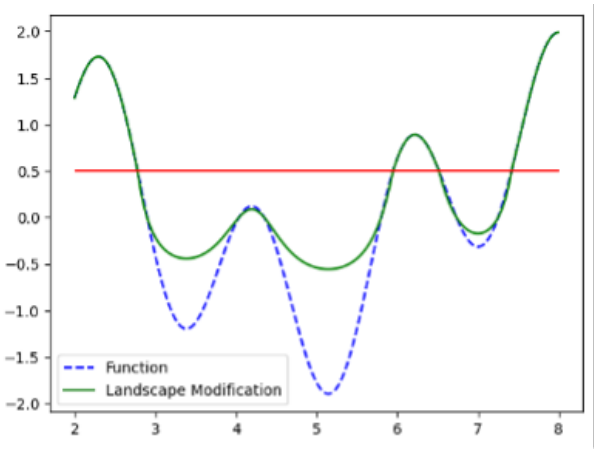


Figure 2: The Change of the Landscape with LM for Maximization

For the maximization problem, LM modifies the landscape so that algorithms can reach a better maximum more easily. When the loss is smaller than c, the gradient is divided by a number larger than 1, flattening the landscape below c. Like the minimization problem, this accelerates the optimization step by flattening the loss function significantly below the global minimum.

3 Optimization Algorithms

This chapter introduces optimization algorithms utilized in the experiments in this paper and discusses the application of LM to optimization algorithms. Specifically, I use the three optimizers, RMSprop, Adam, and Adam, in our experimental framework.

3.1 RMSprop

RMSprop [6] is an optimizer that modifies the learning rate of parameters by incorporating the past gradients. It adjusts the learning rate by considering a moving average of the squared gradients and divides the learning rate by the sum of the square root of the average and a small constant to avoid division by 0.

This modification effectively reduces the learning rate when the gradient is large and increases it when the gradient is small. With an auto-adaptive learning rate, the algorithm can work without tuning the learning rate and can avoid the exploding or vanishing gradient, leading to more balanced and efficient optimization.

For minimization and maximization purposes, LM modifies RMSprop as follows:

Algorithm 1 RMSprop LM-Minimization

Require: α: Stepsize, β: Decay rate within $[0,1)$, $h(\theta)$: Stochastic objective function with parameters θ, θ_0: Initial parameter vector, c: Running minimum, f: Function for Landscape Modification, ε_{LM}: A given constant

$r_0 \leftarrow 0$

$\theta \leftarrow \theta_0$

$t \leftarrow 0$

while θ not converged **do**

$t \leftarrow t+1$

$g_t \leftarrow \nabla_\theta h(\theta)$ (Get gradients w.r.t. stochastic objective at timestep t)

$r_t \leftarrow \beta \cdot r_{t-1} + (1-\beta) \cdot g_t^2$ (Update biased second moment estimate)

$LM \leftarrow \frac{1}{f(h(\theta)-c)_+ + \varepsilon_{LM}}$ (Calculate the Landscape Modification component)

$\theta \leftarrow \theta - \alpha \cdot LM \cdot \frac{g_t}{\sqrt{r_t}+\varepsilon}$ (Update parameters)

end while

return θ

Algorithm 2 RMSprop LM-Maximization

Require: α: Stepsize, β: Decay rate within $[0,1)$, $h(\theta)$: Stochastic objective function with parameters θ, θ_0: Initial parameter vector, c: Running maximum, f: Function for Landscape Modification, ε_{LM}: A given constant

$r_0 \leftarrow 0$
$\theta \leftarrow \theta_0$
$t \leftarrow 0$
while θ not converged **do**
$t \leftarrow t+1$
$g_t \leftarrow \nabla_\theta h(\theta)$ (Get gradients w.r.t. stochastic objective at timestep t)
$r_t \leftarrow \beta \cdot r_{t-1} + (1-\beta) \cdot g_t^2$ (Update biased second moment estimate)
$LM \leftarrow \frac{1}{f(c-h(\theta))_+ + \varepsilon_{LM}}$ (Calculate the Landscape Modification component)
$\theta \leftarrow \theta + \alpha \cdot LM \cdot \frac{g_t}{\sqrt{r_t}+\varepsilon}$ (Update parameters)
end while
return θ

3.2 Adam

Adam [7] is a popular optimization algorithm in deep learning. It combines momentum and RMSprop to improve learning efficiency. Momentum is a technique that reflects the previous learning direction to modify the learning rate. This technique can flatten the oscillation and adjust the step size for faster convergence by incorporating the value of the past gradients.

Moreover, incorporating the average of the past gradients reduces oscillations in the optimization path, leading to smoother convergence. Combined with RMSprop, which makes optimization robust against the sudden change of the gradient by adjusting the learning rate, Adam aims to achieve better optimization efficiency.

Similarly to RMSprop, I introduce the application of LM for minimization and maximization.

Algorithm 3 Adam LM -Minimization, based on [7]

Require: α: Stepsize, $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates, $h(\theta)$: Stochastic objective function with parameters θ, θ_0: Initial parameter vector, c: Running minimum, f: Function for Landscape Modification, ε_{LM}: A given constant

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t+1$

$g_t \leftarrow \nabla_\theta h_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)

$LM \leftarrow \frac{1}{f(h_t(\theta_{t-1})-c)_+ + \varepsilon_{LM}}$ (Calculate the Landscape Modification component)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot LM \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}$ (Update parameters)

end while

return θ_t (Resulting parameters)

Algorithm 4 Adam LM -Maximization

Require: α: Stepsize, $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates, $h(\theta)$: Stochastic objective function with parameters θ, θ_0: Initial parameter vector, c: Running maximum, f: Function for Landscape Modification, ε_{LM}: A given constant

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t+1$

$g_t \leftarrow \nabla_\theta h_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)

$LM \leftarrow \frac{1}{f(c-h_t(\theta_{t-1}))_+ + \varepsilon_{LM}}$ (Calculate the Landscape Modification component)

$\theta_t \leftarrow \theta_{t-1} + \alpha \cdot LM \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}$ (Update parameters)

end while

return θ_t (Resulting parameters)

3.3 NAdam

Nesterov-accelerated Adaptive Moment Estimation (NAdam) [8] is an optimization algorithm that further develops Adam by replacing the regular momentum adopted in Adam with Nesterov's accelerated gradient (NAG). According to [9], NAG is conceptually and empirically superior to regular momentum, which can be expressed as an improved momentum.

The NAdam optimization calculates the first and second moments in the same way as Adam. However, the first moment is then bias-corrected using the Nesterov momentum, considering the future direction of the gradient. This calculation, with the future direction of the gradient, provides the optimizer with more information about the update direction. This way of updating is more aligned with the future landscape, potentially leading to better convergence.

Similarly, I introduce the application of LM to NAdam for maximization and maximization.

Algorithm 5 NAdam LM -Minimization, based on [8]

Require: $\alpha_0, \ldots, \alpha_T; \mu_0, \ldots, \mu_T; \nu; \varepsilon$, $h(\theta)$: Stochastic objective function with parameters θ, c: Running minimum, f: Function for Landscape Modification, ε_{LM}: A given constant

$m_0 \leftarrow 0$ (first/second moment vectors)

$n_0 \leftarrow 0$

while θ_t not converged **do**

$g_t \leftarrow \nabla_{\theta_{t-1}} h_t(\theta_{t-1})$

$m_t \leftarrow \mu_t m_{t-1} + (1-\mu_t) g_t$

$n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$

$\hat{m}_t \leftarrow (\mu_{t+1} m_t / (1 - \prod_{i=1}^{t+1} \mu_i)) + ((1-\mu_t) g_t / (1 - \prod_{i=1}^{t} \mu_i))$

$\hat{n}_t \leftarrow \nu n_t / (1 - \nu^t)$

$LM \leftarrow \frac{1}{f(h_t(\theta_{t-1}) - c)_+ + \varepsilon_{LM}}$ (Calculate Landscape Modification component)

$\theta_t \leftarrow \theta_{t-1} - \alpha_t \cdot \hat{m}_t \cdot LM / (\sqrt{\hat{n}_t} + \varepsilon)$

end while

return θ_t

Algorithm 6 NAdam LM -Maximization

Require: $\alpha_0, \ldots, \alpha_T; \mu_0, \ldots, \mu_T; \nu; \varepsilon$, $h(\theta)$: Stochastic objective function with parameters θ, c: Running maximum, f: Function for Landscape Modification, ε_{LM}: A given constant

$m_0 \leftarrow 0$ (first/second moment vectors)

$n_0 \leftarrow 0$

while θ_t not converged **do**

$g_t \leftarrow \nabla_{\theta_{t-1}} h_t(\theta_{t-1})$

$m_t \leftarrow \mu_t m_{t-1} + (1-\mu_t) g_t$

$n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$

$\hat{m}_t \leftarrow (\mu_{t+1} m_t / (1 - \prod_{i=1}^{t+1} \mu_i)) + ((1-\mu_t) g_t / (1 - \prod_{i=1}^{t} \mu_i))$

$\hat{n}_t \leftarrow \nu n_t / (1 - \nu^t)$

$LM \leftarrow \frac{1}{f(c - h_t(\theta_{t-1}))_+ + \varepsilon_{LM}}$ (Calculate Landscape Modification component)

$\theta_t \leftarrow \theta_{t-1} + \alpha_t \cdot \hat{m}_t \cdot LM / (\sqrt{\hat{n}_t} + \varepsilon)$

end while

return θ_t

4 GAN

[1] introduced the Generative Adversarial Networks, which utilizes two models: A generator G and a discriminator D. The generator captures the features of training data x, and the discriminator projects the probability that a sample came from training data over the generator's output. The term 'adversarial' comes from the characters of GAN, which alternatively trains the generator and the discriminator. The generator trains to deceive the discriminator by learning the distribution of training data and generates samples from given random noise z. Then, the discriminator trains to correctly distinguish training data and the generated samples $G(z)$. GAN repeats this cycle to obtain the optimized generator.

The algorithm becomes the minimax game of the function $V(G,D)$ in terms of the discriminator and the generator [1]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{4}$$

As a result, the discriminator trains to maximize the function $V(G,D)$, and then the generator trains to minimize the second part of the function $V(G,D)$ so that the generator produces reasonable estimates of training data.

5 MMD GAN

MMD GAN is a derivation of GAN, whose loss function is replaced with Maximum Mean Discrepancy (MMD), a statistical measure to test the difference between two distributions. Applying LM to MMD GAN, this paper aims to examine the effectiveness of LM in the field of GAN.

5.1 Maximum Mean Discrepancy

MMD is an integral probability metric measuring the distance between two distributions. The metric provides a way to quantify how different the two distributions are. The definition of MMD is the following [10]:

$$\mathrm{MMD}[\mathbf{F}, p, q] := \sup_{f \in \mathbf{F}} (\mathbb{E}_{x \sim p}[f(x)] - \mathbb{E}_{y \sim q}[f(y)]) \tag{5}$$

where F is a class of functions $f : X \to \mathbb{R}$, x and y are random variables defined on a topological space $\mathbf{X}$, and each follows respective borel probability measures p and q ($x \sim p$ and $y \sim q$) [10].

I introduce the case in which the function F is in the unit ball in Reproducing Kernel Hilbert Space H. In RKHS, the probability distributions can be expressed in single points, the mean embeddings of the distributions. For example, the mean embedding of p: $\mu_p(t) = \langle \mu_p, k(t,\cdot) \rangle_H = \mathbb{E}_x k(t,x)$, where $f = \phi(t) = k(t,\cdot)$. $\phi(x)$ is a future mapping from $\mathbf{X}$ to $\mathbb{R}$ and $k(\cdot,\cdot)$ represents a kernel [10]. Therefore, the MMD between two distributions is then conceptualized as the difference between their mean embeddings in RKHS. The squared MMD can be obtained easily in RKHS.

$$\mathrm{MMD}^2[\mathbf{F}, p, q] = \|\mu_p - \mu_q\|^2_{\mathbf{H}} \tag{6}$$

When expressed in terms of a kernel function given random samples from x and y,

$$\mathrm{MMD}^2[\mathbf{F}, p, q] = \mathbb{E}_{x,x'}[k(x,x')] - 2\mathbb{E}_{x,y}[k(x,y)] + \mathbb{E}_{y,y'}[k(y,y')] \tag{7}$$

When a kernel is characteristic, then MMD becomes 0 if and only if the two distributions

are the same [2]. This makes the MMD possible to work as a loss for machine learning since we can guarantee that MMD gets close to 0 as the distributionz of real images and generated images approach.

One of the characteristic kernels is a Radial Basis Function (RBF) Kernel.

$$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \tag{8}$$

RBF Kernel calculates the similarity of the two points. The evaluation of the similarity can be affected by the sigma parameter. A suitable sigma parameter is needed for the optimal calculation of MMD, which requires tuning. In this paper, for convenience, algorithms apply a mixture of k RBF kernels, which are the sum of kernel functions with several different sigma parameters. As used in [2], we use five bandwidths, 1, 2, 4, 8, and 16.

$$K(x,y) = \sum_{q=1}^{k} \exp\left(-\frac{\|x-y\|^2}{2\sigma_q^2}\right) \tag{9}$$

5.2 MMD GAN

MMD GAN utilizes MMD as the loss function so that the difference between two distributions can be expressed in distance rather than probability. In this paper, MMD GAN adopts auto-encoder networks. The auto-encoder extracts features of data, and the decoder converts extracted features into images. The algorithm trains conducting the minimax problem with MMD utilizing the auto-encoder.

5.2.1 Auto-Encoder

An auto-encoder is an artificial neural network that extracts essential information by lowering the dimensions of the original data. Then, the decoder reconstructs the encoded data into the original dimensions based on the essence of the data extracted by the encoder.

There are some benefits of adopting auto-encoder [11]. The first advantage is that it lowers the dimensions so that the essential statistical information required to reconstruct the input data reliably is captured effectively. The transformation helps avoid dealing with complex,

high-dimensional data by making it a more manageable shape. Moreover, an auto-encoder reduces the amount of data needed to produce a reliable MMD estimate since the amount of data increases as the dimensions of the data increase.

The algorithms for the auto-encoder and decoder used in the experiments are below:

Algorithm 7 Encoder

Require: Input Shape - ($batchsize$, nc, $isize$, $isize$), $isize$ - input size, nc - number of channels in the input, k -output size, ndf - number of features in the first layer
Ensure: Conv2d(input channels, output channels, kernel size, stride, padding)
  Conv2d(nc, ndf, 4, 2, 1)
  LeakyReLU(0.2)
  $csize \leftarrow isize/2$
  $cndf \leftarrow ndf$
  while $csize > 4$ **do**
    Input feature $\leftarrow cndf$
    Output feature $\leftarrow cndf \times 2$
    Conv2d(Input feature, Output feature, 4, 2, 1)
    BatchNorm2d(Output feature)
    LeakyReLU(0.2)
    $cndf \leftarrow cndf \times 2$
    $csize \leftarrow csize/2$
  end while
  Conv2d($cndf$, k, 4, 1, 0)
  return Output $(batchnorm, nc, isize, isize)$

Algorithm 8 Decoder

Require: Input Shape - ($batchsize$, k, 1, 1), $isize$ - input size, nc - number of channels in the input, k -output size, ngf - number of features in the first layer
$cngf \leftarrow ngf//2$
$tsize \leftarrow 4$
while $tsize != isize$ **do**
$cngf \leftarrow cngf \times 2$
$tsize \leftarrow tsize \times 2$
end while
ConvTranspose2d(k, $cngf$, 4, 1, 0)
BatchNorm2d($cngf$)
ReLU
$csize = 4$
while $csize < isize//2$ **do**
Input feature $\leftarrow cngf$
Output feature $\leftarrow cngf//2$
ConvTranspose2d(Input feature, Output feature, 4, 2, 1)
BatchNorm2d(Output feature)
ReLU(Output feature)
$cngf \leftarrow cngf \times 2$
$csize \leftarrow csize/2$
end while
ConvTranspose2d($cngf$, nc, 4, 2, 1)
Tanh(nc)
return Output $(batchnorm, nc, isize, isize)$

5.2.2 Algorithms

The objective function of MMD GAN becomes similar to the GAN, which is a minimax formulation [2].

$$\min_{\theta} \max_{\phi} M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \sim g_\theta(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2 \tag{10}$$

where X is sampled from training data, Z is random noise, y is generated samples from Z, and λ is a predetermined constant to multiply the reconstruction loss,

In the algorithm, with the introduction of the auto-encoder, I denote the generator as g_θ and the discriminator as f_ϕ. During the training of the discriminator, the algorithm also trains to minimize the reconstruction loss. Therefore, the discriminator has to have both the encoder and the decoder, defining $\phi = \{\phi_e, \phi_d\}$, which are the encoder and decoder, respectively.

The left-hand side is MMD between the two distributions: training data and generated samples from the generator. The discriminator trains to maximize the MMD. Alternatively, the generator trains to minimize the MMD. The right-hand side is the formula expressing the auto-encoder reconstruction error, which calculates the expectation of squares of the difference between y and the reconstructed y, in which the auto-encoder of the discriminator extracts the features, and the decoder of the discriminator reconstructs the encoded y. During the training of the discriminator, the parameters of the discriminator also try to minimize the reconstruction error since the minimax function subtracts the reconstruction error from the MMD.

The algorithm of MMD GAN is as follows:

Algorithm 9 MMD GAN (Original), based on [2]

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update
Initialize generator parameters θ and discriminator parameters ϕ
while θ has not converged **do**
  for $t = 1$ to n_c **do**
    Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
    $f_\phi \leftarrow \nabla_\phi M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$
    $\phi \leftarrow \phi + \alpha \cdot \text{Optimizer}(\phi, f_\phi)$
  end for
  Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
  $g_\theta \leftarrow \nabla_\theta M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$
  $\theta \leftarrow \theta - \alpha \cdot \text{Optimizer}(\theta, g_\theta)$
end while

Moreover, [2] introduced weight clipping of the discriminator's parameters. The trick prevents the extreme gradient, which causes the explosion or vanishment of the gradient, and encourages stable training. The algorithms change to below:

Algorithm 10 MMD GAN Clippinig [2]

Require: α: the learning rate, c: the clipping parameter, B: the batch size, n_c: the number of iterations of discriminator per generator update
Initialize generator parameters θ and discriminator parameters ϕ
while θ has not converged **do**
for $t = 1$ to n_c **do**
Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
$f_\phi \leftarrow \nabla_\phi M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$
$\phi \leftarrow \phi + \alpha \cdot \text{Optimizer}(\phi, f_\phi)$
$\phi \leftarrow \text{clip}(\phi, -c, c)$
end for
Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
$g_\theta \leftarrow \nabla_\theta M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$
$\theta \leftarrow \theta - \alpha \cdot \text{Optimizer}(\theta, g_\theta)$
end while

5.3 MMD GAN with Landscape Modification

This paper introduces four ways to implement Landscape Modification in MMD GAN based on Algorithm 9 to understand how LM works for the MMG GAN. The first algorithm employs LM only to the discriminator's training, and the second applies only to the generator. The third and the fourth algorithms use LM for both the discriminator and the generator. However, the treatment of the running minimum and maximum differ.

5.3.1 MMD GAN with LM -D

The first algorithm applies LM only to the discriminator's training. LM modifies the gradient of the loss landscape to train the discriminator's parameters. Here, LM works based on the discriminator's running maximum applying Equation 2. The loss landscape of the discriminator constantly changes as the generator also improves to minimize the loss. Therefore, the algorithm resets the running maximum of the discriminator every time the generator conducts optimization. The algorithm trains the generator with default optimizers. The final algorithm is below:

Algorithm 11 MMD GAN LM -D

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update, c: the running maximum of loss
Initialize generator parameters θ and discriminator parameters ϕ
while θ has not converged **do**
  $c \leftarrow$ arbitrary small value
  for $t = 1$ to n_c **do**
    Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
    $loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$
    $c \leftarrow max(c, loss)$
    $f_\phi \leftarrow \nabla_\phi loss$
    $\phi \leftarrow \phi + \alpha \cdot OptimizerLM_{Maximization}(\phi, f_\phi, c)$
  end for
  Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
  $g_\theta \leftarrow \nabla_\theta M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$
  $\theta \leftarrow \theta - \alpha \cdot \text{Optimizer}(\theta, g_\theta)$
end while

5.3.2 MMD GAN with LM -G

In the same way, the next algorithm applies LM only for the generator, not the discriminator. Here, the discriminator uses the original optimizer, and the generator uses the optimizer with LM. The objective of the generator is to minimize the loss. Thus, with LM, the gradient of the loss landscape gets flattened when the current loss is higher than the running minimum applying Equation 1. Otherwise, the landscape does not get altered. The running minimum of the generator gets initialized every predetermined k iteration.

Algorithm 12 MMD GAN LM -G

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update, c: the running minimum of loss

Initialize generator parameters θ and discriminator parameters ϕ

while θ has not converged **do**

$c \leftarrow$ arbitrary huge value

for $t = 1$ to n_c **do**

Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$

$f_\phi \leftarrow \nabla_\phi \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$

$\phi \leftarrow \phi + \alpha \cdot \text{Optimizer}(\phi, f_\phi)$

end for

Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$

$loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$

$c \leftarrow min(c, loss)$

$g_\theta \leftarrow \nabla_\theta loss$

$\theta \leftarrow \theta - \alpha \cdot OptimizerLM_{Minimization}(\theta, g_\theta, c)$

end while

5.3.3 MMD GAN with LM -Both (No Reset)

The third algorithm applies LM for both the discriminator and the generator. Similarly to the other algorithms, the discriminator trains to maximize the loss modified according to the relationship with the running maximum (Equation 2), and the loss is minimized throughout the generator's training with the LM according to the running minimum (Equation 1). This algorithm does not reset the running minimum and maximum and uses the running extremum throughout the whole iteration. The algorithm becomes the following:

5.3.4 MMD GAN with LM -Both (Reset)

The fourth algorithm's structure is almost identical to the previous algorithm, but this algorithm resets the running extrema, considering that the running extrema becomes less worthwhile as the optimization process modifies the landscape. Each extremum is initialized with every i, j iteration predetermined for each optimizer.

Algorithm 13 MMD GAN LM -Both (Without reset of running extrema)

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update, c_{min}: the running minimum of loss, c_{max}: the running maximum of loss

Initialize generator parameters θ and discriminator parameters ϕ

while θ has not converged **do**

$c_{min} \leftarrow$ arbitrary huge value

$c_{max} \leftarrow$ arbitrary small value

for $t = 1$ to n_c **do**

Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$

$loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$

$c_{max} \leftarrow max(c_{max}, loss)$

$f_\phi \leftarrow \nabla_\phi loss$

$\phi \leftarrow \phi + \alpha \cdot OptimizerLM_{Maximization}(\phi, f_\phi, c_{max})$

end for

Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$

$loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$

$c_{min} \leftarrow min(c_{min}, loss)$

$g_\theta \leftarrow \nabla_\theta loss$

$\theta \leftarrow \theta - \alpha \cdot OptimizerLM_{Minimization}(\theta, g_\theta, c_{min})$

end while

Algorithm 14 MMD GAN LM -Both (With reset of running extrema)

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update, g_c: the interval of every c_{min} iterations, c_{min}: the running minimum of loss, c_{max}: the running maximum of loss

Initialize generator parameters θ and discriminator parameters ϕ

while θ has not converged **do**

$c_{min} \leftarrow$ arbitrary huge value

$c_{max} \leftarrow$ arbitrary small value

for $t = 1$ to n_c **do**

$c_{max} \leftarrow$ arbitrary small value (Reset the running maximum)

Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$

$loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$

$c_{max} \leftarrow max(c_{max}, loss)$

$f_\phi \leftarrow \nabla_\phi loss$

$\phi \leftarrow \phi + \alpha \cdot OptimizerLM_{Maximization}(\phi, f_\phi, c_{max})$

end for

if g_c mod generator iterations = 0 **then**

$c_{min} \leftarrow$ arbitrary huge value

end if

Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$

$loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$

$c_{min} \leftarrow min(c_{min}, loss)$

$g_\theta \leftarrow \nabla_\theta loss$

$\theta \leftarrow \theta - \alpha \cdot OptimizerLM_{Minimization}(\theta, g_\theta, c_{min})$

end while

5.4 MMD GAN with Landscape Modification and Clipping

Moreover, this essay examines how the modification techniques interact with each other. Here, I introduce two algorithms that employ both LM and Clipping for the discriminator's training. There are two ways to implement them. The first one implements LM first, then clips the parameters, and vice versa.

The difference between the two algorithms is the timing of clipping. Algorithms 15 clip the parameters after optimization. Therefore, the parameters are kept in the clipping range. On the other hand, algorithm 16 clamps the parameter before the optimization process, so the parameters can be out of the clipping range after the last training of the discriminator.

Algorithm 15 MMD GAN LM/CP

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update, c_{max}: the running maximum of loss, c: the clipping parameter
Initialize generator parameters θ and discriminator parameters ϕ
while θ has not converged **do**
  $c_{max} \leftarrow$ arbitrary small value
  for $t = 1$ to n_c **do**
    Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
    $loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$
    $c_{max} \leftarrow max(c_{max}, loss)$
    $f_\phi \leftarrow \nabla_\phi loss$
    $\phi \leftarrow \phi + \alpha \cdot OptimizerLM_{Maximization}(\phi, f_\phi, c_{max})$
    $\phi \leftarrow \text{clip}(\phi, -c, c)$
  end for
  Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
  $g_\theta \leftarrow \nabla_\theta M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$
  $\theta \leftarrow \theta - \alpha \cdot \text{Optimizer}(\theta, g_\theta)$
end while

Algorithm 16 MMD GAN CP/LM

Require: α: the learning rate, B: the batch size, n_c: the number of iterations of discriminator per generator update, c_{man}: the running maximum of loss, c: the clipping parameter
Initialize generator parameters θ and discriminator parameters ϕ
while θ has not converged **do**
$c_{max} \leftarrow$ arbitrary small value
for $t = 1$ to n_c **do**
$\phi \leftarrow \text{clip}(\phi, -c, c)$
Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
$loss \leftarrow M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z))) - \lambda \mathbb{E}_{y \in X \cup g(Z)} \|y - f_{\phi_d}(f_{\phi_e}(y))\|^2$
$c_{max} \leftarrow max(c_{max}, loss)$
$f_\phi \leftarrow \nabla_\phi loss$
$\phi \leftarrow \phi + \alpha \cdot OptimizerLM_{Maximization}(\phi, f_\phi, c_{max})$
end for
Sample minibatches $\{x_i\}_{i=1}^{B} \sim \mathbb{P}(X)$ and $\{z_j\}_{j=1}^{B} \sim \mathbb{P}(Z)$
$g_\theta \leftarrow \nabla_\theta M_{f_{\phi_e}}(\mathbb{P}(X), \mathbb{P}(g_\theta(Z)))$
$\theta \leftarrow \theta - \alpha \cdot \text{Optimizer}(\theta, g_\theta)$
end while

6 WGAN

Wasserstein GAN (WGAN) is another deviation of GAN, which employs Wasserstein distance as a loss function. There are some variants of WGAN, such as WGAN with clipping. In this section, I introduce the improved WGAN with gradient penalty [3] and implement LM to the WGAN algorithm, examining how it affects learning efficiency.

6.1 Wasserstein Distance

Wasserstein distance is an integral probability metric similar to MMD, which measures the distance between two distributions. Wasserstein distance is also called the Earth-Mover distance since the distance measures the minimum cost of transportation from one distribution to another distribution [12]. The more similar the two distributions are, the smaller the amount needed to transport from one to another. Therefore, the Wasserstein distance becomes smaller. Wasserstein distance can be expressed as follows [3]:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{11}$$

where $\mathbb{P}_r$ is the real data distributions, $\mathbb{P}_g$ is the distribution of generated data, and $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is a set of all joint distributions $\gamma(x,y)$.

6.2 WGAN

WGAN replaces the loss function of GAN [1] with Wasserstein distance and conducts training minimizing the Wasserstein distance between real images and generated images. In WGAN, the critic, the discriminator in GAN and MMD GAN, must be in the space of 1-Lipschitz functions to achieve more stable learning. There are two ways to enforce this condition: weight clipping and gradient penalty [3]. This paper uses WGAN with gradient penalty for the experiments.

6.2.1 Structure of WGAN

The objective function of the WGAN becomes as follows:

$$\min_{G} \max_{D \in \mathbf{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] \tag{12}$$

where **D** is the set of 1-Lipschitz functions. While critic D tries to maximize the loss function, generator G trains to minimize the loss, and they repeat the cycle adversarially. In order to enforce the critic to be 1-Lipschitz function, [3] proposes loss with gradient penalty.

$$L = \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right] \tag{13}$$

where $\hat{x}$ is interpolated samples from both real and fake images. The gradient penalty part enforces the critic loss to be 1-Lipschitz by penalizing the norm of the gradient of the critic's output when the norm exceeds 1. Moreover, with the gradient penalty, the algorithm trains so that the norm of the gradient stays close to 1, leading to stable training.

6.2.2 Algorithms

Incorporating the gradient penalty with WGAN, the algorithm of WGAN-GP becomes as follows. Furthermore, I introduce WGAN-GP with the implementation of LM into the critic training. After taking the gradient penalty into the loss function, the optimization algorithm with LM modifies its gradient, and the algorithms display how the overlay of the two tricks works.

Algorithm 17 WGAN with Gradient Penalty (WGAN-GP) [13]

Require: gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
  Initialize w with w_0, θ with θ_0.
  while θ has not converged **do**
    for $t = 1$ to n_{critic} **do**
      for $i = 1$ to m **do**
        Sample real data $x \sim \mathbb{P}_r$, latent variable $z \sim p(z)$, random number $\varepsilon \sim U[0,1]$.
        $\tilde{x} \leftarrow G_\theta(z)$
        $\hat{x} \leftarrow \varepsilon x + (1-\varepsilon)\tilde{x}$
        $L^{(i)} \leftarrow D_w(\tilde{x}) - D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$
      end for
      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m}\sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
    end for
    Sample a batch of latent variables $z^{(i)} \sim p(z)$
    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^m -D_w(G_\theta(z^{(i)})), \theta, \alpha, \beta_1, \beta_2)$
  end while

Algorithm 18 WGAN-GP LM

Require: gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, batch size m, Adam hyperparameters α, β_1, β_2, the running maximum of the critic loss c, A positive constant ε_{LM}
Require: initial critic parameters w_0, initial generator parameters θ_0.
  Initialize w with w_0, θ with θ_0, $c \leftarrow$ arbitrary large value
  while θ has not converged **do**
    for $t = 1$ to n_{critic} **do**
      for $i = 1$ to m **do**
        Sample real data $x \sim \mathbb{P}_r$, latent variable $z \sim p(z)$, random number $\varepsilon \sim U[0,1]$.
        $\tilde{x} \leftarrow G_\theta(z)$
        $\hat{x} \leftarrow \varepsilon x + (1-\varepsilon)\tilde{x}$
        $L^{(i)} \leftarrow D_w(\tilde{x}) - D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$
      end for
      $L \leftarrow \frac{1}{m}\sum_{i=1}^m L^{(i)}$
      $c \leftarrow max(c, L)$
      $w \leftarrow AdamLM_{Maximization}(\nabla_w L, w, \alpha, \beta_1, \beta_2, c, \varepsilon_{LM})$
    end for
    Sample a batch of latent variables $z^{(i)} \sim p(z)$
    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^m -D_w(G_\theta(z^{(i)})), \theta, \alpha, \beta_1, \beta_2)$
  end while

7 Methodology

7.1 Experiment Settings

I use two datasets for the experiments: CIFAR-10 [14] and LSUN Bedroom [15]. MMD GAN and WGAN training utilize CIFAR-10 and only MMD GAN conducts experiments on LSUN Bedroom. CIFAR-10 comprises 50,000 training images, ranging from images of horses, deer, and dogs to ships, airplanes, and automobiles. LSUN Bedroom datasets comprise 3,000,000 images of bedrooms. For the training, I used NVIDIA Tesla T4 on Google Collaboration.

The generator iteration for the experiments is predetermined. For the MMD GAN with CIFAR-10, the iteration is 25,000. The MMD GAN with LSUN Bedroom is 30,000, and I set 15,000 as the generator iteration for the training of WGAN with CIFAR-10.

7.2 Evaluation Metrics

I use two methods to evaluate the generated samples: Inception Score and Fréchet Inception Score. The two evaluation scores are calculated based on 10,000 images generated from the generator after it finishes the training.

7.2.1 Inception Score

The inception score [16] evaluates the quality of generated images and their diversity using the Inception model, and the score returns results similar to the conception of humans. The higher the quality and diversity of the generated images, the higher the inception score becomes. The calculation of the inception score utilizes the Inception model, which is the pre-trained deep learning model specializing in image recognition. The model labels each generated image, and the higher the probability of one particular label, the higher the inception score. On the other hand, when the probability of the labels is not concentrated on one label but scattered, the inception score returns a lower score. Moreover, the inception model evaluates the diversity of the generated images with the distributions of the categories to which each generated image is assigned. The more categories of generated images the generator generates, the higher the inception score.

7.2.2 FID

The inception score evaluates generated images' quality and diversity but never compares them with real images. Conversely, Fréchet Inception Distance (FID) [17] measures the similarity between real and generated images. The more similar the two data are, the lower the FID becomes. The calculation of the FID utilizes the inception model, which extracts the features of real and generated images. The Fréchet Distance is calculated using the extracted features.

8 Experiment Results

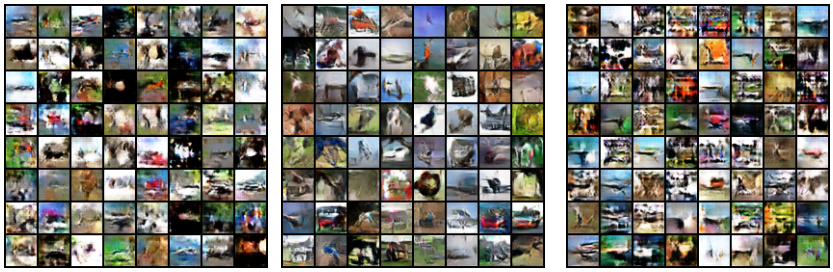
8.1 Experiments on MMD GAN

8.1.1 Evaluation of MMD GAN with Landscape Modification

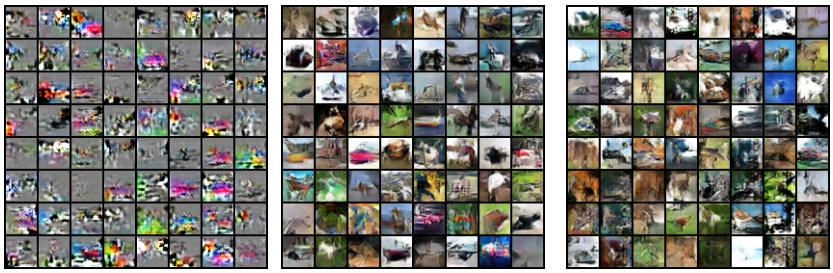
The experiments in this section are based on MMD GAN, varying the optimizers and adopted modification techniques. Three optimizers, RMSProp, Adam, and NAdam, are used for the optimizers to see how each optimizer behaves differently with LM. I run original MMD GAN (Algorithm 9), MMD GAN Clipping (Algorithm 10), and MMD GAN LM -D (Algorithm 11) in order to measure the effectiveness of LM. Since the weight clipping is used only for the auto-encoder of the discriminator, MMD GAN LM applies LM only for the discriminator for the direct comparison. The generated images and the evaluation scores are below:

Table 1: Performances of MMD GAN on Different Optimizers and Algorithms

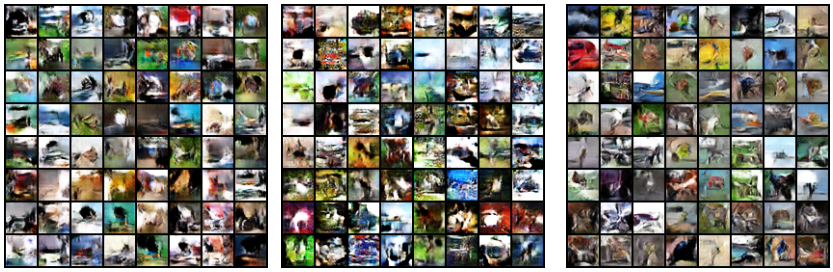
Optimizer	**Algorithm**	**FID**	**IS(mean $\pm$ sd)**
RMSprop	Original	102.554	3.424 ± 0.086
RMSprop	Clipping	**57.109**	$\mathbf{5.323 \pm 0.137}$
RMSprop	LM	88.301	4.067 ± 0.094
Adam	Original	118.968	2.966 ± 0.064
Adam	Clipping	**59.595**	$\mathbf{5.221 \pm 0.113}$
Adam	LM	88.046	3.892 ± 0.107
Nadam	Original	78.879	4.042 ± 0.134
Nadam	Clipping	88.457	4.028 ± 0.115
Nadam	LM	**67.783**	$\mathbf{4.788 \pm 0.084}$



(a) Original -RMSprop (b) Clipping -RMSprop (c) LM -RMSprop



(d) Original -Adam (e) Clipping -Adam (f) LM -Adam



(g) Original -NAdam (h) Clipping -NAdam (i) LM -NAdam

Figure 3: The Generated Images of MMD GAN for the Experiments on Different Optimizers and Algorithms

Observations:

The results are similar for RMSprop and Adam. MMD GAN Clipping worked best for both optimizers in FID and IS. The generated images with MMD GAN Clipping are much more detailed, and we can recognize the object's shape, leading to the lower FID and the higher IS. Moreover, MMD GAN LM works better than the original MMD GAN, which indicates that LM has a certain effect on improving MMD GAN training. Although the evaluation scores are not better than MMD GAN Clipping, both FID and IS have significantly better scores than the original MMD GAN. The generated images are slightly blurred, but the images are completely clearer than the original MMD GAN, especially for Adam, which possibly causes mode collapse.

The results of NAdam are different from those of the two optimizers. MMD GAN LM

significantly outperforms the original MMD GAN and MMD GAN Clipping. The FID and IS are the best among the three algorithms, and we can see the shape of objects in the images, while the generated images of the other two algorithms are blurred, making it hard to recognize the objects. Moreover, contrary to RMSprop and Adam, the MMD GAN Clipping does not work well for clipping, whose scores are worse than the original algorithm. Although IS are almost the same, FID is much lower for the original MMD GAN, indicating that generated images of the original MMD GAN are more similar to real ones.

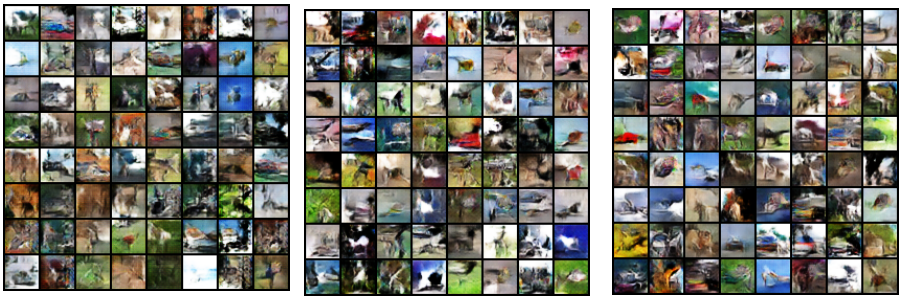
Interpretation:

The LM technique has a certain effect on accelerating the training in all three optimizers compared to the original MMD GAN. The better result can be attributed to the function of LM to flatten the loss landscape below the running maximum. LM reduces the cost of going down and up the valley again in the maximization problem of the discriminator, leading to stable learning and the prevention of being stuck in the local maximum.

Furthermore, there are differences in the behavior of LM between the optimizers. Particularly, MMD GAN LM works the best on the experiments on NAdam. The difference between NAdam and the other two optimizers is that NAdam incorporates the gradient of the next step, which is possibly the reason for the poor results of MMD GAN Clipping. There is a possibility that the moment in NAdam interferes with the parameter clipping, diminishing its optimization efficiency, which results in worse outputs than the original MMD GAN. Therefore, although the clipping works better than LM with RMSprop and Adam, LM might perform better when the optimizer does not harmonize with implemented algorithms, and LM can output the best result.

8.1.2 Modifying f

The next experiments are based on MMD GAN LM -D (Algorithm 11) using Adam, modifying the function f. Similarly to the previous section, only the discriminator adopts LM. The algorithm applies the three different f functions: $f(x) = x$, $f(x) = x^2$, and $f(x) = \sqrt{x}$. The function $f(x) = x^2$ flattens the loss landscape more substantially than the original $f(x) = x$ when x is larger than 1, while it flattens the loss landscape more moderately when x is smaller than 1, and vice versa for the function $f(x) = \sqrt{x}$. The generated images and the evaluation scores are below:



(a) $f(x) = x$ (b) $f(x) = x^2$ (c) $f(x) = \sqrt{x}$

Figure 4: The Generated Images of MMD GAN LMon Different *f(x)* with Adam

Table 2: Performances of MMD GAN LM on on Different *f(x)* with Adam

Optimizer	**f(x)**	**FID**	**IS (mean $\pm$ sd)**
Adam	x	88.046	3.892 ± 0.107
Adam	x^2	73.985	4.772 ± 0.107
Adam	$\sqrt{x}$	**67.002**	$\mathbf{4.823 \pm 0.125}$

Observation:

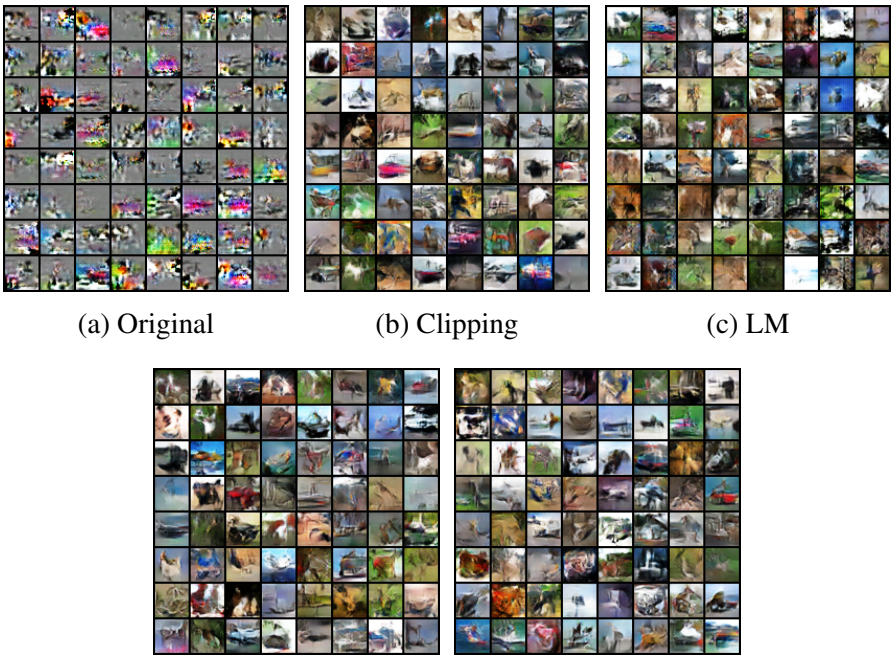
The generated images of MMD GAN with the function $f(x) = \sqrt{x}$ results in the best evaluation scores. The result of the function $f(x) = x^2$ follows closely to the function $f(x) = \sqrt{x}$. The FID score is slightly better for the function $f(x) = \sqrt{x}$ than $f(x) = x^2$ with almost the same IS scores. The default function $f(x) = x$ generated the worst images compared to the other f functions. The quality of the images of the function $f(x) = x$ is obviously lower than the other two functions, and both FID and IS are significantly lower.

Interpretation:

Intuitively, the results of the three functions follow the order of the order of the modification of power of f function. However, the experiments revealed that the results do not follow the order of the f function's strength to modify the landscape, indicating that the selection of the f function is sensitive. In the initial part of training, the function $f(x) = x^2$ flattens the landscape more compared to the function $f(x) = \sqrt{x}$ since the loss is larger than 1. However, the strength of modification changes when the loss is below 1 in the later optimization steps. It is important to ensure that the loss landscape gets flattened while preserving the gradient steep enough to conduct effective learning. Therefore, the optimal f function is distinctive for each experimental setting, and tuning the f function is needed to select the best f function.

8.1.3 Combination of Different Modification Techniques

There are various modification techniques to improve MMD GAN. [2] provides MMD GAN Clipping (Algorithm 10) and this paper introduces MMD GAN with LM. This section examines how the combinations of these two techniques behave. There are two ways to implement clipping and LM simultaneously. The one implements LM at first, then clips the discriminator's parameters (Algorithm 15). The second one implements the two techniques in the reversed order (Algorithm 16). I examine the effects of these combined implementations, comparing them to the original MMD GAN (Algorithm 9), MMD GAN clipping (Algorithm 10), and MMD GAN LM -D (Algorithm 11). The results are the following:



(a) Original (b) Clipping (c) LM

(d) LM then Clipping (e) Clipping then LM

Figure 5: The Generated Images of MMD GAN on Different Combinations of Techniques

Table 3: Performances of Different Combinations of Techniques on MMD GAN

Optimizer	**Algorithm**	**FID**	**IS (mean $\pm$ sd)**
Adam	Original	118.968	2.966 ± 0.064
Adam	Clipping	59.595	5.221 ± 0.113
Adam	LM	88.046	3.892 ± 0.107
Adam	LM/CP	**57.858**	$\mathbf{5.313 \pm 0.128}$
Adam	CP/LM	58.423	5.311 ± 0.135

Observations:

Although the difference is not significant, both MMD GAN LM/CP and CP/LM outperformed MMD GAN Clipping in both FID and IS. Especially for the IS, there might not be any improvement, considering the standard deviation. However, I can conclude that at least the algorithms with combinations do not worsen the training compared to the MMD GAN Clipping. The generated images of the three algorithms look alike, and there are no distinctively better images.

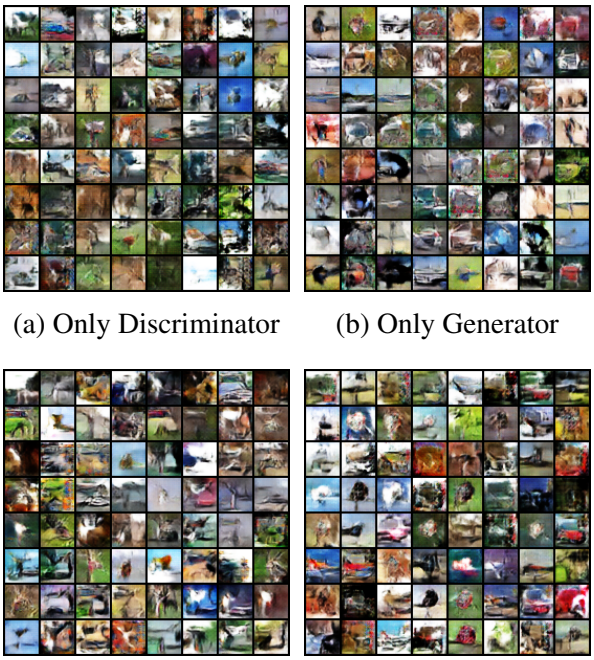
Interpretation:

The MMD GAN LM/CP conducts LM first and then clips the parameter. The algorithm applies LM before clipping, so the constraints by clipping hold all the time after the training of the discriminator. The optimized parameters with LM are diminished by clipping, but the clipping enforces the constraints, applying the 1-Lipshitz constraint. On the other hand, the MMD GAN CP/LM can be understood so that the parameters are constrained by clipping and optimized with LM under the already constrained condition. Therefore, there might be a possibility that the later algorithm results in weakened constraints by clipping since it applies LM after clipping.

The two changes by LM to MMD GAN Clipping do not result in a notable improvement, which suggests that double modifications do not sum up their effects simply, and the improvement with the additional technique is limited in this experiment setting.

8.1.4 Modifying Strategies to Implement Landscape Modification

The experiments in this section are based on the different strategies for implementing LM into MMD GAN. In the previous sections, the algorithms adopted LM in the discriminator, but this section compares four implementations of LM. The first algorithm is MMD GAN LM -D (Algorithm 11), which adopts LM only for the discriminator. The second algorithm, MMD GAN LM -G (Algorithm 12), utilizes LM only for the generator. Since the landscape changes every time the generator and discriminator train, the algorithms reset their running extrema every 5 iterations of the discriminator and the generator, respectively. The third and the fourth algorithms adopt LM for both the discriminator and the generator but with different strategies for resetting the running extrema. While the third algorithm, MMD GAN LM Both NoReset (Algorithm 13) does not reset the running extrema throughout the training, the fourth algorithm, MMD GAN LM Reset (Algorithm 14), resets the running extrema of both discriminator and generator every 5 iterations. The experiment results are below:



(a) Only Discriminator (b) Only Generator

(c) Both (No reset) (d) Both (Reset)

Figure 6: The Generated Images of MMD GAN on Different Implementations of LM

Table 4: Performances of MMD GAN on Different Implementations of LM

Optimizer	**LM**	**FID**	**IS (mean $\pm$ sd)**
Adam	Only D	88.046	3.892 ± 0.107
Adam	Only G	**66.029**	$\mathbf{5.099 \pm 0.124}$
Adam	No Reset	69.261	4.862 ± 0.138
Adam	Both	71.209	4.840 ± 0.123

Observations:

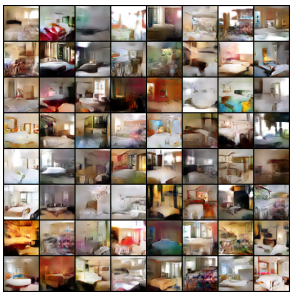
MMD GAN LM -G performs the best among the four algorithms with the best score in both FID and IS, followed by MMD GAN LM for both without reset and with reset. Although the evaluation score of MMD GAN LM -G is the best, the generated images are not distinctively better than the the two MMD GAN LM for both, which seem to generate images with fewer white parts. Especially, the images of MMD GAN LM for both with reset are detailed and have more recognizable objects even though the evaluation scores are worse than MMD GAN LM -G. MMD GAN LM -D underperforms remarkably. Both evaluation scores are far below the other algorithms, and the quality of the generated images is more blurred.

Interpretation:

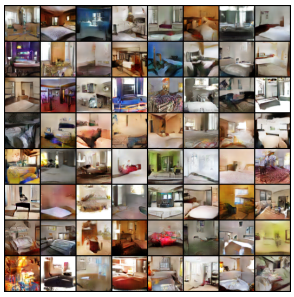
The implementation of LM into the generator might significantly improve the performance of MMD GAN. Although LM has a certain effect to help the discriminator maximize the MMD efficiently, the minimization of MMD by the generator is more crucial since it directly relates to the quality of generated images, considering that all three algorithms with LM for the generator outputs significantly better results than MMD GAN LM -D.

8.1.5 Behavior of Landscape Modification on LSUN

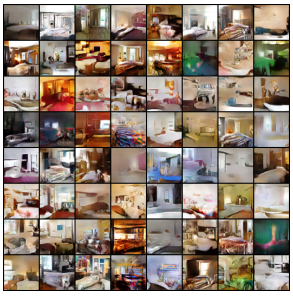
The previous experiments are all conducted on CIFAR-10. The section conducts training on the different datasets, LSUN Bedroom, and examines how LM behaves on the different datasets and if the behavior changes depending on the dataset's feature. I conducted training on four MMD GAN algorithms: original MMD GAN (Algorithm 9), MMD GAN Clipping (Algorithm 10), MMD GAN LM -D (Algorithm 11), and MMD GAN LM/CP (Algorithm 15), which has the best results on CIFAR-10. The generated images and their evaluation scores are below:



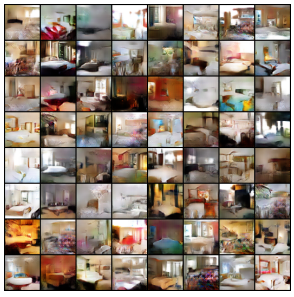
(a) Original MMD GAN



(b) MMD GAN Clipping



(c) MMD GAN LM



(d) MMD GAN LM/CP

Figure 7: The Generated Images of MMD GAN on LSUN Bedroom

Table 5: Performances of MMD GAN on LSUN Bedroom

Optimizer	**Algorithm**	**FID**	**IS (mean $\pm$ sd)**
Adam	Original	165.175	3.486 ± 0.064
Adam	Clipping	141.780	3.660 ± 0.135
Adam	LM	154.987	3.724 ± 0.061
Adam	LM/CP	**141.680**	$\mathbf{3.897 \pm 0.087}$

Observations:

The results are similar to the results of the experiment on CIFAR-10. The quality of MMD GAN LM images is much clearer and more detailed than the original MMD GAN, and both scores of MMD GAN LM are better than the original one. However, the MMD GAN Clipping outperforms the one with LM, the same as the results of CIFAR-10. Especially, FID is much lower, indicating that the generated images of MMD GAN Clipping are more similar to training data.

Moreover, the MMD GAN LM/CP outperforms the one with clipping. The FID is almost the same as the MMD GAN Clipping, which shows that the similarity of their generated images to training sets is virtually identical, but IS is much higher for MMD GAN LM/CP, indicating that images of them are more diverse and distinguishable.

Interpretation:

The results of CIFAR-10 and LSUN Bedroom do not contradict each other. MMD GAN LM/CP performs the best, followed by MMD GAN Clipping, LM, and then the original one. Thus, we can conclude that the MMD GAN with LM can work similarly for various datasets and performs better than the original one.

8.2 Experiments on WGAN

Moving on to WGAN, the experiments examine the combination of gradient penalty and LM on WGAN. There are two algorithms: WGAN-GP and WGAN-GP LM. WGAN-GP (Algorithm 16) applies gradient penalty to enforce the critic to be a 1-Lipschitz function to achieve stable learning. WGAN-GP LM (Algorithm 0) applies LM in the optimization process on top of the gradient penalty. The following images and tables are the results of the two algorithms:

(a) WGAN-GP (b) WGAN-GP LM

Figure 8: The Generated Images of WGAN-GP and WGAN-GP LM with Adam

Table 6: Performances of the Algorithms on on WGAN-GP and WGAN-GP LM

Optimizer	**f(x)**	**FID**	**IS (mean $\pm$ sd)**
Adam	WGAN-GP	**63.413**	5.635 ± 0.133
Adam	WGAN-GP LM	63.970	$\mathbf{5.741 \pm 0.198}$

Observations:

The FID is slightly lower for WGAN-GP, and the IS is higher for WGAN-GP LM, but there is no difference in IS considering the standard deviation. There is no notable improvement with the implemention of LM into WGAN-GP but it does not worsen the results. The generated images also do not have any particular differences, indicating that the implementation of LM does not positively impact WGAN-GP.

Interpretation:

The gradient penalty constrains the gradient, training the algorithm to keep its norm of the gradient close to 1, leading to stable learning. When the gradient penalty sufficiently constrains

the gradient, the effect of LM on stabilizing or accelerating learning might become limiting since the gradient is already modified, kept close to 1 by the gradient penalty. Thus, the impact of modifying the gradient again with LM is limited.

9 Conclusion

Generally, MMD GAN performs better with the implementation of Landscape Modification than the original MMD GAN without any modification techniques, and its effect seems consistent throughout the different datasets. The effectiveness of Landscape Modification differs depending on the algorithms used. In the experiments in this paper, MMD GAN Clipping works better than MMD GAN LM -D with RMSprop and Adam. However, MMD GAN Clipping works best with NAdam, indicating that this Landscape Modification method can enhance algorithms when the other modification method does not have a significant improvement effect.

Furthermore, the optimal f function possibly streamlines the learning process, although careful tuning is required. The behavior of Landscape Modification hugely depends on the f function, and the algorithm is sensitive to it. Matching the f function to the algorithms and the datasets can enhance learning results.

While the integration of techniques such as clipping in MMD GAN and gradient penalty in WGAN does not lead to significant synergy, it does slightly improve the outcome without worsening it. However, the effects of these techniques are diminished when one is applied over another, suggesting a need for further exploration of their combined use.

There is a massive possibility that Landscape Modification will improve GANs. Even though the experiments in this paper focus on implementing LM to the discriminator for comparison with other techniques, the results of 8.1.4 revealed that applying Landscape Modification to the generator significantly accelerates the learning. Landscape Modification in the generator can optimize the GANs, and further research can be done to understand its behavior in the generator. One possible way to understand the behavior of Landscape Modification more is to apply Landscape Modification to the generator and another technique to the discriminator.

Overall, the best ways to implement Landscape Modification are still known, and we need to understand further how Landscape Modification works in different settings. Still, the application of it can be applied to various machine learning methods for improvement.

Source Code: `https://github.com/TaigaYamamoto516/ST4288_Honours_Project_Taiga_Yamamoto`

10 References

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[2] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Póczos, "Mmd gan: Towards deeper understanding of moment matching network," 2017.

[3] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[4] M. C. H. Choi, "On the convergence of an improved and adaptive kinetic simulated annealing," 2020.

[5] I. Todea, "Landscape modification with machine learning optimization," 2022.

[6] G. Hinton and N. S. ad Kevin Swersky, "Neural networks for machine learning lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude," 2012.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[8] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[9] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: https://proceedings.mlr.press/v28/sutskever13.html

[10] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," 2012.

[11] Y. Li, K. Swersky, and R. Zemel, "Generative moment matching networks," 2015.

[12] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," 2017.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” 2017.

[14] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” 2009.

[15] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, “Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop,” 2016.

[16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” 2016.

[17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” 2018.

A Table of Hyperparameters Used in Experiments

Table 7: Hyperparameters Used in the Experiments in Section 8.1.1

Optimizer	Betas	Algorithm	D Learning Rate	G Learnig Rate	Clipping c	$f(x)$	Reset Max	Reset Min
RMSprop	-	Original	0.0001	0.0002	-	-	-	-
RMSprop	-	Clipping	0.0001	0.0002	0.01	-	-	-
RMSprop	-	LM (Discriminator)	0.0001	0.0002	-	x	5	-
Adam	(0.1, 0.99)	Original	0.0001	0.0002	-	-	-	-
Adam	(0.1, 0.99)	Clipping	0.0001	0.0002	0.01	-	-	-
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x	5	-
Nadam	(0.1, 0.99)	Original	0.0001	0.0002	-	-	-	-
Nadam	(0.1, 0.99)	Clipping	0.0001	0.0002	0.01	-	-	-
Nadam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x	5	-

Table 8: Hyperparameters Used in the Experiments in Section 8.1.2

Optimizer	Betas	Algorithm	D Learning Rate	G Learning Rate	Clipping c	$f(x)$	Reset Max	Reset Min
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x	5	-
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	$\sqrt{x}$	5	-
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x^2	5	-

Table 9: Hyperparameters Used in the Experiments in Section 8.1.3

Optimizer	Betas	Algorithm	D Learning Rate	G Learning Rate	Clipping c	$f(x)$	Reset Max	Reset Min
Adam	(0.1, 0.99)	Original	0.0001	0.0002	-	-	-	-
Adam	(0.1, 0.99)	Clipping	0.0001	0.0002	0.01	-	-	-
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x	5	-
Adam	(0.1, 0.99)	LM/CP	0.0001	0.0002	0.01	x	5	-
Adam	(0.1, 0.99)	CP/LM	0.0001	0.0002	0.01	x	5	-

Table 10: Hyperparameters Used in the Experiments in Section 8.1.4

Optimizer	Betas	Algorithm	D Learning Rate	G Learning Rate	Clipping c	$f(x)$	Reset Max	Reset Min
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x	5	-
Adam	(0.1, 0.99)	LM (Generator)	0.0001	0.0002	-	x	-	5
Adam	(0.1, 0.99)	LM (Both Non-resetting)	0.0001	0.0002	-	x	-	-
Adam	(0.1, 0.99)	LM (Both Resetting)	0.0001	0.0002	-	x	5	5

Table 11: Hyperparameters Used in the Experiments in Section 8.1.5

Optimizer	Betas	Algorithm	D Learning Rate	G Learning Rate	Clipping c	$f(x)$	Reset Max	Reset Min
Adam	(0.1, 0.99)	Original	0.0001	0.0002	-	-	-	-
Adam	(0.1, 0.99)	Clipping	0.0001	0.0002	0.01	-	-	-
Adam	(0.1, 0.99)	LM (Discriminator)	0.0001	0.0002	-	x	5	-
Adam	(0.1, 0.99)	LM/CP	0.0001	0.0002	0.01	x	5	-

Table 12: Hyperparameters Used in the Experiments in Section 8.2

Optimizer	Betas	Algorithm	D Learning Rate	G Learning Rate	$f(x)$	Reset Max	Reset Min
Adam	(0.5, 0.999)	WGAN-GP	0.0001	0.0001	-	-	-
Adam	(0.5, 0.999)	WGAN-GP LM	0.0001	0.0001	x	5	-