Robustness, Stability and Performance of Optimization Algorithms for GAN Training

Oskar Larsson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6136
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2021

Abstract

Training Generative Adversial Networks (GANs) for image synthesis problems is a largely heuristical process that is known to be fickle and difficult to set up reliably. To avoid common failure modes and succeed with GAN training, one needs to find very specific hyperparameter settings carefully tuned to the model architectures and datasets at hand. Standard GAN training optimization methods such as Stochastic Gradient Descent Ascent (SGDA) and Adam do not even converge on some very simple min-max problems, which may in part explain the unreliable results that occur when applying them in the more complicated GAN setting.

This thesis compares training GANs using SGDA and Adam with optimization schemes that do converge in convex-concave min-max settings. Specifically, Optimistic Gradient Descent Ascent (OGDA), the ExtraGradient method, and their Adam-variants are treated. Their robustness, stability and performance are evaluated and compared on the state-of-the-art GAN architectures U-Net GAN and StyleGAN 2.

The empirical results on U-Net GAN strongly indicate that the Adam-variants of OGDA and ExtraGradient are more robust to varying choices of hyperparameter settings and less prone to collapse mid-training than the most commonly used optimization schemes in contemporary GAN training, regular SGDA and Adam. However, for an already fine-tuned, well working Adam setup such as StyleGAN 2, end-results were neither improved nor worsened by using the Adam-variants of OGDA and ExtraGradient. This indicates that these algorithms are more reliable and easier to work with than SGDA and Adam, but not necessary for achieving state-of-the-art performance.

Acknowledgements

Thanks to Pontus Giselsson for supervising and guiding the work, and proposing the subject of the thesis.

Contents

1

Introduction

Generative Adversial Networks (GANs) were introduced in 2014 in [Goodfellow et al., 2014]. GANs represent the most successful framework to date for generative modeling using deep learning. The idea behind GANs is to simultaneously train two deep neural networks, a discriminator network and a generator network, where the discriminator is optimized to distinguish between fake data samples (generated by the generator) and real data samples (taken from a training set) and the generator is optimized for making the discriminator being as incorrect as possible. This process can be described as a zero-sum game between two players (the two networks) and the optimization process finds an approximate Nash equilibrium of the game.

The effectiveness of GANs has largely been illustrated through advances in image generation. Using a training set of a large number of images, GANs have been shown to successfully learn features of the dataset and create very convincing fake samples with no counterparts in the training set. The generator network is optimized to take a random noise vector from the so-called latent space as input and generate a convincing sample as output. As seen in Figure 1.1, the latent space has a lot of real-world meaning. By mixing different parts of random input vectors with each other, the output images become semantically combined versions of each other. This shows that different dimensions of the latent space correspond to different interpretable attributes such as colors and face attributes.

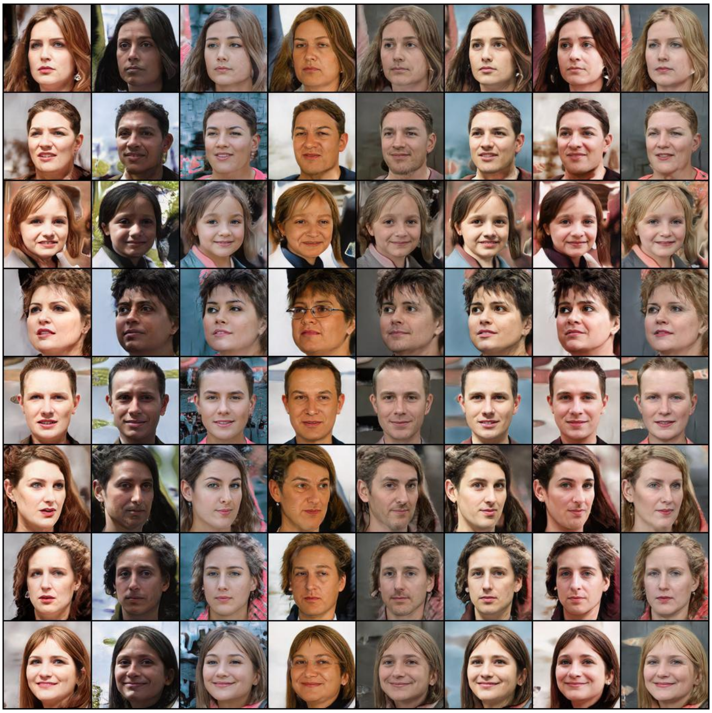


Figure 1.1: GAN generated images with architecture StyleGAN 2 [Karras et al., 2019], and latent vectors mixed. The diagonal contains images generated from different latent vectors and the rows are generated from those same latent vectors but with a small part replaced with a fixed latent vector that is the same throughout the column. Facial features remain mostly the same along rows and color features remain mostly the same along columns.

Since the inception of GANs in 2014, steady progress has been made in the state of the art of image generation. Starting with the DCGAN that was introduced in 2015 in [Radford et al., 2015], showing that deep convolutional networks, that had already been shown to be very effective on image classification tasks, also could be effective at image generation tasks through the GAN training framework. A lot of best practices were developed to train GANs and one big stride was automating image quality judgement through the invention of Inception Score in the paper *Improved Techniques for Training GANs* [Salimans et al., 2016]. The resolution of generated images was scaled up with progressive growing [Karras et al., 2017] and BigGAN [Brock et al., 2018]. StyleGAN [Karras et al., 2018] and StyleGAN 2 [Karras et al., 2019] introduced an architecture and regularization techniques that created amazing results.

Despite the success of GANs in practice, the training procedure is known to be fickle and unreliable. The problems that arise in training are numerous and the most common ones are:

- **Failure to converge**. This is when the generator and discriminator do not find a good equilibrium. This usually happens because either the discriminator or generator becomes much better at their task than the other one. It can happen in two ways.

  - **Discriminator dominates.** If the discriminator dominates, its loss goes to zero and the generator can't learn from the diminished gradients so its output becomes and remains garbage.

  - **Generator dominates.** The generator can also dominate if it learns to output adversarial samples for the discriminator, samples that look nothing like the training data, yet tricks the discriminator into classifying it as real.

- **Mode collapse**. This is when the generator finds a single output or output class it can generate to trick the discriminator. The generator's outputs lose diversity and outputs the same thing (or very few different things) for every random input vector.

To avoid the failure modes and succeed with GAN training, one usually needs to find very specific hyperparameter settings carefully tuned to the model architectures and datasets at hand. There is no theoretical knowledge that can instruct how to select these. The GAN method has been invented from a practical stand-point and is based on heuristics and best practices. This is due to the fact that there is little known about optimizing non-convex functions like deep neural networks, and even less in a multi-objective min-max setting where the objectives conflict with each other such as in GANs.

Standard GAN training optimization methods such as Stochastic Gradient Descent Ascent (SGDA) and Adam [Goodfellow, 2017] do not even converge on some very simple min-max problems such as bilinear objectives. Despite there not being a complete theory on this subject, there is theory on multi-objective optimization in restricted settings, such as convex/concave min-max optimization, that goes beyond SGDA and Adam and has been suggested to translate well to the GAN domain. Two examples of more theoretically sound algorithms for min-max optimization are Optimistic Gradient Descent Ascent and the ExtraGradient method. They do not have any known convergence guarantees in the GAN setting either but they are thought to be a large improvement over regular SGDA and Adam [Gidel et al., 2018].

The purpose of this report is to examine whether Optimistic Gradient Descent Ascent and the ExtraGradient method have practical benefits when training GANs. The algorithms are evaluated on their sensitivity to hyperparameter settings, their stability and proneness to collapse, and their general performance.

2

Background

2.1 Generative Adversial Networks

The general setup for GANs involve a training set $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ and two neural networks, a generator G and a discriminator D. The generator is a parameterized deep neural network that takes a random noise vector $\mathbf{z} \in \mathbb{R}^n$ of chosen dimension n (hyperparameter) as input and outputs a sample $G(\mathbf{z})$ with the same dimensions as a sample from the training set. The discriminator is a parameterized deep neural network that takes a sample of said dimensions and outputs a number in the range $[0,1]$, signifying the discriminator's estimate of the probability that the given sample is from the training data and not from the generator's output.

The very first GAN objective function introduced in [Goodfellow et al., 2014] was formulated as

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{X}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))], \tag{2.1}$$

which is the log-likelihood of the binary classification task between training and generated data. The discriminator parameters are chosen to maximize the correctness of the classification and the generator parameters are chosen to minimize the correctness.

In the original GAN paper, it was immediately noted that this formulation does not work well in practice and the *non-saturating GAN* was proposed, where the generator's objective was switched into instead maximizing

$$\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log D(G(\mathbf{z}))],$$

while the discriminator's objective remained (2.1). This is not a min-max problem but is instead typically reformulated by switching the sign of the equations to arrive at two different coupled loss functions to be jointly minimized.

This formulation was shown to work remarkably well in practice with a standard gradient-based learning rule for both generator and discriminator in conjunction. Algorithm 1 describes the standard training scheme with vanilla stochastic gradient descent used as optimizer.

Algorithm 1: Stochastic Gradient Descent Ascent for training of GANs.

```
Initialize network weights $\theta_G$ and $\theta_D$
Set up learning rates $\eta_D$, $\eta_G$ and batch size $n$
Define batched stochastic loss functions:
$\mathscr{L}_D(\mathbf{X}, \mathbf{Z}, \theta_D, \theta_G) = -\frac{1}{n}\sum_{i=1}^{n} \log D(\mathbf{x}^{(i)}) - \frac{1}{n}\sum_{i=1}^{n} \log(1 - D(G(\mathbf{z}^{(i)})))$
$\mathscr{L}_G(\mathbf{Z}, \theta_D, \theta_G) = -\frac{1}{n}\sum_{i=1}^{n} \log D(G(\mathbf{z}^{(i)}))$
for $t$ = 0 ... $T$ - 1 do
    Sample minibatch of $n$ noise samples $\mathbf{Z} = \{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(n)}\}$,
      $\mathbf{z}^{(i)} \sim \mathscr{N}(0,1)\ \forall i$
    Sample minibatch of $n$ training data samples $\tilde{\mathbf{X}} = \{\tilde{\mathbf{x}}^{(1)}, ..., \tilde{\mathbf{x}}^{(n)}\}$
    Update discriminator parameters: $\theta_D \leftarrow \theta_D - \eta_D \nabla_{\theta_D} \mathscr{L}_D(\tilde{\mathbf{X}}, \mathbf{Z}, \theta_D, \theta_G)$
    Sample new minibatch of $n$ noise samples $\mathbf{Z} = \{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(n)}\}$,
      $\mathbf{z}^{(i)} \sim \mathscr{N}(0,1)\ \forall i$
    Update generator parameters: $\theta_G \leftarrow \theta_G - \eta_G \nabla_{\theta_G} \mathscr{L}_G(\mathbf{Z}, \theta_D, \theta_G)$
end
```

Often, as is described in the next section, an adaptive-step size method such as Adam is used for the gradient-update step.

2.2 Optimization schemes

Algorithm 1 can be slightly modified to allow for a generic optimization scheme to be used for the parameter update steps. This is described in Algorithm 2.

Algorithm 2: General scheme for training of GANs.

```
Initialize network weights $\theta_G$ and $\theta_D$
Set up learning rates $\eta_D$, $\eta_G$ and batch size $n$
Define batched stochastic loss functions:
$\mathscr{L}_D(\mathbf{X}, \mathbf{Z}, \theta_D, \theta_G) = -\frac{1}{n}\sum_{i=1}^{n} \log D(\mathbf{x}^{(i)}) - \frac{1}{n}\sum_{i=1}^{n} \log(1 - D(G(\mathbf{z}^{(i)})))$
$\mathscr{L}_G(\mathbf{Z}, \theta_D, \theta_G) = -\frac{1}{n}\sum_{i=1}^{n} \log D(G(\mathbf{z}^{(i)}))$
for $t$ = 0 ... $T$ - 1 do
    Sample minibatch of $n$ noise samples $\mathbf{Z} = \{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(n)}\}$,
      $\mathbf{z}^{(i)} \sim \mathscr{N}(0,1)\ \forall i$
    Sample minibatch of $n$ training data samples $\tilde{\mathbf{X}} = \{\tilde{\mathbf{x}}^{(1)}, ..., \tilde{\mathbf{x}}^{(n)}\}$
    Update discriminator parameters:
      $\theta_D \leftarrow \mathrm{Step}(\theta_D, \eta_D, \nabla_{\theta_D} \mathscr{L}_D(\tilde{\mathbf{X}}, \mathbf{Z}, \theta_D, \theta_G), t)$
    Sample new minibatch of $n$ noise samples $\mathbf{Z} = \{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(n)}\}$,
      $\mathbf{z}^{(i)} \sim \mathscr{N}(0,1)\ \forall i$
    Update generator parameters: $\theta_G \leftarrow \mathrm{Step}(\theta_G, \eta_G, \nabla_{\theta_G} \mathscr{L}_G(\mathbf{Z}, \theta_D, \theta_G), t)$
end
```

Letting Step = Step$_{\text{SGD}}$ in Algorithm 2, where

$$\text{Step}_{\text{SGD}}(\theta,\eta,g,t) = \theta - \eta g \tag{2.2}$$

makes Algorithm 2 reduce to Algorithm 1. A very common choice of optimization scheme is Adam (Algorithm 2 using the update step in Algorithm 3), introduced in [Kingma and Ba, 2017]. Note that g^2 means element-wise square of the g vector.

```
Algorithm 3: Step_ADAM($\theta,\eta,g,t$).
    Require hyperparameters: $\beta_1,\beta_2,\varepsilon$
    Require initial vectors if $t=0$: $m=0, v=0$
    Update biased first moment estimate: $m \leftarrow \beta_1 m + (1-\beta_1)g$
    Update biased second raw moment estimate: $v \leftarrow \beta_2 v + (1-\beta_2)g^2$
    Compute bias-corrected first moment estimate: $\hat{m} \leftarrow m/(1-\beta_1^t)$
    Compute bias-corrected second raw moment estimate: $\hat{v} \leftarrow v/(1-\beta_2^t)$
    Return $\theta - \eta\hat{m}/(\sqrt{\hat{v}}+\varepsilon)$
```

SGD and Adam are both taken from minimization or maximization settings such as classification training. As we will see in the next section, these methods are not actually very well suited for min-max optimization.

The problem with SGDA

To illustrate why Stochastic Gradent Descent Ascent might not be well suited as an optimizing scheme for min-max problems such as GANs, [Goodfellow, 2017] offered an extremely simple toy example

$$\min_x \max_y xy \tag{2.3}$$

on which Gradent Descent Ascent fails to converge. Considering this very simple example with unique Nash equilibrium at $(x,y)=(0,0)$, we can investigate how a Gradient Descent Ascent scheme behaves when applied to this problem. Algorithm 4[1] will serve as our general numerical min-max optimization scheme that we will plug in different update algorithms to, starting with GDA, (2.2).

[1] Note that Algorithm 4 differs from the GAN training algorithm (Algorithm 2) in that the GAN generator's update step uses the discriminator's parameter from the same iteration, while the general min-max scheme uses the past iteration's parameters for both update steps. The GAN training algorithm follows convention for GAN traning, and is how the original GAN framework was formulated. The results of our theoretical investigation holds for Algorithm 4 but for our purposes, the gist of the results would be the same if $y_{t+1} \leftarrow \text{Step}(y_t,\eta,-\nabla_y V(x_t,y_t),t)$ was replaced with $y_{t+1} \leftarrow \text{Step}(y_t,\eta,-\nabla_y V(x_{t+1},y_t),t)$.

Algorithm 4: General numerical min-max optimization scheme for problems of type $\min_x \max_y V(x,y)$.

```
Require initial positions: $x_0, y_0$
Require learning rate: $\eta$
for $t$ = 0 ... T-1 do
    $x_{t+1} \leftarrow \text{Step}(x_t, \eta, \nabla_x V(x_t, y_t), t)$
    $y_{t+1} \leftarrow \text{Step}(y_t, \eta, -\nabla_y V(x_t, y_t), t)$
end
```

Consider x and y as two different parameters, a step-size η, update function GDA, and the objective function to be optimized for $V(x,y) = xy$, we get the following optimization scheme

$$\begin{cases} x_{t+1} = x_t - \eta \nabla_x V(x_t, y_t) \\ y_{t+1} = y_t + \eta \nabla_y V(x_t, y_t) \end{cases}$$

Calculating the particular gradients gives

$$\begin{cases} x_{t+1} = x_t - \eta y_t \\ y_{t+1} = y_t + \eta x_t \end{cases},$$

which in matrix form is

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & -\eta \\ \eta & 1 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix}.$$

This system has solution

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} 1 & -\eta \\ \eta & 1 \end{pmatrix}^t \begin{pmatrix} x_0 \\ y_0 \end{pmatrix},$$

and we retrieve a closed form solution by diagonalizing the exponentiated matrix

$$\begin{aligned} \begin{pmatrix} 1 & -\eta \\ \eta & 1 \end{pmatrix}^t &= \frac{1}{2} \begin{pmatrix} -i & i \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1-\eta i & 0 \\ 0 & 1+\eta i \end{pmatrix}^t \begin{pmatrix} i & 1 \\ -i & 1 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} -i & i \\ 1 & 1 \end{pmatrix} \begin{pmatrix} (1-\eta i)^t & 0 \\ 0 & (1+\eta i)^t \end{pmatrix} \begin{pmatrix} i & 1 \\ -i & 1 \end{pmatrix}. \end{aligned}$$

Performing the matrix multiplications gives

$$= \frac{1}{2} \begin{pmatrix} (1+\eta i)^t + (1-\eta i)^t & i(1+\eta i)^t - i(1-\eta i)^t \\ -i(1+\eta i)^t + i(1-\eta i)^t & (1+\eta i)^t + (1-\eta i)^t \end{pmatrix}$$

and using the complex conjugate identities $\bar{z}^n = \bar{z^n}$, $\text{Re}(z) = \frac{z+\bar{z}}{2}$ and $\text{Im}(z) = \frac{z-\bar{z}}{2i}$:

$$= \begin{pmatrix} \text{Re}((1+\eta i)^t) & -\text{Im}((1+\eta i)^t) \\ \text{Im}((1+\eta i)^t) & \text{Re}((1+\eta i)^t) \end{pmatrix}.$$

Finally, we apply Euler's formula to get

$$\begin{pmatrix} 1 & -\eta \\ \eta & 1 \end{pmatrix}^t = (\sqrt{1+\eta^2})^t \begin{pmatrix} \cos(\tan^{-1}(\eta)t) & -\sin(\tan^{-1}(\eta)t) \\ \sin(\tan^{-1}(\eta)t) & \cos(\tan^{-1}(\eta)t) \end{pmatrix}.$$

Thus we have arrived at that using the standard Gradient Descent Ascent scheme on the original objective function xy yields the iterates

$$\begin{cases} x_t = (\sqrt{1+\eta^2})^t(\cos(\tan^{-1}(\eta)t)x_0 - \sin(\tan^{-1}(\eta)t)y_0) \\ y_t = (\sqrt{1+\eta^2})^t(\sin(\tan^{-1}(\eta)t)x_0 + \cos(\tan^{-1}(\eta)t)y_0) \end{cases},$$

which is a diverging, oscillating trajectory and the Nash equilibrium will never be reached. Gradient Descent Ascent fails on even this very simple toy problem, which can be seen as a hint to why it might be difficult to train GANs with it since GANs even have added difficulties such as stochasticity, non-convexity, non-differentiability and non-symmetrical objectives.

We plot the oscillating trajectory of the GDA update scheme in 2D in the end of this section.

Optimistic Gradient Descent Ascent (OGDA)

The first optimization scheme we will consider that has better properties for min-max optimization is Optimistic Gradient Descent Ascent (OGDA). It was introduced for GAN training in [Daskalakis et al., 2017]. The algorithm is simple and elegant, and looks like an extension of Gradient Descent with an added term that can be interpreted as a look-back step or "negative-momentum" term. The GAN training algorithm is obtained by using the following update step in Algorithm 2.

Algorithm 5: $\text{Step}_{\text{OGD}}(\theta, \eta, g, t)$.

```
Require initial vector if $t = 0$: $g_{\text{prev}} = 0$
Calculate new parameters: $\theta_{\text{new}} \leftarrow \theta - 2\eta g + \eta g_{\text{prev}}$
Store gradients for next iteration: $g_{\text{prev}} \leftarrow g$
Return $\theta_{\text{new}}$
```

We will now show that OGDA converges for our toy problem. Applying OGDA on our toy problem (2.3) using Algorithm 4 with objective function $V(x,y) = xy$ gives

$$\begin{cases} x_{t+1} = x_t - 2\eta\nabla_x V(x_t, y_t) + \eta\nabla_x V(x_{t-1}, y_{t-1}) \\ y_{t+1} = y_t + 2\eta\nabla_y V(x_t, y_t) - \eta\nabla_y V(x_{t-1}, y_{t-1}) \end{cases},$$

which is

$$\begin{cases} x_{t+1} = x_t - 2\eta y_t + \eta x_{t-1} \\ y_{t+1} = y_t + 2\eta x_t - \eta x_{t-1} \end{cases},$$

with gradients inserted. This can be written in matrix form as

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & -2\eta \\ 2\eta & 1 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{pmatrix} 0 & \eta \\ -\eta & 0 \end{pmatrix} \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix},$$

and in block-matrix form as

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ x_t \\ y_t \end{pmatrix} = \begin{pmatrix} 1 & -2\eta & 0 & \eta \\ 2\eta & 1 & -\eta & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \\ x_{t-1} \\ y_{t-1} \end{pmatrix}.$$

The solution to this is

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ x_t \\ y_t \end{pmatrix} = \begin{pmatrix} 1 & -2\eta & 0 & \eta \\ 2\eta & 1 & -\eta & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}^t \begin{pmatrix} x_1 \\ y_1 \\ x_0 \\ y_0 \end{pmatrix}.$$

We can study the stability of this matrix by studying its eigenvalues that are the roots of its characteristic polynomial

$$P(\lambda) = \lambda^4 - 2\lambda^3 + (1+4\eta^2)\lambda^2 - 4\eta^2\lambda + \eta^2.$$

If all eigenvalues lie within the open unit disk, the system is stable and the optimization process will converge. To simplify the analysis we can apply the Möbius transform $\lambda \mapsto \frac{\lambda+1}{\lambda-1}$, and instead make sure that the transformed polynomial $(\lambda-1)^4 P(\frac{\lambda+1}{\lambda-1})$ has roots within the open left half-plane [Smith, 1970]. After the transformation we retrieve

$$\begin{aligned}&(\lambda+1)^4 - 2(\lambda+1)^3(\lambda-1) + (1+4\eta^2)(\lambda+1)^2(\lambda-1)^2 \\ &-4\eta^2(\lambda+1)(\lambda-1)^3 + \eta^2(\lambda-1)^4.\end{aligned}$$

Expanding the parenthesis and collecting terms gives

$$\underbrace{\eta^2}_{a_4}\lambda^4 + \underbrace{4\eta^2}_{a_3}\lambda^3 + \underbrace{(4-2\eta^2)}_{a_2}\lambda^2 + \underbrace{(8-12\eta^2)}_{a_1}\lambda + \underbrace{4+9\eta^2}_{a_0}.$$

Using the Routh–Hurwitz stability criterion [Hurwitz, 1895] it must hold that

$$\begin{cases} a_4 > 0 \\ a_3 a_2 > a_4 a_1 \\ a_3 a_2 a_1 > a_4 a_1^2 + a_3^2 a_0 \end{cases}.$$

The first and second conditions are true for all η. The third yields

$$4\eta^2(4-2\eta^2)(8-12\eta^2) > \eta^2(8-12\eta^2)^2 + 16\eta^4(4+9\eta^2),$$

which can be algebraically simplified as follows

$$96\eta^4 - 256\eta^2 + 128 > 288\eta^4 - 128\eta^2 + 64$$

which is equivalent to

$$\begin{aligned}
&192\eta^4 + 128\eta^2 - 64 < 0 \\
\iff\ &\eta^4 + \frac{2}{3}\eta^2 - \frac{1}{3} < 0 \\
\iff\ &\frac{1}{3}(3\eta^2 - 1)(\eta^2 + 1) < 0.
\end{aligned}$$

Since η is real, we can divide by $(\eta^2 + 1)$ to get

$$3\eta^2 - 1 < 0 \iff -\frac{1}{\sqrt{3}} < \eta < \frac{1}{\sqrt{3}}.$$

This concludes that for OGDA, in contrast to GDA, there are constant step-size choices of η for which the algorithm converges. Negative step-sizes come up as valid in this calculation since it corresponds to swapping the min/max objective to a max/min objective which has the same solution in this particular toy problem.

It has been shown more generally in [Malitsky and Tam, 2020] that OGDA converges (linearly) to a solution under assumptions that the objective function is smooth and (strongly) convex-(strongly) concave for any step-size $\eta \in (0, \frac{1}{2L})$, where L is a combined Lipschitz constant for the gradients of the min-max objective, see [Mokhtari et al., 2019] for details. For our toy problem, L is 1 and the general result is compatible with our exact condition, although the general result is slightly more restrictive.

[Daskalakis et al., 2017] additionally introduced an Adam-variant of OGD which is described in Algorithm 6. Optimistic Adam is of special interest for training GANs as it combines the theoretically interesting OGD algorithm with the empirically successful Adam algorithm. Algorithm 6 can be used in Algorithm 2 for GAN training or Algorithm 4 for general min-max optimization.

Algorithm 6: $\text{Step}_{\text{OptimADAM}}(\theta, \eta, g, t)$.

```
Require hyperparameters: $\beta_1, \beta_2, \varepsilon$
Require initial vectors if $t = 0$: $m = 0, v = 0$
$\hat{m}_{\text{prev}} \leftarrow m/(1 - \beta_1^{t-1})$
$\hat{v}_{\text{prev}} \leftarrow v/(1 - \beta_2^{t-1})$
Update biased first moment estimate: $m \leftarrow \beta_1 m + (1 - \beta_1)g$
Update biased second raw moment estimate: $v \leftarrow \beta_2 v + (1 - \beta_2)g^2$
Compute bias-corrected first moment estimate: $\hat{m} \leftarrow m/(1 - \beta_1^t)$
Compute bias-corrected second raw moment estimate: $\hat{v} \leftarrow v/(1 - \beta_2^t)$
Return $\theta - 2\eta\hat{m}/(\sqrt{\hat{v}} + \varepsilon) + \eta\hat{m}_{\text{prev}}/(\sqrt{\hat{v}_{\text{prev}}} + \varepsilon)$
```

OGD has no further time cost than regular SGD and Optimistic Adam has no further time cost than regular Adam. OGD has a larger memory cost than SGD since the last iterate needs to be stored, but this cost is constant and often not a concern in practice.

ExtraGradient method (EG)

The second optimization scheme with equally good theoretical properties is the ExtraGradient method introduced for GAN training in [Gidel et al., 2018]. It is different from the other schemes in that it features an extrapolation (also known as look-ahead) step.

Applying ExtraGradient to our toy problem (2.3) gives iterates

$$\begin{cases} x_{t+1/2} = x_t - \eta \nabla_x V(x_t, y_t) \\ y_{t+1/2} = y_t + \eta \nabla_y V(x_t, y_t) \\ x_{t+1} = x_t - \eta \nabla_x V(x_{t+1/2}, y_{t+1/2}) \\ y_{t+1} = y_t + \eta \nabla_y V(x_{t+1/2}, y_{t+1/2}) \end{cases} .$$

Inserting the calculated gradients gives

$$\begin{cases} x_{t+1/2} = x_t - \eta y_t \\ y_{t+1/2} = y_t + \eta x_t \\ x_{t+1} = x_t - \eta y_{t+1/2} \\ y_{t+1} = y_t + \eta x_{t+1/2} \end{cases} ,$$

which can be reorganized into

$$\begin{cases} x_{t+1} = x_t - \eta (y_t + \eta x_t) \\ y_{t+1} = y_t + \eta (x_t - \eta y_t) \end{cases} .$$

Further reorganization gives

$$\begin{cases} x_{t+1} = (1-\eta^2) x_t - \eta y_t \\ y_{t+1} = (1-\eta^2) y_t + \eta x_t \end{cases} ,$$

which in matrix form is

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} 1-\eta^2 & -\eta \\ \eta & 1-\eta^2 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix} .$$

This system has the solution

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} 1-\eta^2 & -\eta \\ \eta & 1-\eta^2 \end{pmatrix}^t \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} ,$$

which can be rewritten by factoring out $\sqrt{(1-\eta^2)^2+\eta^2}$ from the matrix into

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \left(\sqrt{(1-\eta^2)^2+\eta^2} \begin{pmatrix} \frac{1-\eta^2}{\sqrt{(1-\eta^2)^2+\eta^2}} & -\frac{\eta}{\sqrt{(1-\eta^2)^2+\eta^2}} \\ \frac{\eta}{\sqrt{(1-\eta^2)^2+\eta^2}} & \frac{1-\eta^2}{\sqrt{(1-\eta^2)^2+\eta^2}} \end{pmatrix} \right)^t \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} .$$

We can note that the matrix is a rotation matrix multiplied by constant factor $\sqrt{(1-\eta^2)^2+\eta^2}$, which means that for the system to be stable it must hold that

$$\sqrt{(1-\eta^2)^2+\eta^2} < 1 \iff -1 < \eta < 1.$$

This concludes that also for EG, there are sensible choices of η for which the algorithm converges. In fact, there are general results for the convergence of EG as well. For example, EG converges (linearly) to the solution under assumptions that the objective function is smooth and (strongly) convex-(strongly) concave for any step-size $\eta \in (0, \frac{1}{L})$ where L is the objective function's gradients' combined Lipschitz constant [Korpelevich, 1976] [Tseng, 2000]. As stated, L is 1 for the problem we are considering.

The ExtraGradient method can fit in our general framework of optimization schemes, by noting that it is equivalent to stepping from the previous step's parameters every other iteration. This is detailed in Algorithm 7.

Algorithm 7: $\text{Step}_{\text{EG}}(\theta, \eta, g, t)$.

```
if $t$ is even then
    $\theta_{\text{stored}} \leftarrow \theta$
    Return $\theta - \eta g$
else
    Return $\theta_{\text{stored}} - \eta g$
end if
```

In [Gidel et al., 2018] an Adam-variant of EG called ExtraAdam was introduced that is very interesting for GAN training. It is detailed in Algorithm 8.

Algorithm 8: $\text{Step}_{\text{ExtraAdam}}(\theta, \eta, g, t)$.

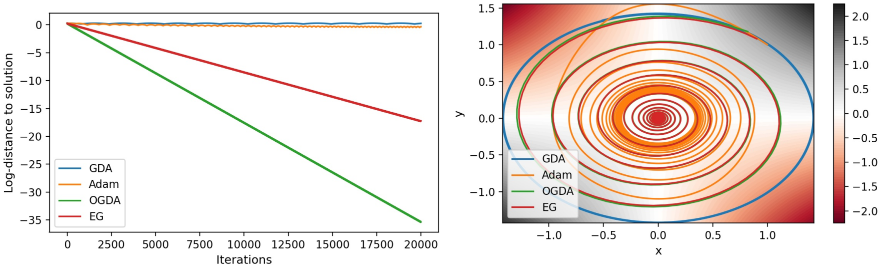
```
Require hyperparameters: $\beta_1, \beta_2, \varepsilon$
Require initial vectors if $t = 0$: $m = 0, v = 0$
Update biased first moment estimate: $m \leftarrow \beta_1 m + (1-\beta_1) g$
Update biased second raw moment estimate: $v \leftarrow \beta_2 v + (1-\beta_2) g^2$
Compute bias-corrected first moment estimate: $\hat{m} \leftarrow m/(1-\beta_1^t)$
Compute bias-corrected second raw moment estimate: $\hat{v} \leftarrow v/(1-\beta_2^t)$
    if $t$ is even then
        $\theta_{\text{stored}} \leftarrow \theta$
        Return $\theta - \eta \hat{m}/(\sqrt{\hat{v}} + \varepsilon)$
    else
        Return $\theta_{\text{stored}} - \eta \hat{m}/(\sqrt{\hat{v}} + \varepsilon)$
    end if
```

Algorithm 7 and Algorithm 8, like the other optimization schemes, can be applied to GAN training using Algorithm 2 and numerical min-max optimization using Algorithm 4.

When stated as in Algorithm 7 and Algorithm 8, the two algorithms have the same time cost per iteration as GD and Adam respectively. The memory cost is, just as with OGD, larger, since one extra set of parameters is stored, but this is a constant cost and often not a concern in practice.

Numerical analysis

The toy problem (2.3) can be studied numerically to visually examine the different algorithms' behaviors. We run the general numerical min-max optimization scheme Algorithm 4 with the different update algorithms GDA (2.2), Adam (Algorithm 3), OGDA (Algorithm 5) and EG (Algorithm 7) for 20 000 iterations. For inspection, we plot the iterates' trajectories and their distances to the unique Nash equilibrium solution at $(x,y) = (0,0)$. An extensive hyperparameter search was performed for all experiments in this section and the choices were made to illustrate the algorithms' best-case performances on the problem. No choices were found that could improve the results materially.



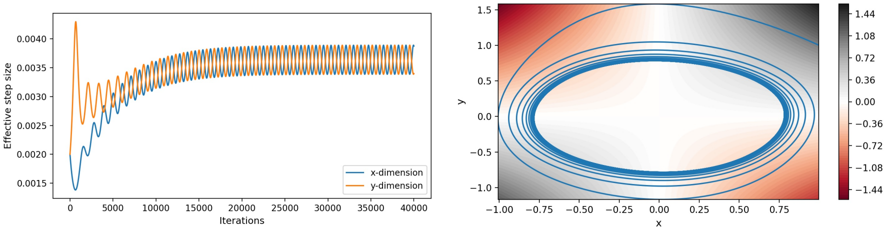
(a) Distances to solution.

(b) Trajectories.

Figure 2.1: GDA, Adam, OGDA, and EG applied to the toy min-max problem $\min_x \max_y xy$. OGDA and EG both converge to the solution at $(x,y) = (0,0)$. Adam does better than GDA but does not converge. The hyperparameters used were $\eta = 0.001$ for GDA, $\eta = 0.09$ for OGDA and EG, and $\eta = 0.002, \beta_1 = 0.5, \beta_2 = 0.999$ for Adam. The initial positions were all $(x_0, y_0) = (1,1)$.

In Figure 2.1 we observe the results we expect from our theoretical investigations. GDA does not converge and slowly oscillates outwards, while OGDA and EG both successfully converge to the solution. OGDA is exactly twice as fast as EG. This may be because the stabilizing steps of both algorithms perform similarly on this problem, but the stabilizing step in EG (its extrapolation step) is twice as costly since it invokes an additional gradient descent ascent step. From the numerical analysis, we see that Adam's convergence properties are slightly better than GDA but ultimately never converges either.

Adam's behaviour is worth investigating more closely, especially considering it is the most common optimization scheme used for GANs. The Adam algorithm can be understood as adjusting the effective learning rate in the different dimensions over time during optimization. From Algorithm 3, we can extract a quantity we will call *effective step size*, $\eta_{\text{effective}} = \frac{\hat{m}}{\sqrt{\hat{v}}+\varepsilon}$. In Figure 2.2, we see how the dynamic step sizes are advantageous in the beginning of the process but the effective step sizes ultimately converge to an oscillating limit cycle and no further progress to the solution is made.

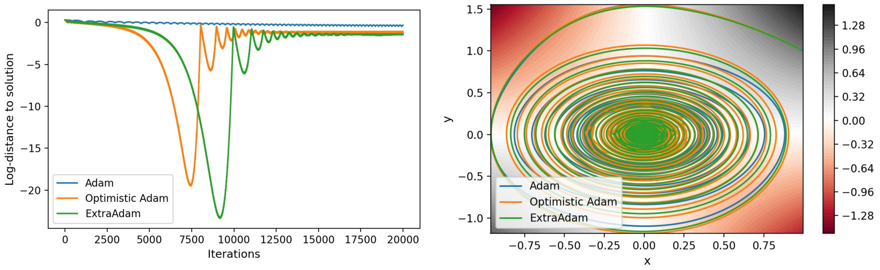


(a) Effective step sizes in x- and y-dimensions.

(b) Trajectory.

Figure 2.2: The behaviour of Adam on the min-max problem (2.3). The dynamic learning rates improves Adam's performance over GDA as the iterates make initial progress towards the solution but eventually get stuck in an orbiting trajectory. The hyperparameters used were $\eta = 0.002, \beta_1 = 0.1, \beta_2 = 0.999$. The initial position was $(x_0, y_0) = (1, 1)$.

Observing the Adam-variants in the same setting is slightly puzzling, yet promising. In Figure 2.3, we see Optimistic Adam (Algorithm 6) and ExtraAdam (Algorithm 8) momentarily reach very close to the solution, but do not completely succeed by staying there. Overall, this is more than regular Adam achieves and is therefore a promising sign moving into the more unpredictable and uninterpretable GAN regime.



(a) Distances to solution.

(b) Trajectories.

Figure 2.3: Adam, Optimistic Adam, and ExtraAdam applied to the toy min-max problem (2.3). Optimistic Adam and ExtraAdam both momentarily reach very close to the solution but fail to stay there. The hyperparameters used were $\eta = 0.002, \beta_1 = 0.1, \beta_2 = 0.999$ for Adam, and $\eta = 0.02, \beta_1 = 0.1, \beta_2 = 0.999$ for Optimistic Adam and Extraadam. The initial positions were all $(x_0, y_0) = (1, 1)$.

2.3 GANs in practice

GANs have been heavily studied and improved upon since the original formulation of the framework. This section will survey a few notable inventions that have improved the state-of-the-art of GANs and have been incorporated into the experiments of this thesis.

Objectives

There has been a lot of experimentation with different loss functions, and two that have had large empirical success are WGAN and its close relative WGAN-GP.

WGAN. The Wasserstein GAN objective was introduced in [Arjovsky et al., 2017] to alleviate training problems with the standard GAN objective, and was motivated by the fact that the standard divergence that classical GANs minimize, *Jensen-Shannon divergence*, can potentially be discontinuous with respect to the generator's parameters. The WGAN objective is

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{X}}(\mathbf{x})}[D_{\text{critic}}(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{Z}}(\mathbf{z})}[D_{\text{critic}}(G(\mathbf{z}))].$$

If the discriminator network (or *critic* as it's called in the WGAN setting) is a 1-Lipschitz function[2], there are some theoretical benefits over the standard objective. In the original WGAN paper, the proposed method to enforce this was to clip all weights in the critic to be in the closed range $[-c, c]$, for some selectable hyperparameter $c > 0$. The critic network also differs from a typical discriminator

[2] Lipschitz continuous function with Lipschitz constant $L \in (0, 1]$.

network in that it does not have a sigmoid function on its output and thus can not be interpreted as a probability that its input is real or fake. It is interpreted as a score on how real the input looks (on an arbitrary and undefined scale).

An added benefit to the WGAN objective compared to the standard objective is that its function value during training has been empirically shown to correlate with human judgement on sample quality.

WGAN-GP. The WGAN objective was further improved by replacing the weight clipping with a gradient penalty regularisation term with the objective to promote 1-Lipschitz continuity of the critic network. This variation of WGAN is called the WGAN-Gradient Penalty (or WGAN-GP) and has been shown empirically to be an improvement over regular WGAN. [Gulrajani et al., 2017]

The gradient penalty is applied to the critic's loss function which for a batch size n is stated as

$$L_{\text{critic}} = \frac{1}{n} \sum_{i=1}^{n} D_{\text{critic}}(G(z_i)) - D_{\text{critic}}(x_i) + \lambda(||\nabla D_{\text{critic}}(\hat{x}_i)||_2 - 1)^2,$$

where z_i is a randomly sampled latent vector, x_i a randomly selected training sample, $\hat{x}_i = \varepsilon x_i + (1-\varepsilon)G(z_i)$ for a uniformly randomly selected $\varepsilon \in [0,1]$, and $\lambda \geq 0$ is a hyperparameter. The gradient penalty is enforced on points lying on straight lines between training data and generated data which was motivated when it was introduced by the fact that enforcing it everywhere is intractable and this method resulted in good experimental results.

Averaging

Averaging is a technique that does not affect the training directly but is implemented to improve the final generator model. By taking the average of all weights used throughout training, the final evaluation results can often be improved. There are two common averaging schemes used in GAN training, the first one being uniform average

$$\theta_{\text{final}} = \frac{1}{T} \sum_{t=0}^{T-1} \theta_t,$$

where θ_t is the model weights after t training iterations, T is the total number of training iterations and θ_{final} is the model weights used for evaluation after training is completed.

Exponential averaging is another common averaging scheme and is defined as

$$\theta_{\text{final}} = \sum_{t=0}^{T-1} \beta^{T-t} \theta_t,$$

where $0 < \beta < 1$ is a hyperparameter.

Averaging can be implemented online during training using

$$\bar{\theta}_t = (1-\rho_t)\bar{\theta}_{t-1} + \rho_t \theta_t,$$

where $\rho_t = \frac{1}{t}$ for uniform averaging and $\rho_t = 1-\beta$ for exponential averaging. Using the online scheme, at the end of training $\bar{\theta}_T$ is θ_{final} in the respective cases.

Evaluation metrics

A problem with GAN training and image generation broadly, is the issue of evaluation. The goal of the training process isn't simply to minimize the two networks' loss functions, and it is debatable if it is even to find a Nash equilibrium between the two objectives. Separate evaluation metrics such as the *Inception Score* and the *Frechet Inception Distance* have been invented to score generator models' performance in both fidelity and diversity.

Inception Score (IS). The Inception Score (introduced in [Salimans et al., 2016]) is an automatic evaluation metric of the quality of generated samples from generative models. It has been shown to correlate well with human judgement on the quality of generated images.

The Inception Score works only in class conditional settings where the data is separated into different classes. Typical classification datasets can therefor be evaluated with the Inception Score as the method relies on using a pre-trained classifier network (with high accuracy).

Given a generative model G that encodes a distribution p_G from which we can sample images, and a classifier model that approximates a distribution p from which we can retrieve estimates of $p(y|\mathbf{x})$, the probability of a sample $\mathbf{x}$ belonging to a class y, the Inception Score IS is defined as

$$\mathrm{IS}(G) = e^{\mathbb{E}_{\mathbf{x}\sim p_G} D_{KL}(p(y|\mathbf{x})\ ||\ p(y))},$$

where $D_{KL}(q\ ||\ r)$ is the Kullback–Leibler divergence between the distributions q and r.

With q being the conditional class distribution $p(y|\mathbf{x})$ and r being the marginal class distribution $p(y) = \int_{\mathbf{x}} p(y|\mathbf{x}) p_G(\mathbf{x})$, their KL-divergence will be high if the conditional class distribution has low entropy (signifying that the classifier classifies the generated images with certainty) and the marginal class distribution has high entropy (signifying that the generated images are varied and represent many different classes).

With N randomly sampled generated images we can approximate the expectation of the KL-divergence as

$$\mathrm{IS}(G) \approx e^{\frac{1}{N}\sum_i^N D_{KL}(p(y|\mathbf{x_i})\ ||\ p(y))},$$

where $p(y) = \frac{1}{N}\sum_i^N p(y|\mathbf{x_i})$ is the approximated marginal class distribution. The class conditional distribution itself $p(y|\mathbf{x_i})$ is calculated with a pre-trained classifier model, typically an Inception Network (hence the name).

Since the Inception Score is a good metric on the quality of the generator's output, it begs the question why not use it as an objective to optimize for directly. The problem with doing that is that the generator will overfit to the classifier's peculiarities and will generate adverserial samples to it. To circumvent this, one must use some kind of heavy regularization, such as ensembling a multitude of classifiers or aggressive data augmentation on the generator's output, which can be successful but hard to scale up to large and complex datasets. This was found out during the work on this thesis by succeeding on MNIST (handwritten digits) but not managing to achieve anything on more complicated data sets.

Fréchet Inception Distance (FID). While Inception Score uses the classification predictions of the Inception Network to score the diversity and legibility of the generated samples, the Fréchet Inception Distance (FID) side-steps the need for classes in the training data by instead using the activations of an inner layer.

The FID score is the Wasserstein distance between two multidimensional Gaussian distributions with mean and standard deviations fitted to the activations of a deep layer in the Inception network, when fed with the training data and the generated images respectively. Formally it's defined as

$$\mathrm{FID}(G) = ||\mu_{\text{real}} - \mu_{\text{gen}}||_2^2 + \mathrm{tr}(\Sigma_{real} + \Sigma_{\text{gen}} - 2\sqrt{\Sigma_{\text{real}}\Sigma_{\text{gen}}}),$$

where μ_{real} and Σ_{real} are the mean vector and covariance matrix of the activations of the chosen layer of the training data and μ_{gen} and Σ_{gen} are the mean and covariance of the activations of the chosen layer of the generated data.

The inner layer chosen is typically the last pool layer of the InceptionNet-v3 architecture [Szegedy et al., 2015]. The input of InceptionNet-v3 can be any tensor with three channels and width and height larger than 75 by 75. Any data smaller than this can be used by first upscaling it.

The FID score can be calculated for any training data without having a specific pretrained classifier model and thus works just as well for an unlabeled dataset. Usually, an Inception network trained on ImageNet is used.

The FID score has been shown to correlate even better than the Inception Score with human judgement on sample quality and is the most common evaluation metric for GANs. Despite not having any constraint for sample diversity, overfitting for FID score has not been an issue since it is only used for evaluation and not optimized for directly.

Architectures

The biggest breakthroughs in GAN results have arguably been by creating meticulously crafted architectures.

BigGAN and U-Net GAN. The BigGAN architecture was introduced in [Brock et al., 2018] and became state-of-the-art in high-resolution image synthesis. Its main contribution was, as the name implies, to scale up GANs in parameter count, training batch size, and output resolution.

The architecture is modularly defined as a sequence of ResBlocks, which are typical convolutional network components that split the input up into two parallell computation tracks that are later joined together as the output of the block. The generator and discriminator have different ResBlocks where the main difference is that the generator's contain upsampling layers that double the width and height of the image while the discriminator's ResBlocks' convolutions downsample the image. Both architectures also include non-local blocks, which are variants of ResBlocks introduced in [Wang et al., 2018] that adds more non-local dependencies in convolutional nets that otherwise can have limited *receptive fields*, where a receptive field is the size of the region in the input that affects a feature [Araujo et al., 2019].

The latent vector is introduced in the generator model with skip connections, with a method called skip-z that splits it up into smaller chunks which are passed into the different ResBlocks one by one at different levels.

The BigGAN paper also introduced a truncation trick, which was based on an observation that truncating or shrinking the sampling space that latent vectors are sampled from in the evaluation phase improves the results. This is probably because this space was more often sampled in training. This comes at a cost in sample variation and a trade-off has to be made between quality and variation when selecting the degree of truncation.

All in all, the BigGAN architecture was a leap forward in GAN architectures, but it was not without issues. It could become unstable and experience training collapse, which will be explored in Chapter 4.

The U-Net GAN, introduced in [Schönfeld et al., 2020], was a further development of the BigGAN architecture. The U-Net is a successful segmentation architecture from [Ronneberger et al., 2015] that U-Net GAN uses as discriminator. The main idea behind using a segmentation network as discriminator is to provide per-pixel feedback to the generator, so it can know which pixels specifically that tip off the discriminator. Together with some novel data augmentation techniques and regularisation involving cut-mixing generator and training data into the discriminator's training, the U-Net GAN improved BigGAN's results further.

The U-Net discriminator can be combined with any generator architecture and not just with the BigGAN generator.

StyleGAN 2. StyleGAN, published by researchers at NVIDIA in [Karras et al., 2018], is the best performing GAN to date.

Its main invention is a style-based generator that feature a mapping network at its head, where the random latent vector is passed through a deep fully-connected network before being passed into the convolutional generator network through skip connections. The idea behind the mapping network is to allow the network to learn a highly structured intermediate latent space of style vectors.

The generator network is an otherwise fairly standard convolutional network with upsampling layers. The style vector is introduced into the convolutional network through skip connections at different depths and these are of a certain adaptive instance normalization (AdaIN) type that is taken from the style transfer literature. This operation was simplified and stripped away in StyleGAN 2 through *weight demodulation*. StyleGAN also uses *progressive growing* to increase the output resolution in phases during training but this is also removed in StyleGAN 2. It's also notable that before each skip connection is an addition of unstructured or "non-mapped" noise that is there to provide variation and randomness to the generated images.

The main design goal of StyleGAN was to create meaning in the latent space. Apart from the aforementioned features, a novel regularization technique called *style mixing* or *mixing regularization* was introduced that amounts to passing parts from different latent vectors to different skip connections in the generator network which is said to encourage localizing different blocks to different levels of details in the images.

StyleGAN 2 was presented shortly after StyleGAN in [Karras et al., 2019], and apart from the simplification it offered in terms of replacing AdaIN with weight demodulation and removing progressive growing, it introduced two new regularization techniques called path length regularization and lazy regularization.

Weight demodulation was an insight that sped up the execution time of the StyleGAN network significantly. They realised that the desired behavior from the operations in the skip connections in AdaIN could effectively be replaced by modulating the convolutional layers' weights with the style vector's weights instead.

Path length regularization is a regularization term to the loss function with the goal of encouraging steps in the style space to affect change in the image space proportional to the step length. Lazy regularization amounts to not calculating the regularization terms in the loss function every iteration, but only in fixed intervals, which yielded a large efficiency boost without affecting results. Overall, StyleGAN 2 is an optimizied version of StyleGAN that allows it to be trained on larger resolutions, batch sizes and for more iterations.

StyleGAN 2 typically uses either WGAN-GP or the original non-saturating loss as loss functions. When WGAN-GP is used, the gradient penalty term is lazily calculated (only every few iterations).

Datasets

A very common way to benchmark GAN performance is face-generation, that is generating convincing images of faces. There is a ton of variety, yet clear structure, and humans can easily notice how well the finer details are generated.

CelebA. The CelebA dataset (or *the Large-scale CelebFaces Attributes Dataset*) was introduced in [Liu et al., 2015] and has become a classical GAN benchmark dataset. It contains 200 000 portrait photos of celebrities that have been scraped from the internet and made available for non-commercial research purposes. The image resolutions are 178 x 218 pixels (width x height) but are often cropped to squares and resized to smaller resolutions for GAN training.

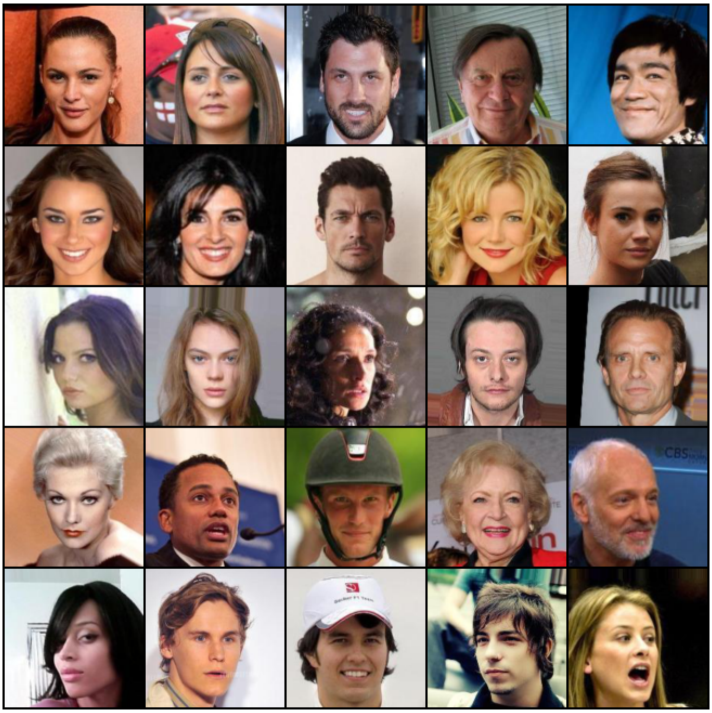


Figure 2.4: A random sample of CelebA training images. The images have been center cropped and resized to 128 x 128 pixels.

Flickr-Faces-HQ (FFHQ). The FFHQ dataset (or *Flickr-Faces-HQ*) was introduced in [Karras et al., 2018] specifically as a GAN benchmark dataset and is another image library of human faces. The dataset contains 70,000 high-resolution square images of 1024 x 1024 pixel size. Compared to CelebA, it contains more variation in terms of age, ethnicity and image backgrounds, and it also contains more samples of accessories such as sunglasses and hats. It is made and published by NVIDIA Research Projects for non-commercial purposes.

Figure 2.5: A random sample of FFHQ training images. The images have been resized to 128 x 128 pixels.

2.4 Computational setup

The experiments of this thesis were implemented in *PyTorch*[3] and run in single-GPU environments in *Google Colab*[4], most often using Nvidia K80 and T4 GPU models. The U-Net GAN experiments were conducted using the official implementation[5] for [Schönfeld et al., 2020] as its base and the StyleGAN 2 experiments were implemented using a PyTorch port[6] of the official implementation for [Karras et al., 2019] as its base. The official implementation[7] of [Gidel et al., 2018] was used for implementations of the optimizers Optimistic Gradient, ExtraGradient, Optimistic Adam, and ExtraAdam. For calculating FID scores, [Seitzer, 2020] was used.

[3] https://pytorch.org/

[4] https://colab.research.google.com/

[5] https://github.com/boschresearch/unetgan

[6] https://github.com/lucidrains/stylegan2-pytorch

[7] https://github.com/GauthierGidel/Variational-Inequality-GAN

3

Sensitivity to hyperparameter settings

The big problem with GAN training is how fine-tuned the setup has to be to work well. The network architectures and hyperparameters must be carefully crafted to work well on specific training sets, and making small adjustments can break the setup. Furthermore, it's more expensive to tune these things than in for example a regular classification setting, since in GAN training it is all about making two neural networks play well with each other. Say that testing different settings for one deep neural network costs n test runs, testing how different settings for two deep neural networks work in combination costs n^2 which makes the problem much less tractable.

To see if using OGDA and ExtraGradient make GAN training less dependent on specific hyperparameter choices, we test several different learning rates for both the generator and discriminator and compare the performances.

The training data chosen for this experiment is 32 by 32 resolution CelebA and the network architecture is U-Net GAN. For hyperparameters, the batch size is set to 24 and the latent vector dimension to 128. The loss function is the U-Net loss. For the Adam-algorithms, $\beta_1 = 0.5$ and $\beta_2 = 0.9$ are used for both the generator and discriminator.

For each optimization scheme, we test the 6 different learning rates $5 \cdot 10^{-2}$, $2 \cdot 10^{-2}$, $5 \cdot 10^{-3}$, $2 \cdot 10^{-3}$, $5 \cdot 10^{-4}$ and $2 \cdot 10^{-4}$ for both the discriminator and generator. This means that we test 36 different learning rate settings for each scheme. For the Adam-variants the learning rates are $5 \cdot 10^{-3}$, $2 \cdot 10^{-3}$, $5 \cdot 10^{-4}$, $2 \cdot 10^{-4}$, $5 \cdot 10^{-5}$ and $2 \cdot 10^{-5}$ since it was found in preliminary testing that this range is more suitable.

Each setting is evaluated on its FID score at the end of the run, calculated with the last pool layer in InceptionNet-v3. The input into the InceptionNet is upscaled from 32 by 32 to 75 by 75 to be compatible with the network.

Each run is performed by running Algorithm 2 with the specified optimization scheme and hyperparameter settings for 20 000 iterations. The goal of each run is not to train a good model but to see early on in training how well the algorithm does.

It is assumed that the early results will correlate well with performance on longer training.

The computational setup is described in Chapter 2.4. Each run of 20 000 iterations took approximately 1 hour to complete on an Nvidia K80 GPU with no significant difference between different algorithms' run times on the same hardware. This gives a GPU runtime of 216 hours, or 9 days, to populate Tables 3.1-3.6 with FID scores[1].

Table 3.1: FID scores from the learning rate sweep of **SGDA**. The best FID score 336.5 was achieved with the discriminator learning rate $5 \cdot 10^{-3}$ and the generator learning rate $2 \cdot 10^{-2}$.

		Discriminator Learning Rate					
		$5 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$5 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$
Generator Learning Rate	$5 \cdot 10^{-2}$	1117.4	787.7	452.1	437.3	910.4	755.1
	$2 \cdot 10^{-2}$	1120.5	540.9	$\boxed{336.5}$	400.5	711.3	876.2
	$5 \cdot 10^{-3}$	1092.5	362.6	357.8	408.1	581.7	812.5
	$2 \cdot 10^{-3}$	1103.1	365.5	338.1	360.8	771.4	712.7
	$5 \cdot 10^{-4}$	1110.1	639.1	411.5	477.8	1025.2	924.3
	$2 \cdot 10^{-4}$	1111.6	774.8	497.9	586.8	813.9	1011.7

[1] This amounts to approximately 64.8kWh of energy consumption, using Nvidia K80's maximum input power of 300W (https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/Tesla-K80-BoardSpec-07317-001-v05.pdf)

Table 3.2: FID scores from the learning rate sweep of **OGDA**. The best FID score 311.6 was achieved with the discriminator learning rate $5 \cdot 10^{-3}$ and the generator learning rate $5 \cdot 10^{-3}$.

		Discriminator Learning Rate					
		$5 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$5 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$
Generator Learning Rate	$5 \cdot 10^{-2}$	1164.2	1005.5	543.3	389.5	714.8	787.1
	$2 \cdot 10^{-2}$	1129.4	909.5	361.7	473.3	596.7	793.9
	$5 \cdot 10^{-3}$	1115.3	805.5	$\boxed{311.6}$	434.2	577.9	915.4
	$2 \cdot 10^{-3}$	1110.2	834.3	337.2	420.6	791.0	813.4
	$5 \cdot 10^{-4}$	1116.2	926.8	424.4	543.6	743.7	925.0
	$2 \cdot 10^{-4}$	1121.7	998.8	586.1	640.6	796.6	1016.4

Table 3.3: FID scores from the learning rate sweep of **ExtraGradient**. The best FID score 514.2 was achieved with the discriminator learning rate $5 \cdot 10^{-3}$ and the generator learning rate $2 \cdot 10^{-2}$.

		Discriminator Learning Rate					
		$5 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$5 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$
Generator Learning Rate	$5 \cdot 10^{-2}$	3775.7	945.1	554.9	717.2	717.0	1024.7
	$2 \cdot 10^{-2}$	1272.7	1446.3	$\boxed{514.2}$	637.4	809.4	956.3
	$5 \cdot 10^{-3}$	1201.2	3644.8	527.1	572.8	814.3	901.5
	$2 \cdot 10^{-3}$	3379.1	1612.9	1792.7	716.8	851.1	883.4
	$5 \cdot 10^{-4}$	1336.7	2276.0	1615.3	831.3	980.9	1064.4
	$2 \cdot 10^{-4}$	2737.0	1843.2	1379.0	1237.6	1077.4	1104.6

Table 3.4: FID scores from the learning rate sweep of **Adam**. The best FID score 171.3 was achieved with the discriminator learning rate $5 \cdot 10^{-3}$ and the generator learning rate $2 \cdot 10^{-5}$.

		Discriminator Learning Rate					
		$5 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
Generator Learning Rate	$5 \cdot 10^{-3}$	1238.4	1175.7	961.1	954.0	922.3	1517.2
	$2 \cdot 10^{-3}$	1272.3	1172.9	949.0	790.5	1116.5	1070.2
	$5 \cdot 10^{-4}$	1050.3	1048.4	679.0	843.5	1361.2	1271.1
	$2 \cdot 10^{-4}$	1483.4	1061.4	1193.0	852.6	1220.6	1141.7
	$5 \cdot 10^{-5}$	207.7	186.7	183.2	188.1	539.3	674.6
	$2 \cdot 10^{-5}$	$\boxed{171.3}$	198.0	178.7	178.1	258.4	397.6

Table 3.5: FID scores from the learning rate sweep of **Optimistic Adam**. The best FID score 157.7 was achieved with the discriminator learning rate $2 \cdot 10^{-4}$ and the generator learning rate $5 \cdot 10^{-5}$.

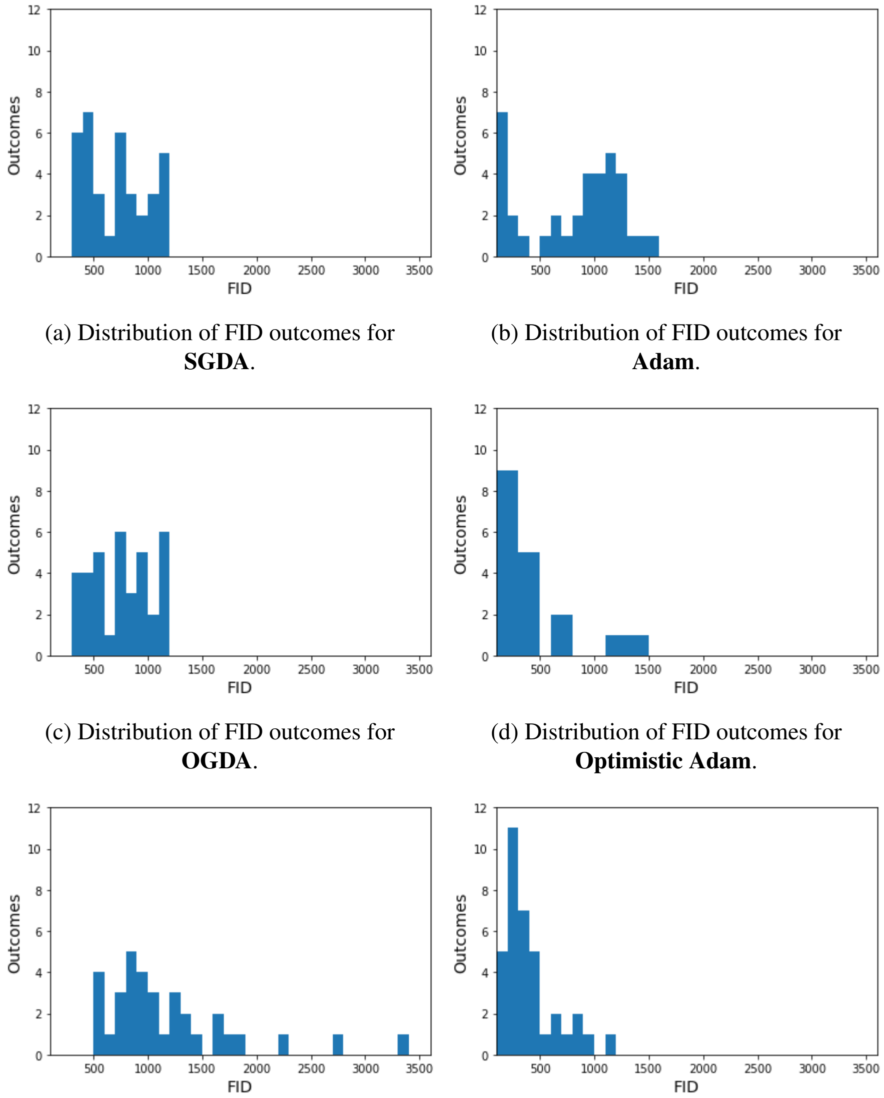
		Discriminator Learning Rate					
		$5 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
Generator Learning Rate	$5 \cdot 10^{-3}$	1118.9	406.3	298.1	342.3	455.2	769.8
	$2 \cdot 10^{-3}$	1239.4	354.5	266.3	328.3	465.6	720.3
	$5 \cdot 10^{-4}$	1403.0	264.1	289.4	239.2	435.1	675.9
	$2 \cdot 10^{-4}$	1303.7	179.1	193.0	190.2	361.6	604.5
	$5 \cdot 10^{-5}$	267.4	187.5	177.3	$\boxed{157.7}$	264.2	424.7
	$2 \cdot 10^{-5}$	205.3	183.4	172.1	179.8	257.8	320.6

Table 3.6: FID scores from the learning rate sweep of **ExtraAdam**. The best FID score 159.7 was achieved with the discriminator learning rate $2 \cdot 10^{-3}$ and the generator learning rate $5 \cdot 10^{-5}$.

		Discriminator Learning Rate					
		$5 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
Generator Learning Rate	$5 \cdot 10^{-3}$	346.5	267.6	427.8	403.0	762.6	1126.8
	$2 \cdot 10^{-3}$	330.7	332.6	301.1	419.4	823.4	977.7
	$5 \cdot 10^{-4}$	221.1	260.8	292.0	313.5	657.6	888.0
	$2 \cdot 10^{-4}$	228.4	192.1	217.7	242.7	552.2	665.4
	$5 \cdot 10^{-5}$	183.3	$\boxed{159.7}$	190.1	202.0	349.2	427.4
	$2 \cdot 10^{-5}$	237.5	212.6	192.3	220.3	316.0	411.8

The results from each run are presented in Tables 3.1-3.6. In general, the Adam-variants outperform the non-Adam-variants by achieving lower FID scores, and furthermore, Optimistic Adam and ExtraAdam achieve lower FID scores than Adam. The same results are presented as histograms in Figure 3.1 for easier interpretation.

Figure 3.1 show the distributions of the outcomes for all algorithms. The non-Adam ExtraGradient performs surprisingly poorly with a lot of high-scoring outcome. The clear best performers are Optimistic Adam and ExtraAdam with very consistently good outcomes for most settings compared to the other algorithms.



(a) Distribution of FID outcomes for **SGDA**.

(b) Distribution of FID outcomes for **Adam**.

(c) Distribution of FID outcomes for **OGDA**.

(d) Distribution of FID outcomes for **Optimistic Adam**.

(e) Distribution of FID outcomes for **ExtraGradient**.

(f) Distribution of FID outcomes for **ExtraAdam**.

Figure 3.1: Histograms of the FID outcomes reported in Tables 3.1-3.6.

Table 3.7: Key statistics for the algorithms’ final outcomes.

Algorithm	Best	Median	Mean	Standard deviation
SGDA	336.5	712.0	697.1	270.6
OGDA	311.6	792.4	754.8	258.0
ExtraGradient	514.2	1044.5	1326.4	837.8
Adam	171.3	951.5	825.2	430.2
Optimistic Adam	$\boxed{157.7}$	$\boxed{309.3}$	436.1	333.2
ExtraAdam	159.7	314.8	$\boxed{398.7}$	$\boxed{244.3}$

The key statistics presented in Table 3.7 tell the same story. Optimistic Adam and ExtraAdam show a consistency across most parameter settings that the other algorithms do not. Adam has some good outcomes but on average is a lot worse.

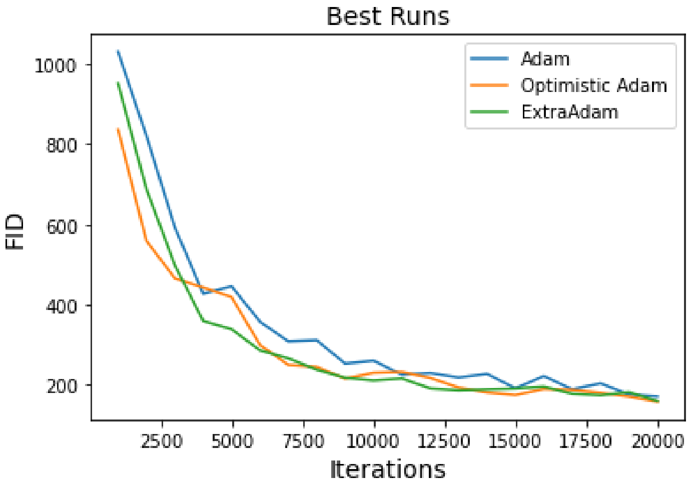


Figure 3.2: FID score over training time for the best performing settings of Adam, Optimistic Adam and ExtraAdam.

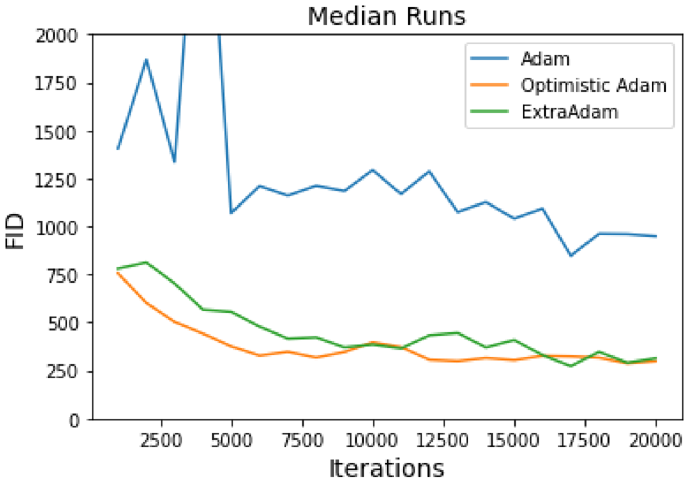
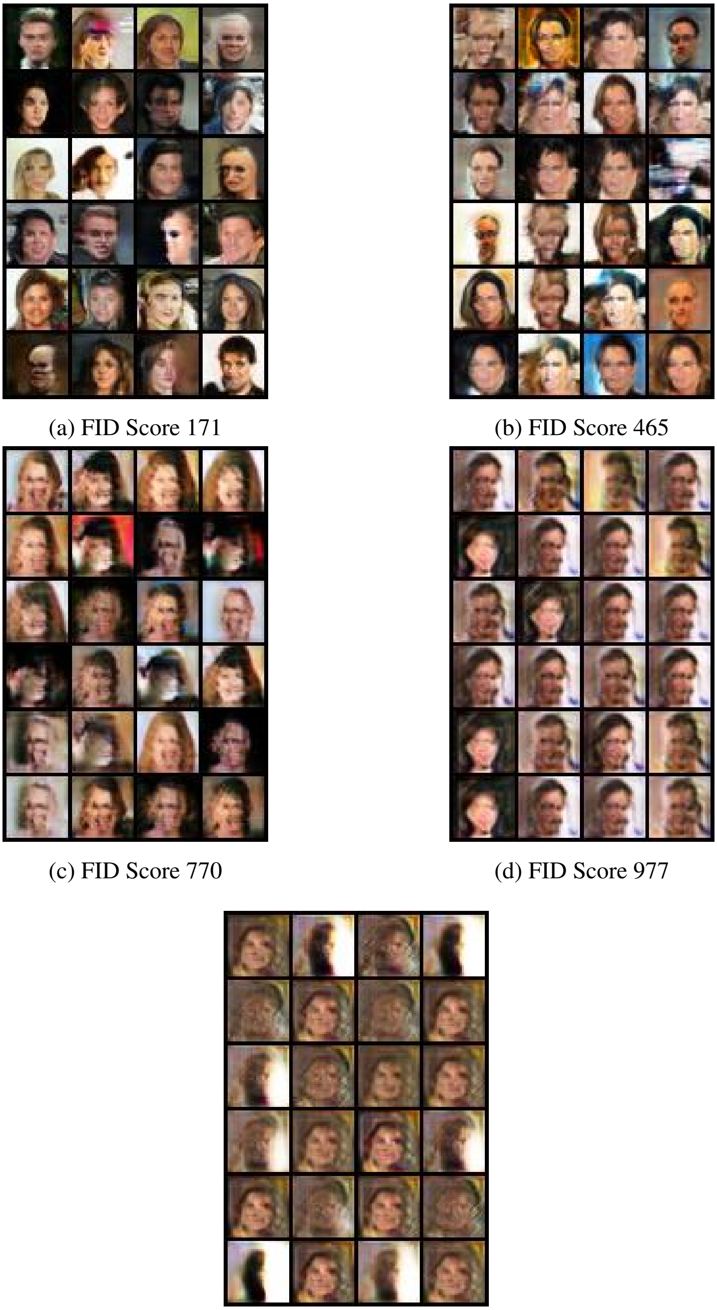


Figure 3.3: FID score over training time for the median performing settings of Adam, Optimistic Adam and ExtraAdam.

To illustrate the difference between Adam, Optimistic Adam, and ExtraAdam, we can observe the best runs in Figure 3.2 and the median runs in Figure 3.3. With their best setting, they all perform relatively similarly but Adam's median performance is much worse than the other two's. The results suggest that Optimistic Adam and ExtraAdam are less sensitive to hyperparameter settings and more robust than Adam.

The results also point to the importance of including the Adam scheme in OGDA and EG which improved both their best and average performances very significantly.

For context on the FID score outcomes, Figure 3.4 is provided to illustrate samples representing different FID scores. Since the training was stopped early for all algorithms, none of the outcomes are especially impressive. What can be seen however is that generators with FID scores of around and above 770 show little diversity and fidelity and basically look like they have collapsed into a failure mode. Generators with FID score around and below 465 have started learning diverse features already and are likely to improve further during continued training.



(a) FID Score 171

(b) FID Score 465

(c) FID Score 770

(d) FID Score 977

(e) FID Score 1125

Figure 3.4: Samples from generators that scored the noted FID scores.

4

Stability

GAN training can collapse, which means that the generator's performance deteriorates, often quickly and suddenly. Collapse can happen after a successful start of the training process and after significant progress has been made to improve the score of the evaluation metric. Due to the stochasticity involved in the training process (weight initialization, minibatch sampling and random latent vector sampling), some GAN training setups will collapse randomly, meaning sometimes they collapse, sometimes they do not and it is possible to have a "lucky run". Setups like this are called unstable.

A specific example of an unstable GAN setup is BigGAN on the FFHQ dataset. It is known to collapse frequently when using Adam or other common optimization schemes. In this report, we run the same setup with ExtraAdam and Optimistic Adam to see if they could alleviate this problem.

The training data chosen for this experiment is therefore 64 by 64 resolution FFHQ and the network architecture is U-Net GAN, which like BigGAN is unstable on this dataset. For hyperparameters, the batch size is set to 24 and the latent vector dimension to 128. The loss function is the U-Net loss.

Each algorithm is run five times for 30 000 iterations on the same settings, only changing the random seed each run for the stochasticity. The algorithms use the same five random seeds, meaning they use the same initial weights and are fed the same training data and random latent vectors. Only the Adam-variants are considered since they were shown in Chapter 3 to perform better.

Results are reported by FID score evaluated every 2000 iterations, calculated with the last pool layer in InceptionNet-v3. The input into the InceptionNet is upscaled from 64 by 64 to 299 by 299 to have the same resolution as the InceptionNet's training data.

The computational setup is described in Chapter 2.4. Each run of 30 000 iterations took approximately 5 hours to complete on an Nvidia K80 GPU with no significant difference between different algorithms' run times on the same hardware.

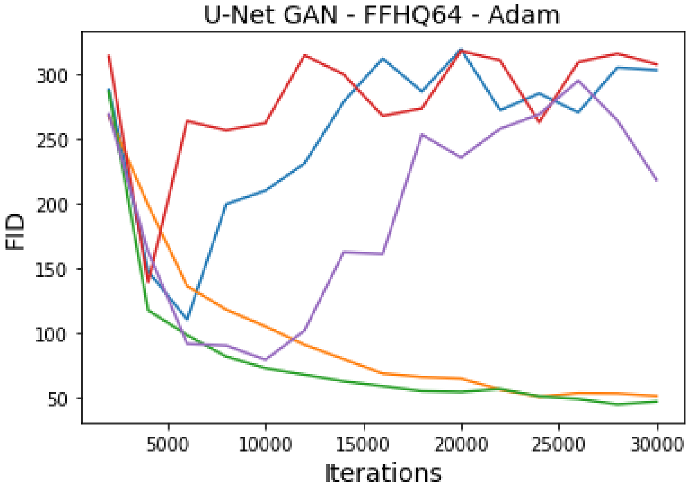


Figure 4.1: FID score over training time for 5 different random seeds using Adam. 3 out of 5 runs collapse and their results deteriorate mid-training.

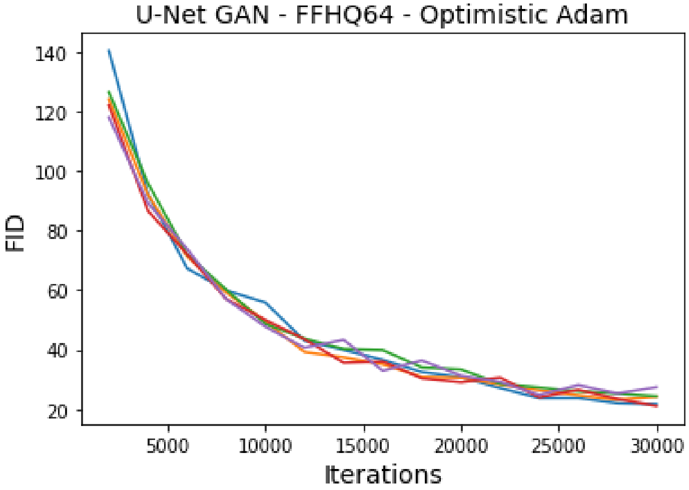


Figure 4.2: FID score over training time for 5 different random seeds using Optimistic Adam. All 5 runs improve their FID score during the full duration of training and none collapse.

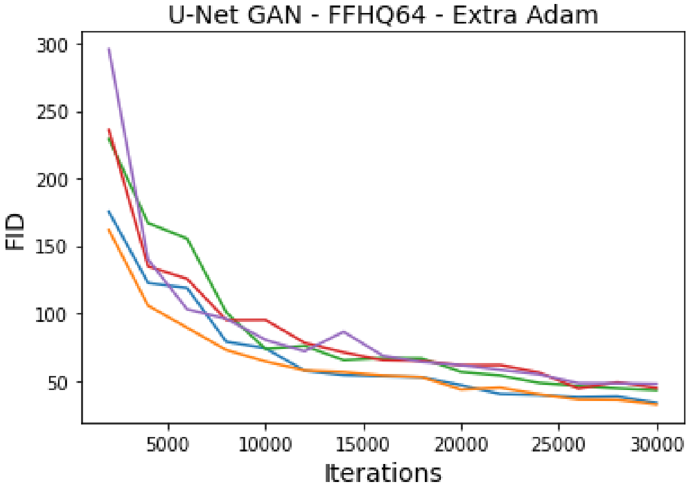


Figure 4.3: FID score over training time for 5 different random seeds using ExtraAdam. Like for Optimistic Adam, all 5 runs improve their FID score during the full duration of training and none collapse.

The results are reported in Figures 4.1-4.3. The results show Adam being unstable, failing to train 3 out of 5 times. Meanwhile Optimistic Adam and ExtraAdam succeed on all of their 5 runs. This strongly indicates that Optimistic Adam and ExtraAdam are superior in this regard and can alleviate stability issues in practical GAN training.

We also note that on this task, Optimistic Adam was more consistent during training and achieved slightly better final FID scores than ExtraAdam on this task.

5

Performance

StyleGAN 2 is already stable and produces amazing results using its well tuned standard optimization option Adam. We will evaluate if Optimistic Adam and ExtraAdam can improve its results even more.

The training data chosen for this experiment is therefore 128 by 128 resolution FFHQ and the network architecture is, as mentioned, StyleGAN 2. For hyperparameters, the batch size is set to 32 and the latent vector dimension to 512. The loss function is the WGAN-GP. Exponential averaging is also evaluated and the decay parameter is $\beta = 0.995$.

Adam, Optimistic Adam and ExtraAdam are evaluated and are each run for 100 000 iterations.

Results are reported by FID score evaluated every 2000 iterations, calculated with the last pool layer in InceptionNet-v3. The input into the InceptionNet is upscaled from 128 by 128 to 299 by 299 to have the same resolution as the InceptionNet's training data.

The computational setup is described in Chapter 2.4. Each run of 100 000 iterations took approximately 17 hours to complete on an Nvidia K80 GPU with no significant difference between different algorithms' run times on the same hardware.

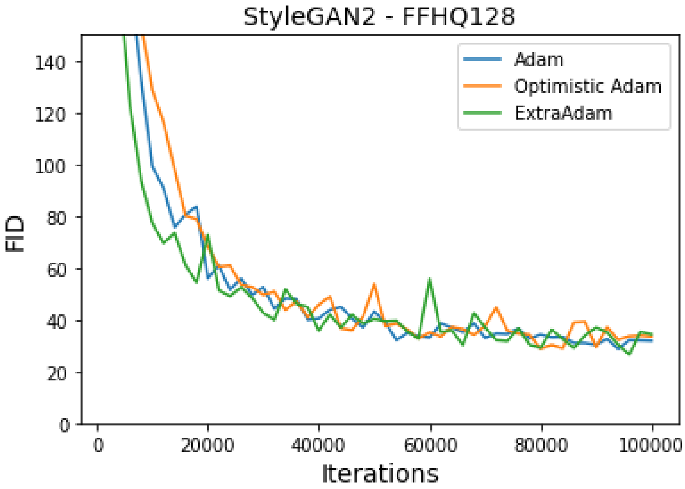


Figure 5.1: FID score over training time for Adam, Optimistic Adam, and ExtraAdam applied to the StyleGAN 2 architecture. All algorithms perform similarly.

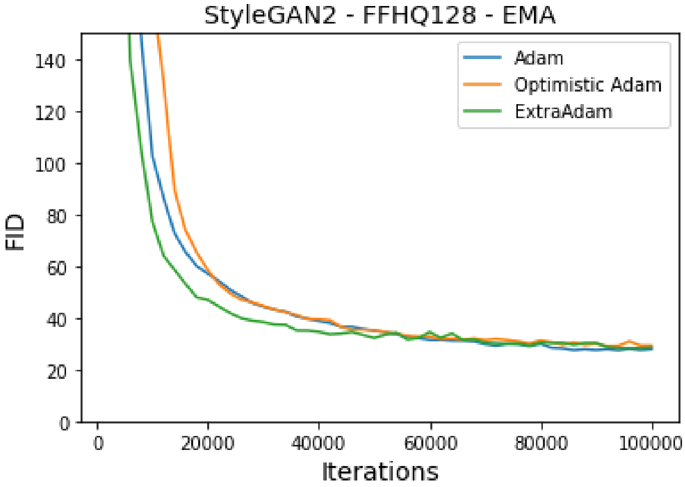


Figure 5.2: FID score over training time for Adam, Optimistic Adam, and ExtraAdam applied to the StyleGAN 2 architecture using exponential moving average (EMA) with $\beta = 0.995$. All algorithms perform similarly. Each algorithm's training progress is smoother and less volatile compared to when EMA was not used (Figure 5.1).

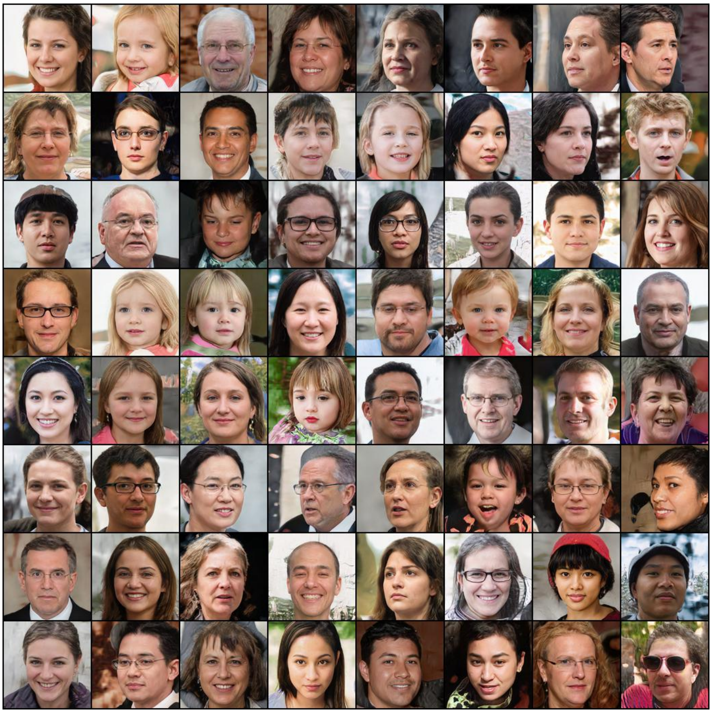


Figure 5.3: Generated samples after training StyleGAN 2 for 100k iterations with Adam.

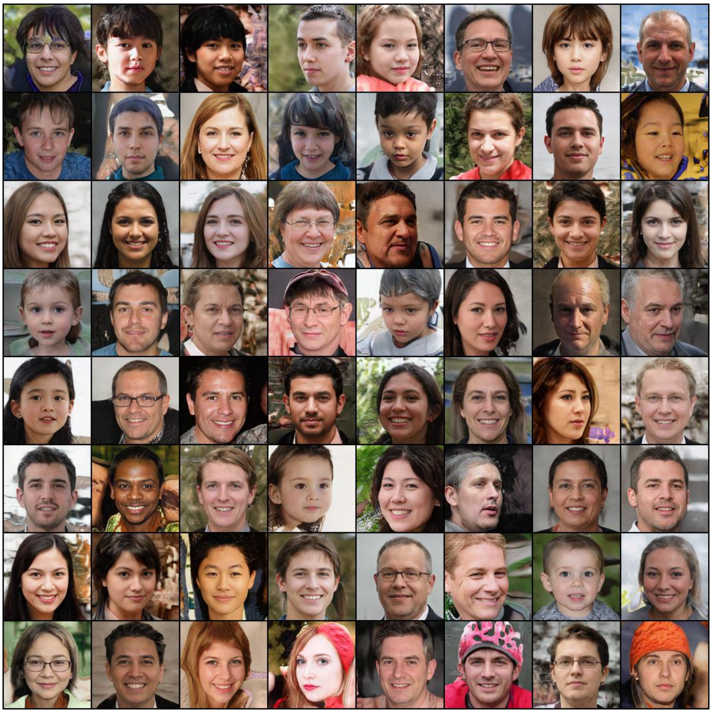


Figure 5.4: Generated samples after training StyleGAN 2 for 100k iterations with Optimistic Adam.

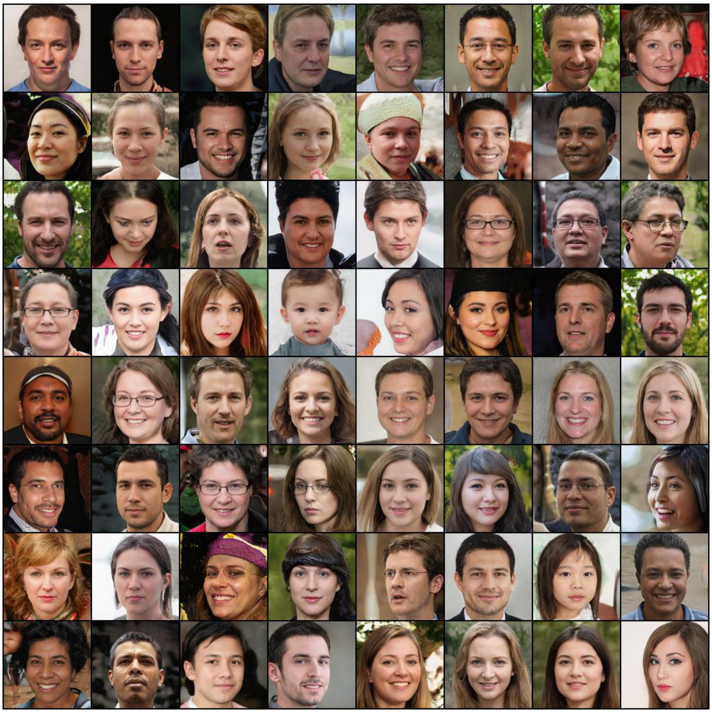


Figure 5.5: Generated samples after training StyleGAN 2 for 100k iterations with ExtraAdam.

In Figures 5.3-5.5, samples from the trained models are displayed. All algorithms trained great models and there are no notable differences between the algorithms' outcomes that can be seen by visual inspection.

Table 5.1: Best achieved FID scores during training for the different algorithms.

Algorithm	No Averaging	EMA
Adam	28.7	27.4
Optimistic Adam	28.7	29.0
ExtraAdam	26.4	28.3

In Table 5.1 and Figures 5.1-5.2, the FID scores of the algorithms' training are reported. It can be noted that averaging did not help significantly except for stability during training. It is possible that the decay parameter β would have needed to be larger for it to have a larger impact on the results. Overall, it did not seem like Optimistic Adam and ExtraAdam had a notable positive effect on the results either and they performed very similar to Adam.

Nearest neighbors

To further illustrate the power of these generative models, Figures 5.6-5.7 show the nearest neighbors of randomly sampled generated images in the training set using two different metrics, pixelwise L^2 and InceptionNet-activations L^2 distances. It is clear that no memorization of training data is occurring and the generators are generating novel data.

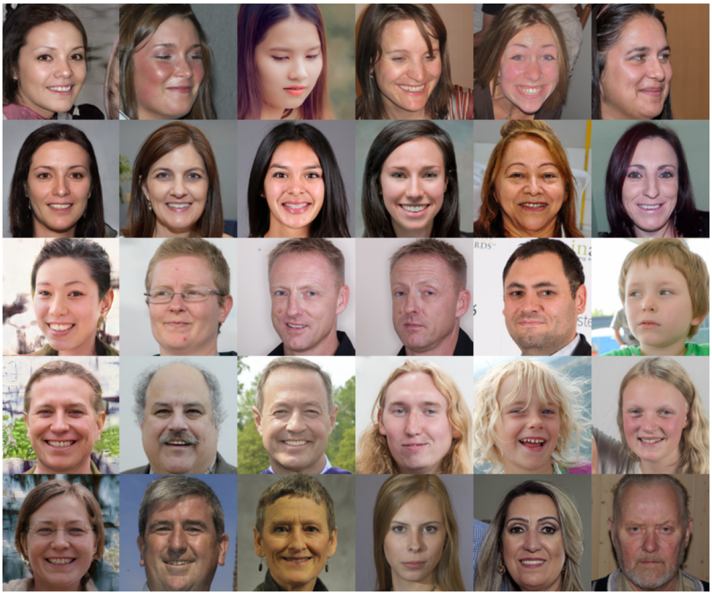


Figure 5.6: L^2 nearest neighbors in the training set of generated samples. The left-most column are 5 generated samples, with their corresponding rows being their nearest neighbors in the training set in increasing distance from left to right.

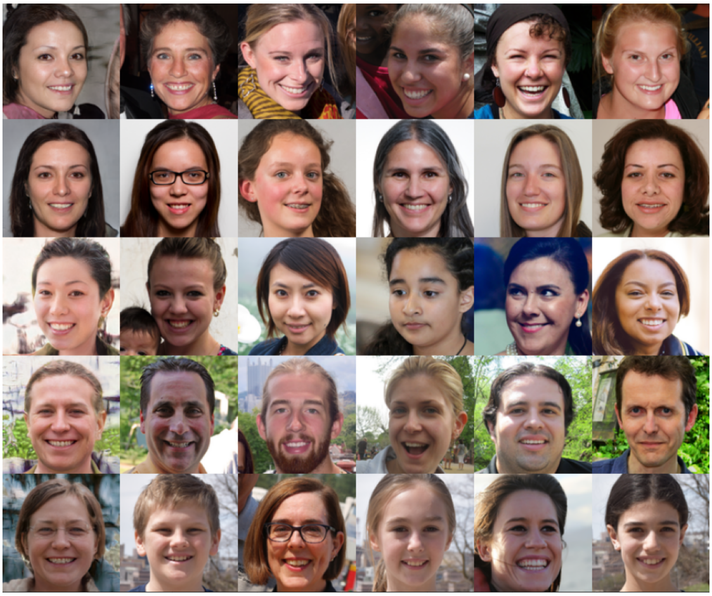


Figure 5.7: InceptionNet-activations nearest neighbors in the training set of generated samples). The left-most column are 5 generated samples, with their corresponding rows being their nearest neighbors in the training set in increasing distance from left to right. The Inception-activations used are the same as for the FID calculations.

6

Conclusions

The results from the sensitivity study strongly indicate that using Adam as the basic algorithm is important to successful GAN training, as the non-Adam-variants did not perform well comparatively.

Furthermore, the results strongly indicate that Optimistic Adam and ExtraAdam are more robust to varying choices of hyperparameter settings and are less prone to collapse mid-training than Adam. However, for already fine-tuned, well working Adam setups like StyleGAN 2, end-results were not improved by using these variants, indicating that they are more reliable and easier to work with but not necessary for achieving better results.

Bibliography

Araujo, A., W. Norris, and J. Sim (2019). “Computing receptive fields of convolutional neural networks”. *Distill*. https://distill.pub/2019/computing-receptive-fields. DOI: 10.23915/distill.00021.

Arjovsky, M., S. Chintala, and L. Bottou (2017). *Wasserstein gan*. arXiv: 1701.07875 [stat.ML].

Brock, A., J. Donahue, and K. Simonyan (2018). “Large scale GAN training for high fidelity natural image synthesis”. *CoRR* **abs/1809.11096**. arXiv: 1809.11096. URL: http://arxiv.org/abs/1809.11096.

Daskalakis, C., A. Ilyas, V. Syrgkanis, and H. Zeng (2017). “Training gans with optimism”. *CoRR* **abs/1711.00141**. arXiv: 1711.00141. URL: http://arxiv.org/abs/1711.00141.

Gidel, G., H. Berard, P. Vincent, and S. Lacoste-Julien (2018). “A variational inequality perspective on generative adversarial nets”. *CoRR* **abs/1802.10551**. arXiv: 1802.10551. URL: http://arxiv.org/abs/1802.10551.

Goodfellow, I. J. (2017). “NIPS 2016 tutorial: generative adversarial networks”. *CoRR* **abs/1701.00160**. arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). *Generative adversarial networks*. arXiv: 1406.2661 [stat.ML].

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville (2017). “Improved training of wasserstein gans”. *CoRR* **abs/1704.00028**. arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

Hurwitz, A. (1895). “Ueber die bedingungen, unter welchen eine gleichung nur wurzeln mit negativen reellen theilen besitzt. english translation: “on the conditions under which an equation has only roots with negative real parts””. *Mathematische Annalen* **46**, pp. 273–284.

Karras, T., T. Aila, S. Laine, and J. Lehtinen (2017). "Progressive growing of gans for improved quality, stability, and variation". *CoRR* **abs/1710.10196**. arXiv: `1710.10196`. URL: `http://arxiv.org/abs/1710.10196`.

Karras, T., S. Laine, and T. Aila (2018). "A style-based generator architecture for generative adversarial networks". *CoRR* **abs/1812.04948**. arXiv: `1812.04948`. URL: `http://arxiv.org/abs/1812.04948`.

Karras, T., S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila (2019). "Analyzing and improving the image quality of stylegan". *CoRR* **abs/1912.04958**. arXiv: `1912.04958`. URL: `http://arxiv.org/abs/1912.04958`.

Kingma, D. P. and J. Ba (2017). *Adam: a method for stochastic optimization*. arXiv: `1412.6980 [cs.LG]`.

Korpelevich, G. M. (1976). "The extragradient method for finding saddle points and other problems". *Ekonomika i Matematicheskie Metody, Vol. 12, No. 4, pp. 747-756*.

Liu, Z., P. Luo, X. Wang, and X. Tang (2015). "Deep learning face attributes in the wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*.

Malitsky, Y. and M. K. Tam (2020). *A forward-backward splitting method for monotone inclusions without cocoercivity*. arXiv: `1808.04162 [math.OC]`.

Mokhtari, A., A. Ozdaglar, and S. Pattathil (2019). *A unified analysis of extragradient and optimistic gradient methods for saddle point problems: proximal point approach*. arXiv: `1901.08511 [math.OC]`.

Radford, A., L. Metz, and S. Chintala (2015). *Unsupervised representation learning with deep convolutional generative adversarial networks*. URL: `http://arxiv.org/abs/1511.06434`.

Ronneberger, O., P. Fischer, and T. Brox (2015). "U-net: convolutional networks for biomedical image segmentation". *CoRR* **abs/1505.04597**. arXiv: `1505.04597`. URL: `http://arxiv.org/abs/1505.04597`.

Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen (2016). *Improved Techniques for Training GANs*. arXiv: `1606.03498 [cs.LG]`.

Schönfeld, E., B. Schiele, and A. Khoreva (2020). "A u-net based discriminator for generative adversarial networks". *CoRR* **abs/2002.12655**. arXiv: `2002.12655`. URL: `https://arxiv.org/abs/2002.12655`.

Seitzer, M. (2020). *pytorch-fid: FID Score for PyTorch*. `https://github.com/mseitzer/pytorch-fid`. Version 0.1.1.

Smith, R. A. (1970). "Möbius transformations in stability theory". *Mathematical Proceedings of the Cambridge Philosophical Society* **68**:1, pp. 143–151. DOI: `10.1017/S0305004100001158`.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2015). "Rethinking the inception architecture for computer vision". *CoRR* **abs/1512.00567**. arXiv: `1512.00567`. URL: `http://arxiv.org/abs/1512.00567`.

Tseng, P. (2000). “A modified forward-backward splitting method for maximal monotone mapping”. *Siam Journal on Control and Optimization - SIAM* **38**.

Wang, X., R. Girshick, A. Gupta, and K. He (2018). *Non-local neural networks*. arXiv: `1711.07971 [cs.CV]`.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2021

Document Number
TFRT-6136

Author(s)
Oskar Larsson

Supervisor
Martin Morin, Dept. of Automatic Control, Lund University, Sweden
Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Robustness, Stability and Performance of Optimization Algorithms for GAN Training

Abstract

Training Generative Adversial Networks (GANs) for image synthesis problems is a largely heuristical process that is known to be fickle and difficult to set up reliably. To avoid common failure modes and succeed with GAN training, one needs to find very specific hyperparameter settings carefully tuned to the model architectures and datasets at hand. Standard GAN training optimization methods such as Stochastic Gradient Descent Ascent (SGDA) and Adam do not even converge on some very simple min-max problems, which may in part explain the unreliable results that occur when applying them in the more complicated GAN setting.

This thesis compares training GANs using SGDA and Adam with optimization schemes that do converge in convex-concave min-max settings. Specifically, Optimistic Gradient Descent Ascent (OGDA), the ExtraGradient method, and their Adam-variants are treated. Their robustness, stability and performance are evaluated and compared on the state-of-the-art GAN architectures U-Net GAN and StyleGAN 2.

The empirical results on U-Net GAN strongly indicate that the Adam-variants of OGDA and ExtraGradient are more robust to varying choices of hyperparameter settings and less prone to collapse mid-training than the most commonly used optimization schemes in contemporary GAN training, regular SGDA and Adam. However, for an already fine-tuned, well working Adam setup such as StyleGAN 2, end-results were neither improved nor worsened by using the Adam-variants of OGDA and ExtraGradient. This indicates that these algorithms are more reliable and easier to work with than SGDA and Adam, but not necessary for achieving state-of-the-art performance.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-53

Recipient's notes

Security classification

http://www.control.lth.se/publications/