Study of Optimiser Performance for Wasserstein Loss Function in Generative Adversarial Networks

Anton Hosgood, Fedor Sergeev, Younes Moussaif
EPFL DS, EPFL CSE, EPFL GM
anton.hosgood@epfl.ch, fedor.sergeev@epfl.ch, younes.moussaif@epfl.ch

***Abstract*—Generative adversarial networks (GANs) are a powerful tool for creating synthetic data and has been successfully used for tasks such as high fidelity image generatrion. However, GANs are notoriously difficult to train as the process is unstable, involves a lot of hyperparameters, and there are little to no theoretical guarantees for convergence. In this work, we systematically study how the choice of an optimiser and initial learning rate affects the speed and stability of GAN training with Wasserstein loss on the MNIST dataset. We show that the choice of an appropriate initial learning rate can have a dramatic impact on the end quality of the generated images, while the choice of an appropriate optimiser can help to avoid loss explosions and make the training more stable.**

I. Introduction

GANs are a recent innovation in machine learning proposed by Goodfellow et al. [1]. Two networks, a discriminator and a generator, are trained jointly. The discriminator classifies samples as being ”real” or ”fake”, and the generator maps vectors from a fixed distribution (for example, random vectors from a standard normal distribution) to sample space aiming to fool the discriminator. The approach is adversarial since the two networks have antagonistic objectives.

Training GANs is often more difficult than training standard neural networks. Often one of two undesirable behaviours is observed: oscillations without convergence or mode collapse. Unlike standard loss minimisation, there is no guarantee that the loss will actually decrease as training progresses, which can result in oscillations. Mode collapse, on the other hand, occurs when the generator learns to fool the discriminator by modelling very well a small sub-population, concentrating on a few modes instead of modelling the entire distribution.

One of the factors that drastically affects the stability and speedup of GAN training is the choice of loss function and optimiser. In previous works, different optimisers have been used for different loss functions: while Adam [2] optimiser has been employed in some cases [1], it has also been reported that RMSProp [3] can be a better choice in others [4].

In this work, we study the use of different optimisers (mini-batch SGD, RMSprop, Adam) and various initial learning rates for a fixed dataset and network architecture using the Wasserstein loss.

II. Related works

Improving GAN training speed and stability is an active research topic. Many approaches have been proposed, with most works proposing either modifications to the network architecture or changes to the training procedure or loss function.

Changes to the architecture that seem to improve the training include addition of dropout [5], various types of normalisation layers [6] (batch [7], layer [8], spectral [9]), as well as adding artificial noise to inputs [10] or intermediate layers [11] to name but a few.

The design choices for the training procedure include various loss functions: the original [1], Wasserstein loss with weight clipping [4] and gradient penalty [12], as well as hinge-loss [13].

Large-scale studies have also been conducted where multiple design and training options were compared [6], [14]. It is highlighted that one of the main difficulties in comparing similarly performing models between each other is a lack of definitive metrics for comparing generated datasets between each other.

Evaluation of GAN performance has been a subject of active research. Currently, there is no universally accepted method for evaluating GANs with a variety of metrics proposed, each with its own pros and cons [15]. This is largely due to the difficulties of comparing similarity between two datasets and the variety of high-dimensional datasets, such as colour images. We use the Fréchet inception distance (FID) as well as visual inspection to evaluate the performance of the particular optimiser.

Many modern GAN architectures accept a condition vector as input into the generator and discriminator [16]. This enables some amount of control over what kind of samples the generator produces. In this work, we only consider unconditional GANs for simplicity.

III. Models and Methods

A. Loss functions

The adversarial game between a generator and a discriminator can be defined as: $\min_G \max_D L(D, G)$. The discriminator D given a sample outputs a value in $[0, 1]$ that indicates how realistic it is (0 being fake and 1 being real). While the discriminator aims to better classify the samples, the generator G tries to produce samples that would confuse the discriminator as much as possible and thus look realistic. $L(D, G)$ is defined as

$$L = \mathbb{E}_{x\sim p_r}[\log D(X)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))], \quad (1)$$

where p_r is the data distribution over real sample x, p_z is the noise distribution (for generator inputs).

The idea behind this function is that when the discriminator is optimal, minimising this loss is equivalent to minimising the Jensen-Shannon divergence between the real and synthetic distributions (see section VI-B).

In practice, such a loss function often results in mode collapse and vanishing gradients [1], [4]. Mode collapse is a phenomenon in which the generator produces an output that fools the discriminator, but receives a low loss value from it. Subsequently, the generator starts producing only this output, rendering the model undesirable. Vanishing gradients occur when the discriminator performs well and thus starts producing small gradients that stall the training. On the other hand, if the discriminator performs poorly, its gradients are not informative and do not lead to an increased performance of the generator. Thus, training a GAN with this loss function is often unstable and challenging in practice.

An alternative loss function is the Wasserstein loss

$$L = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(X)] + \mathbb{E}_{z \sim p_r}[f_w(G(z))], \qquad (2)$$

where $f : \mathbb{R} \to \mathbb{R}$.

The main idea here is to minimise the Wasserstein distance between the synthetic and real distributions, instead of the Jensen-Shannon divergence (see section VI-A). It requires, however, the model f to be K-Lipshitz continuous. To enforce this condition, an additional regularisation term called the gradient penalty needs to be introduced [12]. Note that here the discriminator outputs a value in $\mathbb{R}$ rather then in $[0, 1]$ and thus is often called a "critic". We also note that for the Wasserstein loss it is possible to run discriminator training for multiple iterations at each training epoch. This produces discriminators of better quality, which provide a more informative gradient to the generator.

Although it is not perfect, it has been shown that in practise the Wasserstein loss is more stable than the original [12]. Therefore, in this work we only consider the Wasserstein loss.

B. Optimisers

We consider three popular gradient optimisers: mini-batch stochastic gradient descent [17], RMSprop[3], and Adam [2].

Mini-batch stochastic gradient descent (SGD) is the simplest algorithm of the three. Its main idea is to avoid evaluating the mean gradient of the target function on the whole dataset at each optimisation step (like it is done in vanilla gradient descent). Evaluation on the whole dataset is infeasible for large datasets for which the samples and gradients would not fit into the operative memory of the computer. Instead, at each optimisation step, the mean gradient is evaluated over a random subset (also called a mini-batch) of the training set. Thus, mini-batch SGD can be used on large datasets, with the mini-batch size as a hyperparameter to tune which trades off the number of training iterations with the accuracy of the evaluated gradient. An intuition behind this is with a very large dataset, after a certain batch-size gradients may be "accurate enough", and increasing the batch-size will have less of an effect than the increased number of training iterations brought with a smaller batch-size.

The *Root Mean Squared Propagation algorithm (RMSprop)* extends the mini-batch SGD by using an individual adaptive learning rate for each parameter of the target function. Since the magnitude of partial derivatives can vary between parameters of the target function, using adaptive individual learning rates often speeds up training. In RMSprop, an exponentially decaying average of the squared gradient for each weight from previous time steps is used, at each optimisation step the total gradient is divided by this value.

Adaptive Moment Estimation (Adam) algorithm also uses adaptive individual learning rates for each parameter but, unlike RMSprop, it also adjusting the learning rate by using the average of the gradients themselves. This modification essentially means that the optimiser keeps track of the "momentum" of the gradient, helping avoid local minima.

A detailed description of the algorithms can be found in section VI-B.

C. Evaluation

Performance of a GAN can be evaluated using a variety of methods [15]. Usually they amount to judging how similar a set of generated samples is to a set of real samples from the initial dataset.

If the samples are images, the similarity and variety of synthetic data can be assessed visually. This approach also supports the identification of particular features that differentiate synthetic and real images. While such analysis is not rigorous, it helps to identify shortcomings of the model and understand its causes. For example, checkerboard artifacts [18] in the image suggest that the architecture of the generator should be changed.

A more rigorous approach would be to use some sort of metric to evaluate the quality of the generated images in a quantitative way. *Fréchet Inception Distance (FID)* is one of the most widely used metrics for evaluating the quality of generated images [19], [15]. FID is calculated as a squared Wasserstein distance between the distribution of features of some neural network on synthetic data and on real data. Usually, a powerful convolutional neural network trained on a large dataset of images is used (normally the Inception v3 network trained on the ShapeNet dataset [20], [21]). Since such a network performs very well in image classification tasks, its "perception" of similarity can be interpreted as comparable to human perception.

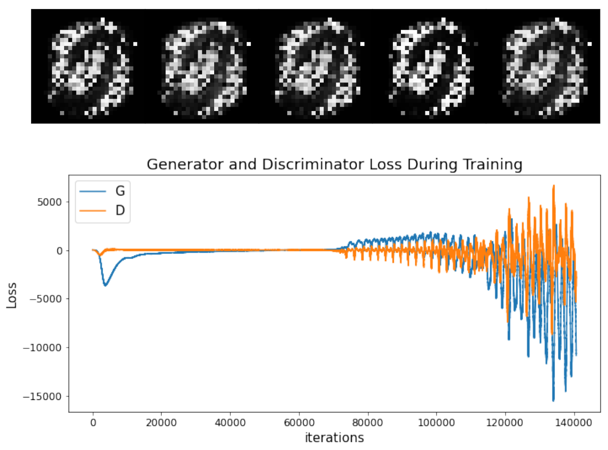
Although Inception was trained on CIFAR10, we choose to use FID with our dataset, since we will only use it to compare results from a GAN that is trained on MNIST, and not with other GANs that have been trained on CIFAR10. For our measurements, we broadcast the greyscale channel of the MNIST samples to the 3 channels that are expected by the Inception V3 model to evaluate the FID. To evalute the FID score, we randomly select 1000 samples from the MNIST training set along with 1000 samples from the generator. The activation statistics in batches of size 32 are then calculated and averaged to evaluate the FID.

IV. Experiments and Results

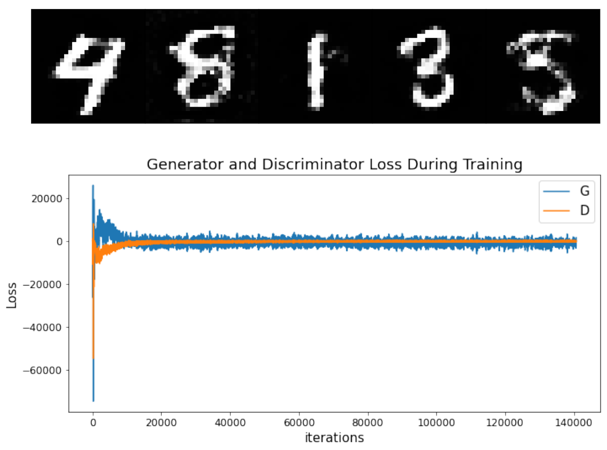
In this study we compare the performance of different gradient based optimisers for the Wasserstein loss with gradient penalty.

The models are trained on the MNIST dataset that contains 60,000 samples (we use only the training set of MNIST) of 28×28 greyscale images of handwritten digits [22]. We consider this particular dataset because it is well known (so the results can be easily compared to those from the literature) and the samples are not very large (we use Google Colaboratory and thus our computational resources are limited).

For all experiments, we use a simple fully connected architecture for both the generator and the discriminator that



(a) Mode collapse (initial learning rate 10^{-6})



(b) Successful training (initial learning rate 10^{-4})

Figure 1: Generator outputs at the end of training and loss profiles for Wasserstein loss and Adam.

is similar to [1] (see section VI-C for detailed description). For simplicity, we consider only unconditional GANs.

We implement the models in Python using PyTorch [23]. The source code is available at github.com/TheodorSergeev/optml_gan.

We compare performance of the Wasserstein GAN loss used with either SGD, RMSprop and Adam optimisers with various learning rates. To evaluate the performance, we compare sets of generated images visually (judging how realistic and varied the synthetic samples are), as well as by using the FID score.

The comparison of the FID values for the final model is presented in table I.

Notice that for large learning rates, mini-batch SGD training breaks down, while RMSprop and Adam are still able to train. This means that an adaptive learning rate stabilises the training by adjusting each parameter of the model separately.

This effect can be especially useful, when the training process is optimised with respect to other parameters such as the network architecture or loss function.

Learning Rate	FID		
	Adam	SGD	RMSprop
1e-1	281.94	N/A	302.97
1e-2	365.71	N/A	319,93
1e-3	90.54	N/A	50.79
1e-4	**44.93**	**29.57**	**46.31**
1e-5	111.33	97.26	318.54
1e-6	230.87	131.53	203.86
1e-7	295.93	361.85	359.73

Table I: Evaluation of the GAN's performance for different optimisers and learning rates via FID (lower is better). N/A indicates that the training was abruptly stopped because its loss values exploded (grew to infinity).

Interestingly, all optimisers perform best when used with the same learning rate of 10^{-4}. We see many instances of mode collapse, especially for smaller learning rates fig. 1. For the larger learning rates there is often no convergence: loss profiles are not smooth, the gradients seem to be uninformative, and the generated pictures are just black.

Visual comparison of the generated images allows us to judge how informative the FID metric is. For FID values above 200, the generated pictures usually show mode collapse (same unrealistic picture) or lack of convergence (noise or black picture). The values of 100 to 200 indicate that the network generates symbol-like shapes that still do not quite resemble the digits. If the FID value is less than 100, the network generates digit-like images with occasional issues. Visually we can see a difference of around 20 points of the FID score (see section VI-D2).

V. Conclusion

Generative Adversarial Networks are difficult to train. The combination of high dimensional data, complex loss functions and large amount of hyper parameters present a challenge when designing a training procedure.

By comparing performance of mini-batch SGD, RMSprop and Adam, we showed that optimisers with an adaptive learning rate make the training more stable and thus simplify the search for optimal training parameters. However, the choice of the initial learning rate remains crucial for achieving good performance of the model as adaptive learning rate still depends on its initial value (e.g., Adam [24]). Our findings allow for quicker and more reliable design of a training procedure for Wasserstein GANs.

The main limitation of our study is that the experiments were not performed multiple times, so a mean value of the FID score for different training initialisation is not reported. The training times for the simple linear architectures we used is approximately 2 hours for 300 epochs with a Nvidia-P100 GPU. Therefore, repeating the experiments was not feasible.

In the future this study can be extended to consider multiple loss functions, employ more metrics, and vary secondary parameters of the optimisers (e.g, ρ in RMSprop).

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014, pp. 2672–2680.

[2] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2015.

[3] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop, coursera: Neural networks for machine learning,” University of Toronto, Tech. Rep., 2012.

[4] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *IMCL*, vol. 70, 06–11 Aug 2017, pp. 214–223.

[5] P. Isola, J.-Y. Zhu, T. Zhou, and A. Efros, “Image-to-image translation with conditional adversarial networks,” in *CVPR*, 07 2017, pp. 5967–5976.

[6] K. Kurach, M. Lučić, X. Zhai, M. Michalski, and S. Gelly, “A large-scale study on regularization and normalization in gans,” in *ICML*, 2019.

[7] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *ICML*, vol. 37, 07–09 Jul 2015, pp. 448–456.

[8] J. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *ArXiv*, vol. abs/1607.06450, 2016.

[9] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” *ArXiv*, vol. abs/1802.05957, 2018.

[10] M. Arjovsky and L. Bottou, “Towards principled methods for training generative adversarial networks,” in *ICLR*, vol. 1050, 01 2017.

[11] J. Zhao, M. Mathieu, and Y. Lecun, “Energy-based generative adversarial network,” in *ICLR*, 04 2017.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” in *NIPS*, 2017, p. 5769–5779.

[13] I. Kavalerov, W. Czaja, and R. Chellappa, “A multi-class hinge loss for conditional gans,” in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2021, pp. 1290–1299.

[14] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly, “Are gans created equal? a large-scale study,” in *NIPS*, 2018, p. 698–707.

[15] A. Borji, “Pros and cons of GAN evaluation measures,” *CVIU*, vol. 179, pp. 41–65, 2019.

[16] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.

[17] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[18] A. Odena, V. Dumoulin, and C. Olah, “Deconvolution and checkerboard artifacts,” *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[19] 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,” *NIPS*, vol. 30, 2017.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *CVPR*, 2016, pp. 2818–2826.

[21] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, “ShapeNet: An Information-Rich 3D Model Repository,” Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

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

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[24] “PyTorch - Adam optimiser, howpublished = https://pytorch.org/docs/stable/generated/torch.optim.adam.html#adam, note = Accessed: 2022-17-06.”

[25] L. Weng, “From gan to wgan,” *arXiv preprint arXiv:1904.08994*, 2019.

[26] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[27] J. Zhang, “Gradient descent based optimization algorithms for deep learning models training,” *arXiv preprint arXiv:1903.03614*, 2019.

VI. Appendix

A. Distribution Similarity Metrics

Given the probability distribution p and q the metrics are defined as follows [25].

1) Kullback-Leibler:

$$D_{KL}(p,q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx.$$

$D_{KL}(p,q) = 0$ when $p(x) = q(x)$ everywhere.

2) Jensen Shannon:

$$D_{JS}(p,q) = \frac{1}{2} D_{KL}(p,q) + \frac{1}{2} D_{KL}(q,p).$$

Jensen-Shannon divergence is symmetric, unlike Kullback-Leibler divergence, and generally smoother.

3) Wasserstein Distance:

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma} ||x-y||,$$

where $\Pi(p,q)$ is the set of all possible joint probability distributions of p and q.

Wassertstein metric is smoother than Jensen-Shannon and thus provides better gradients (e.g. in the case of non-overlapping discrete distributions).

B. Optimisation Algorithms

Below, we provide the pseudocode for the optimisation algorithms used in this work. The algorithms are adapted from [26] and [27].

Here $f(\boldsymbol{\theta})$ is the stochastic objective function with parameters $\boldsymbol{\theta}$ that is optimised; $\boldsymbol{\theta}_0$ is the initial parameter of the objective function; α is the learning rate (step size); ϵ is a small constant for the numerical stability of the division.

Symbol "$\circ$" denotes element-wise product of two vectors of the same size; function $\text{diag}(\boldsymbol{A})$ outputs a matrix with non-zero values only on the diagonal of the original matrix A.

We assume that $f(\boldsymbol{\theta})$ on every iteration is evaluated on a mini-batch of b samples randomly selected from the dataset. Note that in practise, we do not have an easy way to evaluate convergence of the algorithm, so the optimisation procedure is stopped after a set amount of epochs has elapsed.

Algorithm 1 Mini-batch SGD [3]

```
1: $t \leftarrow 0$ (initialise timestep)
2: while $\boldsymbol{\theta}_t$ not converged do
3:     $t \leftarrow t + 1$
4:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
5:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \boldsymbol{g}_t$
6: end while
7: return $\boldsymbol{\theta}_t$
```

Algorithm 2 RMSprop [3]

```
Require: $\rho \in [0, 1)$: weight of the historically accumulated gradients
1: $\boldsymbol{G}_0 \leftarrow \boldsymbol{0}$ (initialise matrix of accumulated gradients)
2: $t \leftarrow 0$ (initialise timestep)
3: while $\boldsymbol{\theta}_t$ not converged do
4:     $t \leftarrow t + 1$
5:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
6:     $\boldsymbol{G}_t \leftarrow \rho \cdot \boldsymbol{G}_{t-1} + (1 - \rho) \cdot \boldsymbol{g}_t$
7:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \boldsymbol{g}_t \circ \sqrt{\text{diag}(\boldsymbol{G}) + \epsilon \cdot \boldsymbol{I}}^{\,-1}$
8: end while
9: return $\boldsymbol{\theta}_t$
```

Algorithm 3 Adam [2]

```
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates
1: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$ (initialize 1st moment vector)
2: $\boldsymbol{v}_0 \leftarrow \boldsymbol{0}$ (initialize 2nd moment vector)
3: $t \leftarrow 0$ (initialise timestep)
4: while $\boldsymbol{\theta}_t$ not converged do
5:     $t \leftarrow t + 1$
6:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
7:     $\boldsymbol{m}_t \leftarrow b_1 \cdot \boldsymbol{m}_{t-1} + (1 - \beta_1) \cdot \boldsymbol{g}_t$
8:     $\boldsymbol{v}_t \leftarrow b_2 \cdot \boldsymbol{v}_{t-1} + (1 - \beta_2) \cdot \boldsymbol{g}_t^2$
9:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
10:    $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$
11:    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \hat{\boldsymbol{m}}_t \circ (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon)^{-1}$
12: end while
13: return $\boldsymbol{\theta}_t$
```

C. Experiment setup

Size of the noise vector z is 128, batch size is 128. Architectures of the generator and discriminator are pesented below (table II and table III, batch dimension is omitted). Optimisers were used with their default parameters: $\rho = 0.99$ for RMSprop, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for Adam [23]. Wasserstein was used with its recommended parameters: 5 critic iterations for each generator iteration, gradient penalty coefficient is 10 [12].

	Layer	Output shape	Activation
Input	z	128	
Intermediate layers	Linear	256	LReLU(0.2)
	Linear	512	LReLU(0.2)
	Linear	1024	LReLU(0.2)
Last layer	Linear	28×28	tanh

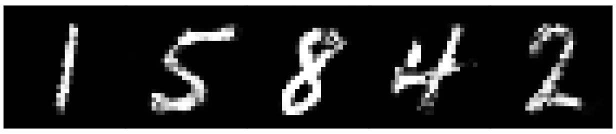
Table II: Generator architecture. Sums to approximately 1.5 million parameters

	Layer	Output shape	Activation
Input	Image	28×28	
Intermediate layers	Linear	1024	LReLU(0.2), Dropout(0.3)
	Linear	512	LReLU(0.2), Dropout(0.3)
	Linear	256	LReLU(0.2), Dropout(0.3)
Last layer	Linear	1	–

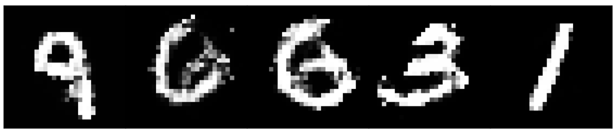
Table III: Discriminator (critic) architecture. Sums to approximately 1.46 million parameters

D. Additional results

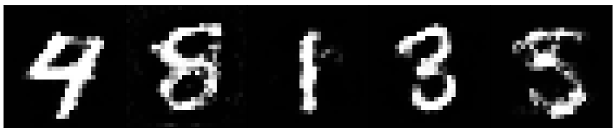
1) Quality of output: Below (fig. 2) we show some images generated by the models with the best FID scores. Notice that while the models trained with the RMSprop and Adam optimisers received the FID score that is 1.5 times worse than that of SGD (table I), visual difference between the samples is not so great. Therefore, it is best to use multiple evaluation strategies (e.g. FID and visual assessment) in case the metric does not behave as expected [15].



(a) SGD (initial learning rate 10^{-4})



(b) RMSprop (initial learning rate 10^{-4})

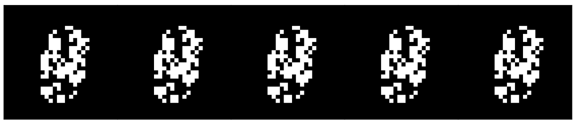


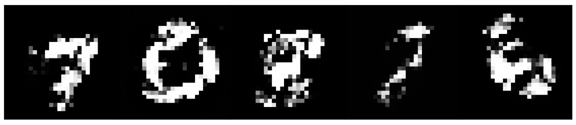
(c) Adam (initial learning rate 10^{-4})

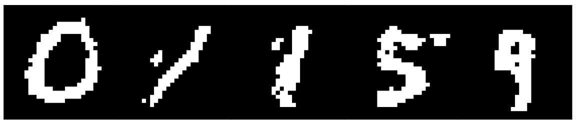
Figure 2: Generator outputs for the models with the best final FID score.

2) Visual quality for different FID values: Visual comparison of samples evaluated in different FID ranges is presented on fig. 3.

3) FID evaluation during training: While loss function profile can signal issues during training(e.g. mode collapse


(a) FID ≈ 280 (initial learning rate 10^{-1})


(b) FID ≈ 110 (initial learning rate 10^{-5})


(c) FID ≈ 90 (initial learning rate 10^{-3})

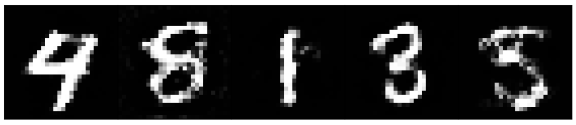
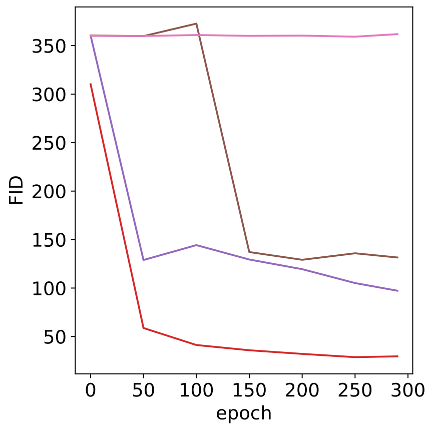
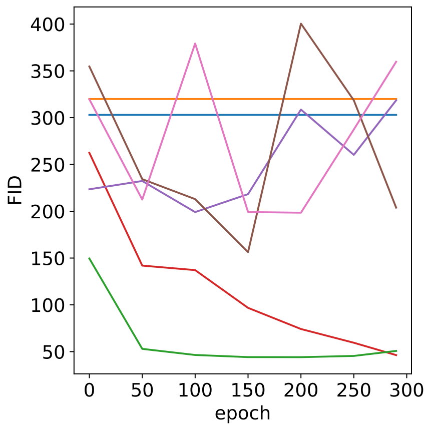

(d) FID ≈ 45 (initial learning rate 10^{-4})

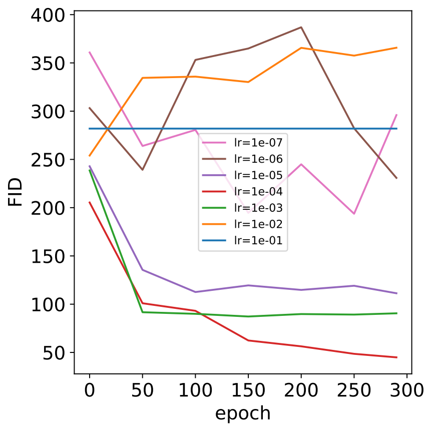
Figure 3: Generator outputs for the models trained with Adam optimiser with FID score in various ranges. Here, the score is evaluated every 50 epochs.



(a) SGD



(b) RMSprop



(c) Adam

Figure 4: FID values during training. Note that SGD was not evaluated on the learning rates $1e-1$, $1e-2$, and $1e-3$ since it's loss values exploded for those learning rates

fig. 1), it is hard to use it to access quality of the current model. We measure FID scores every 50 epochs to understand how of the generator to see how intermediate versions of the generator compare to each other for different training parameters (fig. 4). As we saw from the generated images, learning rates that are too small or too large fail to converge. Thus, when training a GAN, the initial learning rate must be chosen carefully.