Improving GANs using Optimal Transport: Final Report

Chloé Sekkat
ENSAE Paris & ENS Paris-Saclay
chloe.sekkat@ensae.fr

Mathilde Kaploun
ENSAE Paris
mathilde.kaploun@ensae.fr

Thomas Doucet
ENSAE Paris
thomas.doucet@ensae.fr

Abstract

This project was done as final examination of the Optimal Transport course given by Prof. Marco Cuturi at ENSAE Paris. The overall goal is to compare the OT-GAN introduced by [10], which injects Optimal Transport theory into generative models, to other GANs while implementing them from scratch. By doing so, we might assess the overall gain/loss created by the addition of Optimal Transport, both in the theoretical and the practical sides. Our code is available on GitHub [1].

1. Introduction

Generative Adversarial Nets (well-known as GANs) have been introduced by Goodfellow et al. in 2014 [6]. They are models aiming at capturing the underlying structure of the input data so that it can generate new examples that could have been drawn from the input's distribution.

In this report, we present our work on implementing a GAN variant that builds on Optimal Transport, namely the OT-GAN introduced by Salimans et al. in [10]. Section 2 introduces a brief recall on what are generative models and how they work theoretically, highlighting their mathematical properties as well as some well-known practical issues. A focus is made on Vanilla-GAN introduced by Goodfellow et al. [6] as well as DC-GAN introduced by Radford et al. [8]. Section 3 goes over several variants of GANs where Optimal Transport was used (Wasserstein GAN [1], Sinkhorn AutoDiff [5], Cramer GAN [3] and finally OT-GAN [10]). Next, Section 4 presents the various experiments that we did during this project, comparing three GANs variants and Section 5 is a summary of our main results on the MNIST dataset [7].

[1] https://github.com/chloeskt/ot-gan-ensae

2. About GANs

Generative models can be used to generate more data, to be used afterwards by another algorithm for instance. They can be useful in situations where the true data cannot be shared, for anonymity reasons for instance. Sharing a generative model trained on private data could allow the exploitation of the statistical property of this data without sharing the data itself.

Another goal is to use generative modelling to better understand the data. This is based on the hypothesis that a model that successfully learned to generate (and generalize) a dataset should have internally learned some efficient and compressed representation of the information contained in the data. In this case, analysing *a posteriori* the learned representation may give us insights on the data itself.

2.1. Notation

To define more formally the notion of generative models, we need to introduce some notation. Let's call p_D the underlying probability distribution of the data. Our goal is to train a model to represent another probability distribution, p_θ which should approximate p_D.

2.2. Vanilla-GAN

The first GAN was introduced by Goodfellow et al. in [6]. The overall idea is as follows. The architecture of the network is created to model the distribution p_θ as a learned deterministic function applied to a standard noise (to be defined). The sampling process is quite straight-forward: first a noise vector is sampled from a defined distribution: either a standard N-dimensional Gaussian distribution, $\epsilon \sim \mathcal{N}(0; I)$ or a Uniform distribution $\epsilon \sim \mathcal{U}(-I; I))$, then the output is computed as a deterministic function $x = f_\theta(\epsilon)$. The function f_θ is implemented as a neural network, θ representing its learned parameters.

The code idea of GANs is to use another neural network to model the objective: a classifier. The latter, also called discriminator or critic, is trained to distinguish examples from the dataset from examples generated by p_θ. The discriminator D is trained using a classic classifier loss between the two classes defined as the samples generated by either p_D or p_θ (binary cross entropy). This way $D(x)$ can be interpreted as the probability that x came from the real dataset:

$$\mathcal{L}_D = \mathbb{E}_{p_D}\left[-\log D(x)\right] + \mathbb{E}_{p_\theta}\left[-\log\left(1 - D(x)\right)\right] \quad (1)$$

From that, it can be shown that for the generator fixed, the optimal discriminator is given by $D(x) = \frac{p_D(x)}{p_\theta(x)+p_D(x)}$, and when reached, its loss takes a specific value:

$$\mathcal{L}_D = 2\left(\log 2 - JSD(p_\theta \| p_D)\right)$$

where JSD is the JS divergence used to measure the similarity of two distributions.

$$JSD(p_\theta \| p_D) = \frac{1}{2}\mathbb{E}_{p_D}\left[log(\frac{2p_D}{p_D + p_\theta})\right] + \frac{1}{2}\mathbb{E}_{p_\theta}\left[log(\frac{2p_\theta}{p_D + p_\theta})\right] \quad (2)$$

So, training the generator network to maximize the same loss would, assuming the discriminator is always trained up to optimality, minimize the Jensen-Shannon Divergence between p_θ and p_D, and thus bring p_θ closer to p_D. Having the generator trained to maximize $\mathcal{L}_D$ is equivalent to setting its training loss to:

$$\mathcal{L}_G = \mathbb{E}_{p_\theta} \log(1 - D(x)) \quad (3)$$

Overall this is what is called a min-max game, summarized by the following objective:

$$\min_G \max_D \mathcal{L}_{\mathrm{GAN}}(G, D) = \mathbb{E}_{x\sim\mathbb{P}_D}[\log(D(x))] + \mathbb{E}_{z\sim\mathbb{P}_\theta}[\log(1 - D(G(z)))] \quad (4)$$

In [6], Goodfellow et al. show that theoretically, if the networks capacities are unbounded and provided they are trained long enough, the optimal solution $p_D = p_\theta$ is reached. However there are several caveats.

The training of GANs can be extremely complicated and painful. First of all, both the generator and the discriminator must have similar expressive power. Indeed, if the generator is too ”weak”, it will generate images that are far from the underlying data distribution p_D; this will make the task too easy for the discriminator. If the discriminator is too ”strong”, it will reach a loss of ~ 0 since it'll be able to differentiate perfectly the two samples at each iteration. A perfect loss means that the gradients are null, hence there is no more training for the generator. This leads to a vanishing gradient problem. In this case, we will have $JSD(p_\theta||p_D) = \log 2$ and $KL(p_\theta||\, p_D) = \infty$. This is what motives to train the discriminator a fixed number of times between each training iteration of the generator.

Another often-faced issue is the ”mode collapse” one. This occurs when the model produces very similar representations for all masked segments making the problem trivial to solve. The generator finds the single image that makes it impossible for the discriminator to discriminate (i.e. tell whether it is true or fake). It will exploits this failure of the critic and always produce the same image, regardless of the sampled latent vector.

2.3. DCGAN

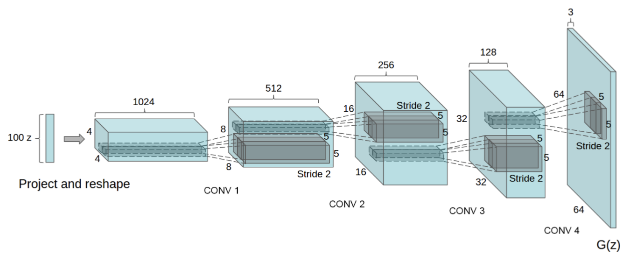


Figure 1. Architecture of the DCGAN generator used to LSUN scene modeling. Extracted from the original DCGAN paper [8]

In 2016, Radford et al. introduced Deep Convolutional Generative Adversarial Networks, also known as DCGANs. Their baseline idea was to help bridge the gap between supervised learning and unsupervised learning for CNNs. Compared to the first version of GANs, DCGAN introduces a series of architecture guidelines. The first main change is the replacement of fully connected layers by upsampling convolutional layers in the generator. All pooling layers are replaced by strided convolutions in the critic and by fractional-strided convolutions in the generator. They use batch normalization and ReLU activation function in all generator's layers except for the output layer which is associated with a Tanh. Finally, they use LeakyReLU for all critic's layers.

Figure 1 depicts the architecture chosen by the Radford et al. for their generator in their LSUN scene modeling experiment. This experiment was used to show how their model scales with more data and higher resolution generation.

3. Optimal Transport in GANs

3.1. Wasserstein GAN

Before [10], other papers have looked into applying Optimal Transport theory in GANs. This is, for instance, the case of the Wassertein GAN paper [1]. Arjovsky et al. propose the Earth-Mover distance or Wasserstein-1 distance as objective for the generator. The underlying intuition is that this distance can be seen as the minimum amount of mass (or pile of dirt) that one has to transport to transform the generator distribution p_θ into p_D, the true data distribution. It writes:

$$W(\mathbb{P}_D, \mathbb{P}_\theta) = \inf_{\gamma \in \Pi(\mathbb{P}_D, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma}[c(x-y)] \tag{5}$$

where c is a cost function (chosen to be the Euclidean distance in the paper) and $\Pi(\mathbb{P}_D, \mathbb{P}_\theta)$ is the set of all joint distribution $\gamma(x,y)$ with marginals $\mathbb{P}_D$ and $\mathbb{P}_\theta$. They prove that for the right choice of c, the latter can be a metric and hence minimizing the Wasserstein-1 distance in $\mathbb{P}_\theta$ becomes a valid objective to find a consistent estimator of $\mathbb{P}_D$ (provided that the generator can model such as distribution).

This is possible due to a series of properties of the Wasserstein-1 distance. First, under mild assumptions on $\mathbb{P}_\theta$, it is both continuous and differentiable. Second, it induces a weaker topology and, as a consequence, allows for easier convergence. This property of convergence is very nice because it means that Wassertein GANs can, theoretically, approximate up to a constant the Wassertein-1 distance through the loss. The latter is then directly linked to the quality of the generated samples.

However, Equation 5 is often untractable and as such we cannot directly optimize on it. Arjovsky at al. [1] propose to focus, as often in optimization field, on the dual through the Kantorovich duality:

$$W(\mathbb{P}_D, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_D}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{6}$$

where now the minimization over γ has turned into a maximization over the set of 1-Lipschitz functions. However this problem is still untractable (search over the set of 1-Lipschitz functions). However the authors argue that it can be approximated by using neural network GANs discriminators instead of the class of 1-Lipschitz functions provided that the norm of their gradient is bound with respect to the image input (in practice, add gradient clipping).

3.2. Sinkhorn AutoDiff

Another approach developed by Genevay et al. [5] proposes to go back to the primal formulation of the Optimal Transport problem. They build on the Sinkhorn distance (introduced by Cuturi [4]). It is a regularized version of the primal problem by adding a entropy constraint on the joint distributions:

$$D_{\text{sinkhorn}}(\mathbb{P}_\text{D}, \mathbb{P}_\theta) = \inf_{\gamma \in \Pi_\beta(\mathbb{P}_\text{D}, \mathbb{P}_\theta)} \mathbb{E}_{x,y\sim\gamma} c(x,y) \tag{7}$$

where $\Pi_\beta(\mathbb{P}_\text{D}, \mathbb{P}_\theta)$ is set set of all joint distribution with entropy of at least β, with β a constant. The authors propose to approximate this distance using mini-batches. Consider two mini-batches $\mathbf{X}, \mathbf{Y}$ containing K data vectors $\mathbf{x}, \mathbf{y}$. The cost function is replaced by a transport cost matrix C of shape $K \times K$ such that $C_{i,j} = c(\mathbf{x}_i, \mathbf{y}_j)$. γ is then replaced by a matrix M of soft-matchings where M has sufficient entropy i.e. $-\operatorname{Tr}[M \log M^T] \geq \alpha$. The resulting distance is:

$$\mathcal{W}_c(X,Y) = \inf_{M \in \mathcal{M}} \operatorname{Tr}\left[MC^T\right] \tag{8}$$

where $\mathcal{M}$ is the set of matrices such that $\sum_{i=1}^n M_{ij} = \sum_{j=1}^n M_{ij} = 1$. This mini-batch Sinkhorn distance is fully tractable however it is no longer a valid metric over the probability distributions due to the fact that for a fixed mini-batch size, we do not get unbiased estimates of the gradients of the original Optimal Transport problem. To solve this, Bellemare et al. [3] propose the Cramer distance.

3.3. Cramer GAN

In [3], Bellemare et al. propose an Energy Distance, the Cramer Distance:

$$D^2_{\text{ED}}(p,g) = 2\mathbb{E}[\|\mathbf{x}-\mathbf{y}\|] - \mathbb{E}\left[\|\mathbf{x}-\mathbf{x}'\|\right] - \mathbb{E}\left[\|\mathbf{y}-\mathbf{y}'\|\right] \tag{9}$$

where $\mathbf{x}, \mathbf{x}' \sim p_D$, independent and $\mathbf{y}, \mathbf{y}' \sim p_\theta$ independent. The authors propose to train the generator with this distance, by minimizing it.

Algorithm 1 Optimal Transport GAN (OT-GAN) training algorithm with step size α, using mini-batch SGD for simplicity

```
Require: $n_{gen}$, the number of iterations of the generator per critic iteration
Require: $\eta_0$, initial critic parameters. $\theta_0$, initial generator parameters
1: for $t = 1$ to $N$ do
2:     Sample $\mathbf{X}, \mathbf{X}'$ two independent mini-batches from real data, and $\mathbf{Y}, \mathbf{Y}'$ two independent mini-batches from the generated samples
3:     $\mathcal{L} = \mathcal{W}_c(\mathbf{X}, \mathbf{Y}) + \mathcal{W}_c(\mathbf{X}, \mathbf{Y}') + \mathcal{W}_c(\mathbf{X}', \mathbf{Y}) + \mathcal{W}_c(\mathbf{X}', \mathbf{Y}') - 2\,\mathcal{W}_c(\mathbf{X}, \mathbf{X}') - 2\,\mathcal{W}_c(\mathbf{Y}, \mathbf{Y}')$
4:     if $t \bmod n_{gen} + 1 = 0$ then
5:         $\eta \leftarrow \eta + \alpha \cdot \nabla_\eta \mathcal{L}$
6:     else
7:         $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \mathcal{L}$
8:     end if
9: end for
```

Figure 2. OT-GAN algorithm in pseudo code

3.4. OT GAN

Salimans et al. in [10] went further and proposed a mini-batch energy distance. The latter is defined over mini-batch distributions $D[g(\mathbf{X}), p(\mathbf{X})]$ and has unbiased mini-batch gradients. They propose to use the Sinkhorn distance so that we have:

$$D^2_{\mathrm{MED}}(p, g) = 2\mathbb{E}\left[\mathcal{W}_c(\mathbf{X}, \mathbf{Y})\right] - \mathbb{E}\left[\mathcal{W}_c\left(\mathbf{X}, \mathbf{X}'\right)\right] - \mathbb{E}\left[\mathcal{W}_c\left(\mathbf{Y}, \mathbf{Y}'\right)\right] \quad (10)$$

where $\mathbf{X}, \mathbf{X}'$ are independently sampled mini-batches from p_D and $\mathbf{Y}, \mathbf{Y}'$ are independently sampled mini-batches from p_θ. This distance is still an invalid metric over probability distributions but is valid over mini-batches.

The authors show that the association between terms $\mathbf{Y}, \mathbf{Y}'$ and $\mathcal{W}_c(\mathbf{X}, \mathbf{Y})$ is what allows us to get unbiased gradients over mini-batches and ensure the statistical consistency of the objective. What's more, this metric still adds the primal form of the optimal transport problem though the mini-batch version of the Sinkhorn distance. Hence it combines all advantages of the previous mentioned methods.

At this point, one needs to choose c the cost function. In their experiments, the authors have found that the Euclidean distance is not the most effective one to choose to ensure statistical efficiency. Instead they propose to learn the cost function adversarially so that it can adapt to the generator distribution p_θ. Hence, they propose to use a neural network to map each input vector to another latent vector in the latent space. Therefore, we have:

$$c_\eta(\mathbf{x}, \mathbf{y}) = 1 - \frac{v_\eta(\mathbf{x}) \cdot v_\eta(\mathbf{y})}{\|v_\eta(\mathbf{x})\|_2 \, \|v_\eta(\mathbf{y})\|_2} \quad (11)$$

where v_η is the neural network and η is chosen to maximize the resulting mini-batch energy distance.

Note that an important point is that in this OT-GAN version, to ensure that c does not become degenerate, the generator is updated more often than the discriminator.

The final algorithm can be found in Figure 2.

4. Experiments

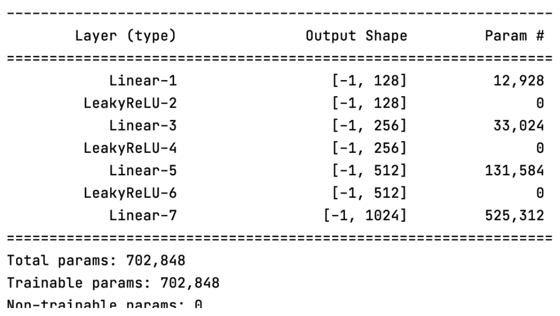
In these experiments we decided to compare three GANs: VanillaGAN ([6]), DCGAN ([8]) and OT-GAN ([10]). They are all trained on the MNIST dataset [2] whose training set contains 60000 images of digits. All experiments were done using Google Colab GPUs, Python and Pytorch library.

4.1. Vanilla-GAN

The Vanilla GAN we have implemented is mainly based on the one described by Goodfellow et al. in [6]. The architecture of the generator and the critic are described in Figures 3 and 4. The generator takes as input a latent vector of dimension 100 and outputs an image of dimension (1,28,28). The critic takes as input an image of dimension (1,28,28) and outputs a scalar.

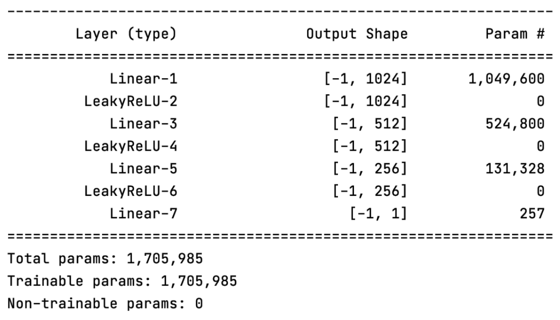
In order to obtain the best results, we performed different experiments with various hyperparameters. First of all, before arriving at our final architecture, the generated images were very noisy and it was very difficult

[2] `http://yann.lecun.com/exdb/mnist/`



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 128]          12,928
         LeakyReLU-2                 [-1, 128]               0
            Linear-3                 [-1, 256]          33,024
         LeakyReLU-4                 [-1, 256]               0
            Linear-5                 [-1, 512]         131,584
         LeakyReLU-6                 [-1, 512]               0
            Linear-7                [-1, 1024]         525,312
================================================================
Total params: 702,848
Trainable params: 702,848
Non-trainable params: 0
```

Figure 3. Architecture of the Vanilla-GAN Generator. Input is a latent vector of dimension 100.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1024]       1,049,600
         LeakyReLU-2                [-1, 1024]               0
            Linear-3                 [-1, 512]         524,800
         LeakyReLU-4                 [-1, 512]               0
            Linear-5                 [-1, 256]         131,328
         LeakyReLU-6                 [-1, 256]               0
            Linear-7                   [-1, 1]             257
================================================================
Total params: 1,705,985
Trainable params: 1,705,985
Non-trainable params: 0
```

Figure 4. Architecture of the Vanilla-GAN Critic. Input is if size (1,32,32).

to identify numbers. The figure 5 represents one of our first attempt.

Then, the main hyperparameters that evolved were the Adam Optimizer parameters (learning rate, betas), the normalization of the input pictures, the type and size of the latent space and the frequency of the critic optimization during training. In the final architecture, it was possible to visualize numbers but we faced the collapse mode with a generator producing mainly 1s, 7s and 9s. For the parameters Adam Optimizer using 2e-4 for the learning rates and values for moments terms (0.5,0.999) allowed us to obtain a good balance between training oscillation and convergence of the algorithm. With this optimizer settings, we were able to add more diversity in our generated images.

4.2. DCGAN

In a similar way to the Vanilla-GAN, we performed different experiments before obtaining our final results. Unlike the previous method, the architecture guidelines for stable Deep Convolutional GANs proposed by Radford et al.in ([8]) allowed us to obtain faster images where numbers could be easily discerned.

Our final architecture is described in Figures 7 and 8.

The Deep Convolutional GAN allowed us to obtain better quality results and the model was generally less sensitive to parameter changes. The use of Gaussian la-

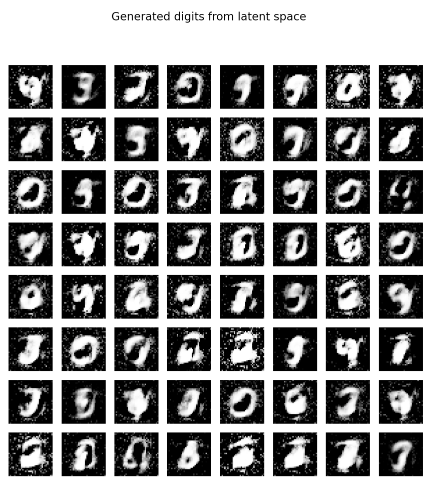


Figure 5. Noisy Digits generated by one of our earliest Vanilla GAN

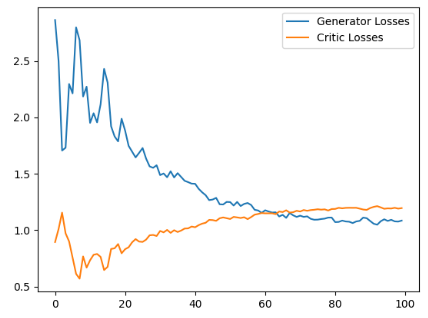
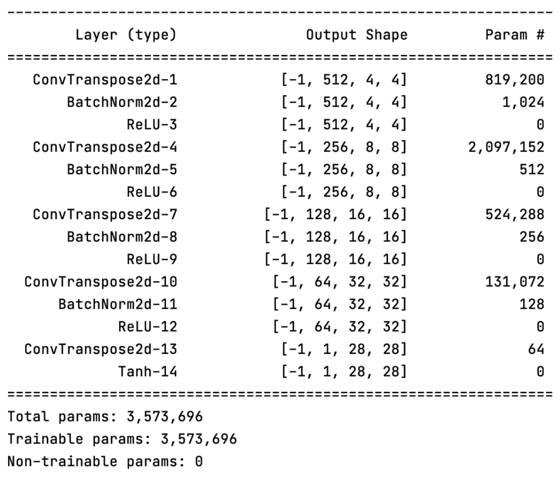


Figure 6. Evolution of losses over epochs for VanillaGAN



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 512, 4, 4]         819,200
       BatchNorm2d-2            [-1, 512, 4, 4]           1,024
              ReLU-3            [-1, 512, 4, 4]               0
   ConvTranspose2d-4            [-1, 256, 8, 8]       2,097,152
       BatchNorm2d-5            [-1, 256, 8, 8]             512
              ReLU-6            [-1, 256, 8, 8]               0
   ConvTranspose2d-7          [-1, 128, 16, 16]         524,288
       BatchNorm2d-8          [-1, 128, 16, 16]             256
              ReLU-9          [-1, 128, 16, 16]               0
  ConvTranspose2d-10           [-1, 64, 32, 32]         131,072
      BatchNorm2d-11           [-1, 64, 32, 32]             128
             ReLU-12           [-1, 64, 32, 32]               0
  ConvTranspose2d-13            [-1, 1, 28, 28]              64
             Tanh-14            [-1, 1, 28, 28]               0
================================================================
Total params: 3,573,696
Trainable params: 3,573,696
Non-trainable params: 0
```

Figure 7. Architecture of the DCGAN Generator. Input is a latent vector of dimension 100.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 16, 16]           1,024
         LeakyReLU-2           [-1, 64, 16, 16]               0
            Conv2d-3            [-1, 128, 8, 8]         131,072
       BatchNorm2d-4            [-1, 128, 8, 8]             256
         LeakyReLU-5            [-1, 128, 8, 8]               0
            Conv2d-6            [-1, 256, 4, 4]         524,288
       BatchNorm2d-7            [-1, 256, 4, 4]             512
         LeakyReLU-8            [-1, 256, 4, 4]               0
            Conv2d-9              [-1, 1, 2, 2]           4,096
          Sigmoid-10              [-1, 1, 2, 2]               0
================================================================
Total params: 661,248
Trainable params: 661,248
Non-trainable params: 0
```

Figure 8. Architecture of the DCGAN Critic. Input is if size (1,28,28).

tent space (instead of uniform latent space) and images scale to the range of the Tanh activation function [-1,1] have resulted in images with more diversity. The settings for the Adam optimizer have remained the same as for the Vanilla.

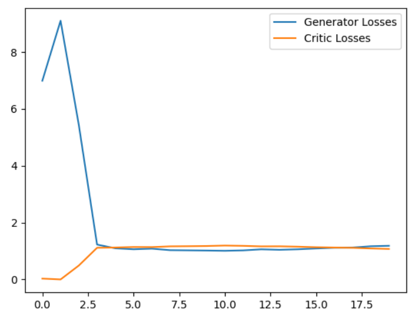


Figure 9. Evolution of losses over epochs for DCGAN

4.3. OT-GAN

The OT-GAN we have implemented is based on the architecture given by Salimans et al. [10] in their Appendix B when they present their experiment on the famous CIFAR-10 dataset.

The chosen Generator architecture for a latent dimension of 50 is a depicted in Figure 10. We corresponding Critic is given in Figure 11.

We made several experiments and trials with various hyperparameters before reached the final OT-GAN that we are proposing. The best hyperparameters can be found in Table 1.

Our first approach was to use the exact same architecture than the one proposed in the paper [10], developed for CIFAR-10 dataset. We chose, as the authors

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
          Linear-1                  [-1, 8192]         417,792
             GLU-2                  [-1, 4096]               0
         Reshape-3            [-1, 256, 4, 4]               0
        Upsample-4            [-1, 256, 8, 8]               0
          Conv2d-5            [-1, 256, 8, 8]         590,080
             GLU-6            [-1, 128, 8, 8]               0
        Upsample-7          [-1, 128, 16, 16]               0
          Conv2d-8          [-1, 128, 16, 16]         147,584
             GLU-9           [-1, 64, 16, 16]               0
       Upsample-10           [-1, 64, 32, 32]               0
         Conv2d-11           [-1, 64, 32, 32]          36,928
            GLU-12           [-1, 32, 32, 32]               0
         Conv2d-13            [-1, 1, 32, 32]             289
           Tanh-14            [-1, 1, 32, 32]               0
================================================================
Total params: 1,192,673
Trainable params: 1,192,673
Non-trainable params: 0
```

Figure 10. Architecture of the OT-GAN Generator. Input is a latent vector of dimension 50.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1           [-1, 32, 32, 32]             320
           CReLU-2           [-1, 64, 32, 32]               0
          Conv2d-3           [-1, 64, 16, 16]          36,928
           CReLU-4          [-1, 128, 16, 16]               0
          Conv2d-5            [-1, 128, 8, 8]         147,584
           CReLU-6            [-1, 256, 8, 8]               0
          Conv2d-7            [-1, 256, 4, 4]         590,080
           CReLU-8            [-1, 512, 4, 4]               0
         Reshape-9                 [-1, 8192]               0
    L2Normalize-10                 [-1, 8192]               0
================================================================
Total params: 774,912
Trainable params: 774,912
Non-trainable params: 0
```

Figure 11. Architecture of the OT-GAN Critic. Input is of size (1,32,32).

Batch size	200
Maximum number of epochs	100
Latent dimension	50
Latent type	Gaussian
Kernel size	3
Critic learning rate	1e-4
Generator learning rate	1e-4
Generator hidden dimension	256
Critic hidden dimension	32
Weight decay	0
Adam optimizer β_1	0.5
Adam optimizer β_2	0.999
Number of iterations of the generator per critic iteration	3
Regularization factor (Sinkhorn algorithm)	1
Number of iterations (Sinkhorn algorithm)	100

Table 1. Hyperparameters leading to the best OT-GAN we obtained.

proposed, a kernel size of 5, the latent space of uniform ($\mathcal{U}_{[-1,1]}$), the critic was updated every 3 generator updates. The choice for the two Sinkhorn-related parameters, though both set to 500 in Salimans paper, was bit more complex on our side. Indeed, when settings the number of iterations to a too high value (in practice, more than 200), the Sinkhorn algorithm had trouble to converge. Consequently the whole GAN did not converge in a reasonable time. However when setting this number too low (less than 100), visual results, as in images generated, only looked like blobs or seahorses.

After several trials, we reached reasonable results in terms both of visual results and time to converge setting the entropy penalty to 1 and the number of iterations in the Sinkhorn algorithm to 100.

However, the results were still no satisfying as can be seen in Figure 12. These are still not numbers though we were closer than before. A great time in this project was therefore allocated to the process of trials and errors. Among all the variations we attempted were:

- changing the hidden dimension of both the generator and the critic
- changing the number of iterations of the generator per critic iteration
- changing the learning rate of both the generator and the critic
- changing the β_1 and β_2 in Adam optimizer
- changing the batch size (though we were limited by the available VRAM of Google Colab GPUs)
- changing the kernel size
- changing the type of the latent space i.e. the distribution from which the noise vector are drawn
- changing the dimension of the latent space

The ones having the most impact on the results were changing the kernel size, changing the type of latent space and changing its dimension. By changing the kernel size and the latent space dimension respectively to 5 and 50, we were able to obtain images that look like digits (Figure 13).

A huge caveat is which we have fallen is the mode collapse issue as can be seen in Figure 13 where only 0s, 1s and 9s are generated. As explained in our theoretical section, this is one of the short-comings of the GAN training scheme where the generator finds an ”easy” way to always fool the critic. Taking inspiration from the literature, we decided to switch to a Gaussian latent space instead of a uniform one in order to alleviate this representation collapse issue. This allowed us to reach our best result, detailed in the next section. Figure 14 display the evolution of the critic and the generator losses as a function of time (epoch). We can see that convergence

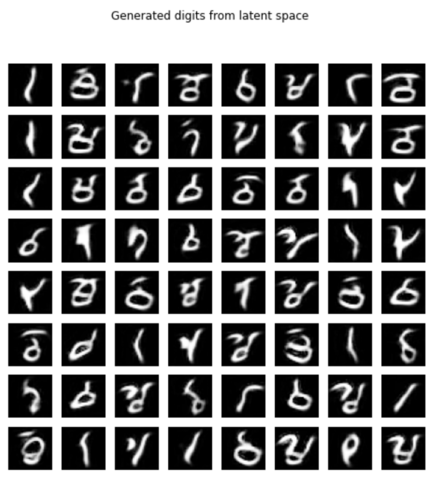


Figure 12. Digits generated by one of our earliest OT-GAN.

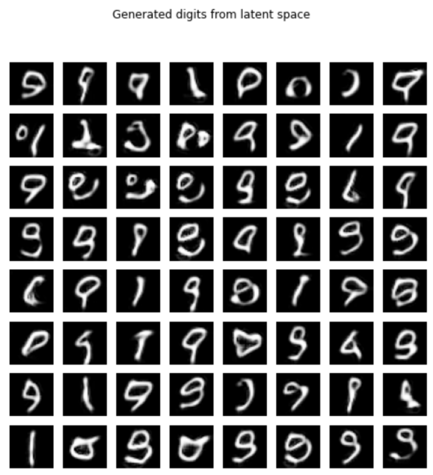


Figure 13. Digits generated by one of our improved OT-GAN.

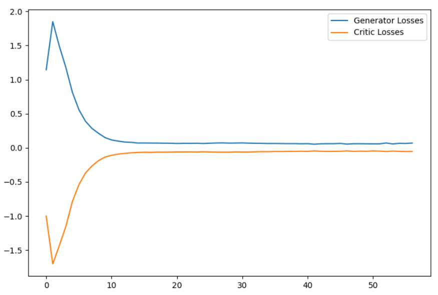


Figure 14. Evolution of critic and generator loss over epochs - OT-GAN

is actually attained quite fast compared to VanillaGAN since the losses are quasi-constant after epoch 15.

Note that following the considerations of Salimans et al. [10] around the size of the batch size, we did several experiments with this parameter. We first set it to 100 but the results were too unstable. We increased it to 200 and even 400. However, for 400, we found that the results were not as good as for batch size of 200, after more than 70 epochs: they were unstable and quite blurry.

5. Results

5.1. Visual comparison

It is often complicated to find a good metric to compare generative models as suggested by the prolific literature on the matter. However one of the first metric is visual comparison of the generated images. The procedure adopted is quite straight-forward: for each generator we sampled 64 vectors from the latent space (either Gaussian or Uniform depending on the model hyperparameters), fed them to the trained generator and displayed the results in an image grid.

5.1.1 VanillaGAN

The results obtained with the VanillaGAN lack sharpness. Although it is possible to discern several digits, they remain globally noisy. In particular, it was difficult to obtain results that combined sharpness and diversity. Indeed, a sharper image generally caused a collapse mode.

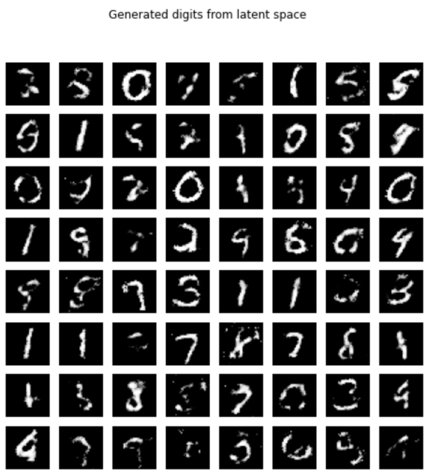


Figure 15. Digits generated by our final Vanilla GAN.

5.1.2 DCGAN

The results obtained with our DCGAN are much better. Indeed, the generated images are sharper and all the digits can be found.

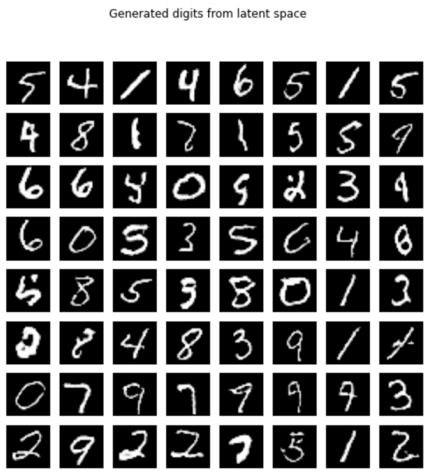


Figure 16. Digits generated by our final DCGAN.

5.1.3 OT-GAN

Our best OT-GAN produces the digits displayed in Figure 17. Compared to our previous OT-GANs, here the mode collapse issue seems to be avoided: there are 0s, 1s, 3s, 4s, 5s, 6s, 7s, 8s and 9s. However we are still missing the number 2 which is harder for our model to produce. The diversity of the generated images is greater, therefore the results are decent but clearly not perfect, highlighting the difficulty to train such generative models.

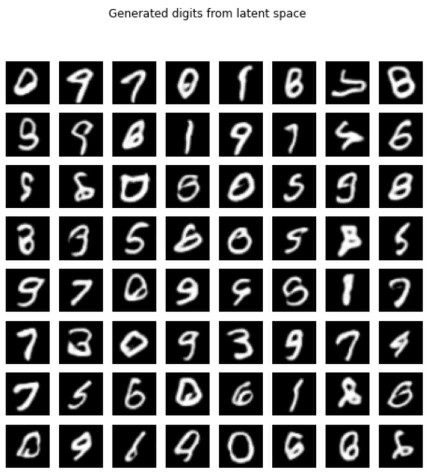


Figure 17. Digits generated by our final OT-GAN.

5.2. Interpolation

Another experiment that we tried to was interpolate between 4 digits in order to better understand the space into which the generator maps the latent space. The procedure is as follows: we fix 4 vectors from the latent space (Gaussian or Uniform depending on the model hyperparameters) and we linearly interpolate between the 4 vectors, while feeding all interpolated vectors to the generator. Finally, we plot these images into a grid to get a feeling of how the space is structured.

5.2.1 VanillaGAN

In the VanillaGAN, the evolution of the images during the interpolation remains quite stable. In Figure 18, we mainly observe 0s, 5s and 3s that are relatively identical. There is not much diversity.

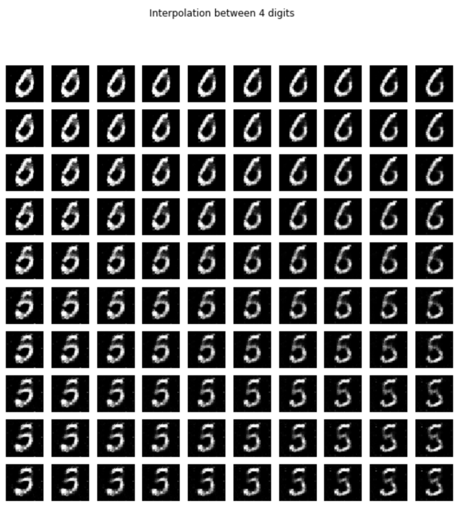


Figure 18. Interpolation for Vanilla GAN

5.2.2 DCGAN

The best results for the interpolation experiment were reached by DCGAN. Indeed, in addition to a higher sharpness, we can also observe in Figure 19,a great diversity of numbers with 2, 3, 8, 6, 9 and 0.

5.2.3 OT-GAN

In Figure 20, we see how a 1 gets first changed slowly into a 5 before becoming a 5 and finally turning into a 9. At each point in the image space, the interpolated vector looks like a really digit (as can be found in the dataset).

5.3. Inception score

5.3.1 Theory

The Inception Score is a widely used metric for assessing the generated image outputs of Generative Adversarial Networks (GANs). It was first proposed in ”Improved techniques for training GANs” [9]. The score has been found to be consistently correlated to human evaluation of GANs. It is therefore used as an automatic alternative to human assessment.

The score measures the performance of the network over two criteria :

- Image Diversity : there is a wide range of objects generated
- Image Quality : the subject of the image is clearly identifiable

The Inception Score gets its name of the Inception v3 model [11]. Inception v3 is an image recognition model originally trained on the ILSVRC 2014. It is a convolution neural network developed with the goal of

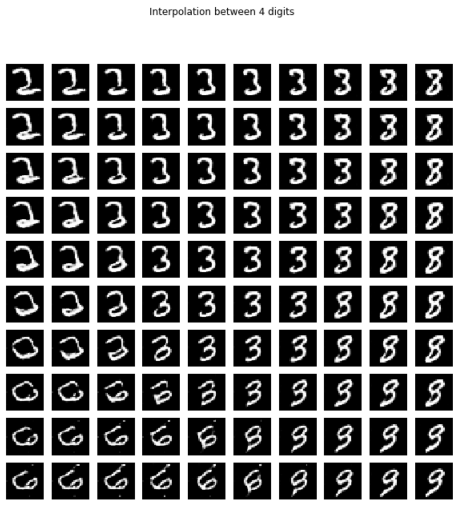


Figure 19. Interpolation for DCGAN

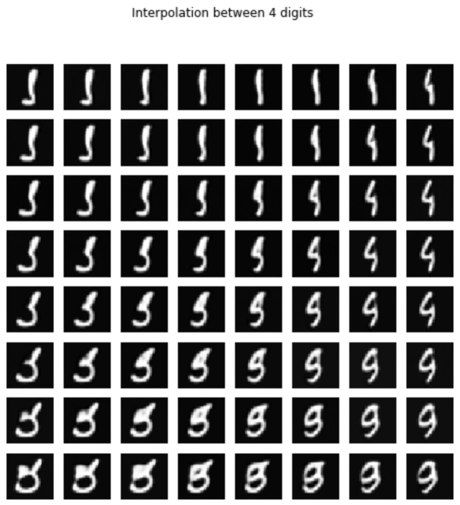


Figure 20. Interpolation for OT-GAN

making a deeper network while keeping the number of parameters manageable. It has under 25 millions parameters, which is quite small compared to the SOTA model AlexNet which has 60 millions parameters.

Inception v3 outputs a probability distribution over labels for the image. This allows to evaluate what is recognizable in the image, and how much so. In order to go from the probability distribution to the score, we combine the label probability distribution of a large number of generated images. This creates an aggregated distribution, which indicates the variety in the generators output.

Ideally we want the individual probability distribution to have one probability much higher than the others, while our aggregated distribution should be flat, uniform.

To formalize this measure, the authors use a statistical distance : the Kullback-Leiber divergence (KL divergence), also called relative entropy. It is defined as such for P and Q two discrete probability distributions defined on the same probability space :

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

Images that contain easily distinguishable objects should have a conditional label distribution with low entropy whereas the aggregated distribution should have high entropy. Combining these requirements the authors propose the score :

$$\exp(\mathbb{E}_x KL(p(x|y)||p(y)))$$

The measure is exponentiated so that the resulting values are more easy to compare.

5.3.2 Limitations

Unfortunately the Inception score, albeit popular, has been found to have a few limitations [2]. Most come back to the fact that the score is dependant on the underlying network and the data on which it has been trained (Inception v3 is traditionnaly trained on ILSVRC 2014). This means that if the generated images belong to categories that are not in the original dataset, or if they get their diversity from differences more subtles than the original labels (for example a network which generates different breeds of dogs, when the Inception was trained to differenciate between animals), will score artificially low.

In addition, the Inception score is very reliant on the features that allow the classifier to labelize images, and can therefore ”cheat”. For example, a GAN which generates horses with two heads might have a high score because the classifier can identify them correctly.

There is also no measure of intra-class diversity, meaning that the GAN can generate very similar image for one given class and still score high. There is also no penalization to proximity with the training data.

In our case, the first remark is very significant : most pre-trained Inception v3 have not been trained to recognize digits, and therefore give very poor results on our MNIST inspired images. That is why we tried to retrain Inception to recognize digits but the task proved very time consuming.

Consequently, we decided instead to take inspiration from the score, and develop our own, based on a simple classifier rather than on Inception. We retrained a classifier specifically on the MNIST Dataset, and derived a score from the same metric as Inception. This shields us from the dependency on the training dataset but exposes us to the other limitations of the metric.

5.3.3 Results

The previous subsection exposed the limits of the Inception score and presented our choice of retraining a classifier on the MNIST dataset to derive our own inception-like metric. Table 2 presents the results. Higher score means higher image quality.

	Inception Score	MNIST Score
OT-GAN	1.80	**1.14**
DCGAN	1.57	1.06
VanillaGAN	**2.33**	1.13

Table 2. Inception score and own MNIST score for our best models.

With our own MNIST score, all three best models are quite similar, but OT-GAN and Vanilla-GAN are slightly better. This is **not** confirmed by the visual results that we exposed in the previous section 5.1. Indeeed, in the previous section, it seems that DCGAN and OT-GAN are the two generative models given the best, human readable, results both in terms of diversity of images generates (DCGAN has the widest range) and image quality (again, DCGAN has the best visual digits generated). The discrepancy between the MNIST score and the visual results we exposed can be explained by the limitations mentioned above.

6. Conclusion

In this project we presented and implemented from scratch several generative models. We have compared their performances on the MNIST dataset. As it is often hard to evaluate the performances of generative models, we focused on visual comparison of generated images (DCGAN and OT-GAN seem to produce decent results though not perfect). We also looked at the interpolation process between four digits to get a better understanding of what the generator is doing. On this matter, it is DC-GAN that proposes the greatest diversity of generated digits and maps all interpolated vectors to a real digit. Finally, taking inspiration from the Inception Score, we developed our own MNIST score but the results were not totally in agreement with our visual inspection.

Overall we noticed that training GANs is complicated in practice and that the results are far from perfect. Notice that this is the case on a simple dataset such as MNIST.

To go further, we could check if the distribution of generated images by the OT-GAN follows the same distribution as the one of the dataset. This could be done by comparing the classes' proportion in both a pool of generated images and the MNIST dataset. In theory, since OT-GAN builds on Optimal Transport, the two empirical frequencies should be similar, while this might not be the case for VanillaGAN and DCGAN.

References

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

[2] S. Barratt and R. Sharma. A note on the inception score, 2018.

[3] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos. The cramer distance as a solution to biased wasserstein gradients, 2017.

[4] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances. 2013.

[5] A. Genevay, G. Peyré, and M. Cuturi. Learning generative models with sinkhorn divergences, 2017.

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

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

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

[10] T. Salimans, H. Zhang, A. Radford, and D. Metaxas. Improving gans using optimal transport, 2018.

[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015.