OPTIMAL TRANSPORT

OT-GAN

Gabriel Kasmi, Hugo Thimonier

ENSAE 3A - 2019-2020

Abstract

In this report we present our replication of the results of Genevay et al. (2017b). In the first section we review the literature and recall some theoretical aspects on GANs, optimal transport and OT-GAN. Then, in the second section we present our approach and the results we've obtained.

1 Theoretical background

1.1 Generative adversarial networks

Generative adversarial networks have been introduced by Goodfellow et al. (2014) and consists in two neural networks, a *generator*, whose goal is to map low dimensional vectors z onto a higher dimensional space in order to mimic the distribution P_{data} of the data. The generator itself is a deterministic and parametrized function $g_\theta : \mathbb{R}^d \to \mathbb{R}^n$ where n is the dimension of the data and d the latent space dimension. Usually, $d \ll n$. Put otherwise, the objective of the generator is to define a distribution P_g to be as close as possible to P_{data}.
In order to reach this goal, another network called the *discriminator* is defined and trained in the same time as the generator. The objective of this network is to be able to distinguish between the real and the fake images.
There is therefore an *adversarial* game between the two networks : the generator aims at fooling the discriminator, which is in turn trained so as to be able to distinguish between real and fake images, therefore pushing the generator to generate more and more precise outputs. Denoting G the generator and D the discriminator, the two networks play the following game:

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

It can be shown that (1) has a global optimum for $P_g = P_{data}$. The results is based on the fact that for G fixed, the optimal discriminator D is given by

$$D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}$$

Using the latter expression in (1) for D allows to rewrite the game as follows:

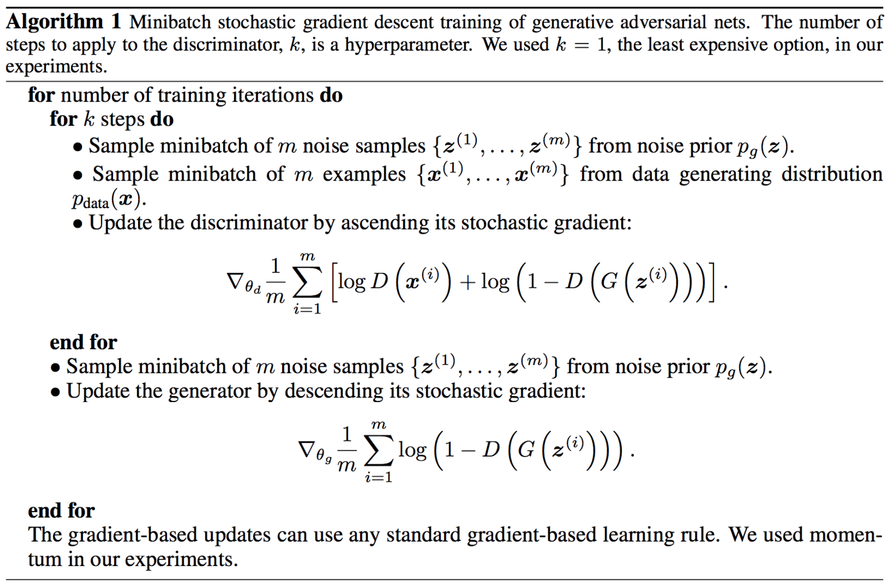
$$\begin{aligned} C(G) = \max_G V(G,D) &= \mathbb{E}_{x\sim P_{data}}\left[\log \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right] + \mathbb{E}_{z\sim P_g}\left[\log \frac{P_g(x)}{P_{data}(x)+P_g(x)}\right] \\ &= -\log 4 + KL\left(P_{data}, fracP_{data} + P_g 2\right) + +KL\left(P_g, fracP_{data} + P_g 2\right) \\ &= -\log 4 + 2JSD(P_{data}, P_g) \end{aligned}$$

Where KL denotes the Kullback-Leibler divergence and JSD the Jensen-Shannon Divergence. Since $JSD(p,q) \geq 0$ and $JSD(p,q) = 0 \iff q = p$, it follows that the optimum is reached for $P_g = P_{data}$.

In practice GANs are implemented using the algorith depicted on figure 1. Another major result from Goodfellow et al. (2014) is that this for all G, the discriminator converges to its optimum and if P_G is updated so as to improve

$$\mathbb{E}_{x \sim P_{data}} [\log D^*(x)] + \mathbb{E}_{x \sim P_g} [\log(1 - D^*(x))]$$

then P_G converges to P_{data}.



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:
        $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$
  end for
  • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:
      $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 1: GAN algorithm. Source : Goodfellow et al. (2014)

1.2 Optimal transport

1.2.1 Reminders

Optimal Transport is a theory that allows us to compare two (weighted) points clouds (X, a) and (Y, b), where $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^{m \times d}$ are the locations of the n (resp. m) points in dimension d, and $a \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ are the weights. We ask that the total weights sum to one, i.e. $\sum_{i=1}^n a_i = \sum_{j=1}^m b_j = 1$.
The basic idea of Optimal Transport is to "transport" the mass located at points X to the mass located at points Y.
Let us denote by $\mathscr{U}(a,b) = \left\{ P \in \mathbb{R}^{n \times m} \,|\, P \geq 0, \sum_{j=1}^m P_{ij} = a_i, \sum_{i=1}^n P_{ij} = b_j \right\}$ the set of admissible transport plans.
If $P \in \mathscr{U}(a,b)$, the quantity $P_{ij} \geq 0$ should be regarded as the mass transported from point X_i to point Y_j. For this reason, it is called a transport plan.
We will also consider a cost matrix $C \in \mathbb{R}^{n \times m}$. The quantity C_{ij} should be regarded as the cost paid for transporting one unit of mass from X_i to Y_j. This cost is usually computed using the positions X_i and Y_j, for example $C_{ij} = \|X_i - Y_j\|$ or $C_{ij} = \|X_i - Y_j\|^2$.
Then transporting mass according to $P \in \mathscr{U}(a,b)$ has a total cost of $\sum_{ij} P_{ij} C_{ij}$.
In "Optimal Transport", there is the word Optimal. Indeed, we want to find a transport plan $P \in \mathscr{U}(a,b)$ that will

minimize its total cost. In other words, we want to solve

$$\min_{P\in\mathscr{U}(a,b)} \sum_{ij} C_{ij}P_{ij}.$$

This problem is a Linear Program: the objective function is linear in the variable P, and the constraints are linear in P. We can thus solve this problem using classical Linear Programming algorithms, such as the simplex algorithm. If P^* is a solution to the Optimal Transport problem, we will say that P^* is an optimal transport plan between (X, a) and (Y, b), and that $\sum_{ij} P^*_{ij}C_{ij}$ is the optimal transport distance between (X, a) and (Y, b): it is the minimal amount of "energy" that is necessary to transport the initial mass located at points X to the target mass lcoated at points Y.

1.2.2 Sinkhorn's algorithm

In real applications, and especially in Machine Learning, we often have to deal with huge numbers of points. In this case, the linear programming algorithms which have cubic complexity will take too much time to run.
That's why in practice, among other reasons, people minimize another criterion given by

$$\min_{P\in\mathscr{U}(a,b)} \langle C, P\rangle + \epsilon \sum_{ij} P_{ij}[\log(P_{ij}) - 1].$$

When ϵ is sufficiently small, we can consider that a solution to the above problem (often refered to as "Entropy-regularized Optimal Transport") is a good approximation of a real optimal transport plan. This entropy regularized method has been introduced by Cuturi (2013).
In order to solve this problem, one can remark that the optimality conditions imply that a solution P^*_ϵ necessarily is of the form $P^*_\epsilon = \operatorname{diag}(u)\, K \operatorname{diag}(v)$, where $K = \exp(-C/\epsilon)$ and u, v are two non-negative vectors.
P^*_ϵ should verify the constraints, i.e. $P^*_\epsilon \in \mathscr{U}(a, b)$, so that

$$P^*_\epsilon 1_m = a \text{ and } (P^*_\epsilon)^T 1_n = b$$

which can be rewritten as

$$u \odot (Kv) = a \text{ and } v \odot (K^T u) = b$$

Then Sinkhorn's algorithm alternate between the resolution of these two equations, and reads

$$u \leftarrow \frac{a}{Kv} \text{ and } v \leftarrow \frac{b}{K^T u}$$

1.3 OT-GAN

1.3.1 Overview

The problem laid out by generative modelling can be seen as matching two probability distributions. In many cases and especially when dealing with high dimensional densities, standard metrics may not exist. In this context, GANs are an attempt to overcome these difficulties but they can be unstable in some cases, as pointed out by Arjovsky and Bottou (2017). In order to overcome these difficulties, Arjovsky et al. (2017) tried to determine under which conditions a sequence of probability distribution $(P_g)_t$ will converge towards a target distribution P_{data} and how to define a divergence between P_g and P_{data}.
The authors therefore reinterpreted the GAN modelling in the context of optimal transport by proposing the

1-Wasserstein distance (or earth mover distance, EMD) as the objective for generative modelling:

$$D_{EMD}(p,g) = \inf_{\gamma \in \Pi(p,g)} \mathbb{E}_{x,y\sim\gamma} c(x,y) \tag{2}$$

Where c is a cost, chosen by Arjovsky et al. (2017) to be the the Euclidean distance. However, the formulation of equation (2) is intractable, so the authors proposed instead the dual formulation, given by

$$D_{EMD} = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p} f(x) - \mathbb{E}_{y\sim g} f(y) \tag{3}$$

The minimization over the class of 1-Lipschitz functions is still intractable, but Arjovsky et al. (2017) argue that it can be approximated by GAN discriminators, that are now reinterpreted as critics. In practice, (3) is substituted as the objective for the discriminator in (1). From a theoretical perspective, this paper highlighted a connection between optimal transport and GANs and how the latter can be used to approximate optimal transport problems. This connection has been further explored by Genevay et al. (2017a) and Tolstikhin et al. (2017) for instance.

Instead of approximating the dual formulation, Genevay et al. (2017b) chose to approximate the primal formulation of OT using generative modelling. Instead of considering the EMD distance, the authors choose the entropic regularization of optimal transport, introduced by Cuturi (2013). This approach is fully tractable. However, the main drawback is that the gradients computed over the minibatches of data are no longer an unbiased estimator of the original problem given by:

$$D_{Sinkhorn}(p,g) = \inf_{\gamma \in \Pi_\varepsilon(p,g)} \mathbb{E}_{x,y\sim\gamma} \left[c(x,y)\right] \tag{4}$$

Where ε is the entropic regularization parameter. These drawbacks have been discussed in Salimans et al. (2018). In the next section, we recall the theoretical results and present the approach proposed by Genevay et al. (2017b).

1.3.2 Sinkhorn divergences for generative models

Denisty fitting problem with the Sinkhorn loss The problem is to minimize a given loss with respect to a vector of parameters θ between the generator measure P_g and the distribution of the data P_{data}. Since a generator g_θ is a parametrized mapping from a low dimensional space with a probability measure ζ, we can write $P_g \equiv P_\theta = g_{\theta\#}\zeta$, where # is the standard "push forward" operator used in optimal transport. Formally, we have :

$$\theta \in \arg\min_\theta \mathscr{L}(P_\theta, P_{data}) \tag{5}$$

A well known loss in statistics is the opposite of the maximum likelihood. However, as pointed out by Arjovsky et al. (2017), such a criterion may well not exist in non euclidean spaces and optimal transport distances can instead be preferred. The authors introduce the regularized optimal transport problem, as defined in (4). Denoting π_ε the solution to (4), the associated Wasserstein distance writes :

$$\mathscr{W}_{c,\varepsilon} = \mathbb{E}_{\pi_\varepsilon(x,y)} c(x,y) = \int c(x,y) \mathrm{d}\pi_\varepsilon(x,y)$$

The Sinkhorn loss between two probability measures μ and ν is then defined as:

$$\bar{\mathscr{W}}_{c,\varepsilon}(\mu,\nu) = 2\mathscr{W}_{c,\varepsilon}(\mu,\nu) - \mathscr{W}_{c,\varepsilon}(\mu,\mu) - \mathscr{W}_{c,\varepsilon}(\nu,\nu) \tag{6}$$

And the authors show that it has the following behavior in ε (see theorem 1 in Genevay et al. (2017b)):

$$\text{If } \varepsilon \to 0 \ , \bar{\mathscr{W}}_{c,\varepsilon}(\mu,\nu) \to 2\mathscr{W}_{c,\varepsilon}(\mu,\nu)$$
$$\text{If } \varepsilon \to \infty \ , \bar{\mathscr{W}}_{c,\varepsilon}(\mu,\nu) \to MMD_{-c}(\mu,\nu)$$

Where MMD is the maximum mean discrepancy for kernel k is defined as :

$$MMD_k := \|\mu,\nu\|_k = \mathbb{E}_{\mu\otimes\mu}k(X,X') + \mathbb{E}_{\nu\otimes\nu}k(Y,Y') - 2\mathbb{E}_{\mu\otimes\nu}k(X,Y) +$$

Denoting $E_\varepsilon(\theta) := \bar{\mathscr{W}}_{c,\varepsilon}(\mu,\nu)$, the density fitting problem (5) can be rewritten as :

$$\min_\theta E_\varepsilon(\theta)$$

The AutoDiff algorithm As already said, computing OT exactly can be costly if not intractable, so is the computation of the gradient of the Wasserstein loss. To overcome this difficulty without using the dual formulation of the OT problem, the authors approximate $E_\varepsilon(\theta)$ by an estimate computed over a minibatch of size (n,m) and using the Sinkhorn algorithm to approximate the optimal transport plan. Two approximations are made and the resulting objective is denoted $\hat{E}_\varepsilon^{(L)}$ where L is the number of iterations of the Sinkhorn algorithm. Since the Sinkhorn algorithm gives us an approximation of the optimal transport plan P_L given a cost c, we have $\hat{E}_\varepsilon^{(L)} = \langle c, P_L \rangle$ Rephrased in terms of OT-GAN, the discriminator is called a critic learns along with the generator. In the Sinkhorn framework, this amounts to learning the cost function, which is important in high-dimensionnal settings. The cost function is denoted $c_\varphi(x,y) = \|f_\varphi(x) - f_\varphi(y)\|$ where f_φ is a neural network parametrized with the vector φ. This allows us to rewrite the problem as a minimax game similar to (1) but with an alternative objective function :

$$\min_\theta \max_\varphi \bar{\mathscr{W}}_{c_\varphi,\varepsilon}(\mu,\nu)$$

The algorithm is summarized on figure 2 and depicted on the flow diagram on figure 6 in the appendix.

```
Algorithm 1 SGD with Auto-diff
Input: $\theta_0$, $\varphi_0$, $(y_j)_{j=1}^n$ (the real data), m(batch size),
  $L$ (number of Sinkhorn iterations), $\varepsilon$ (regularization
  parameter), $\alpha$ (learning rate)
Output: $\theta$, $\varphi$
  $\theta \leftarrow \theta_0$, $\varphi \leftarrow \varphi_0$,
  for $k = 1, 2, \dots$ do
      for $t = 1, 2, \dots, n_c$ do
          Sample $(y_j)_{j=1}^m$ from the observations.
          Sample $(z_i)_{i=1}^m \overset{\text{i.i.d}}{\sim} \zeta$, $(x_i)_{i=1}^m \overset{\text{def.}}{=} g_\theta(z_1^m)$
          $\mathtt{grad}_\varphi \leftarrow \mathtt{AutoDiff}_\varphi\Big(2\hat{W}_{\varphi,\varepsilon}^{(L)}(x_1^m, y_1^m)$
                  $-\hat{W}_{\varphi,\varepsilon}^{(L)}(x_1^m, x_1^m) - \hat{W}_{\varphi,\varepsilon}^{(L)}(y_1^m, y_1^m)\Big)$
          $\varphi \leftarrow \varphi + \alpha\mathtt{RMSProp}(\mathtt{grad}_\varphi)$.
          $\varphi \leftarrow \mathtt{clip}(\varphi, -c, c)$
      end for
      Sample $(y_j)_{j=1}^m$ from the observations.
      Sample $(z_i)_{i=1}^m \overset{\text{i.i.d}}{\sim} \zeta$, $(x_i)_{i=1}^m \overset{\text{def.}}{=} g_\theta(z_1^m)$
      $\mathtt{grad}_\theta \leftarrow \mathtt{AutoDiff}_\theta\Big(2\hat{W}_{\varphi,\varepsilon}^{(L)}(x_1^m, y_1^m)$
              $-\hat{W}_{\varphi,\varepsilon}^{(L)}(x_1^m, x_1^m) - \hat{W}_{\varphi,\varepsilon}^{(L)}(y_1^m, y_1^m)\Big)$
      $\theta \leftarrow \theta - \alpha\mathtt{RMSProp}(\mathtt{grad}_\theta)$.
  end for
```

Figure 2: Sinkhorn AutoDiff algorithm. Source : Genevay et al. (2017b)

Experiments The authors conduct several experiments. Our goal is to replicate one of them, namely learning the digits of the MNIST dataset. For this case, the cost function doesn't need any learning and is simply the ℓ_2 norm between x and y, i.e. $c(x, y) = \|x - y\|_2$. The generator is a multilayer perceptron with one hidden layer of 500 units, an output layer of 28*28 units and an input layer of 2 units. The dimension of the latent space is therefore 2.

2 Implementation and results

Our results regarding the replicatoin of the experiment are summarized in the notebook `sinkhorn_gan.ipynb` available here. Additionnally, we designed and trained our own GAN as a benchmark. Section 2.1 highlights some key features of this bechmark model and appendix B presents the characteristics of the GAN.

2.1 Benchmark

The generator is comprised of three hidden layers of respective sizes 32, 64 and 128. The latent space dimension is 100 and the output 28*28. The activation functions are ReLU activation functions. The discriminator has a symmetric architecture, the except that dimension of the output layer is 1. Figure 3 displays some images generated by our networ after 100 traning epochs over the MNIST dataset.

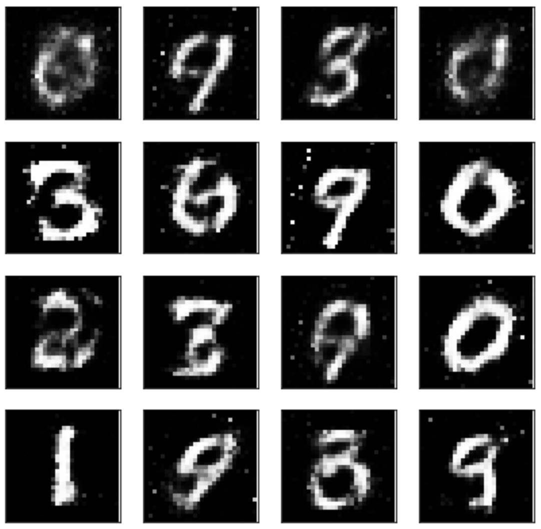


Figure 3: Images generated by the trained baseline GAN

All necessary material for the replication of the results is contained in the folder `vanilla_gan` available here. A pretrained model as well as its generated samples is stored in the folder `model` and can be used in the notebook `visualize_vanilla_gan.ipynb`. The model has been trained using the notebook `define_train_gan.ipynb`. This network had initially been constructed for comparison purposes.

2.2 Replication and discussion

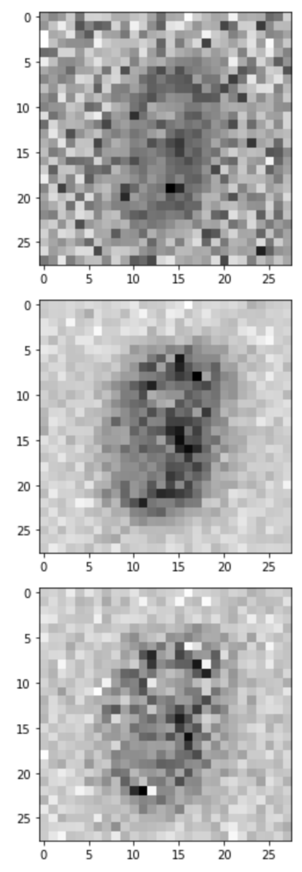
Our approach was the following : we defined our network and simply benchmark it on dummy examples with a standard MSE loss in order to see whether the model was behaving as expected. Figure **??** shows that indeed the model behaves as expected : it learns the "average" image when trained on the whole dataset or learn to generate one image if it is always tested against the same image.

We then tried to follow the paper's guidelines and defined three blocks : a cost matrix, the Sinkhorn loop which returned the estimation of the Wasserstein disance and a wrapper computing the Sinkhorn loss as given in (6). The first difficulty encountered was the numerical instability of a ℓ_2 cost matrix. After several trials and no matter the normalization or regularizations set in place, it turned out that the computation of the loss often yielded `nans`. In order to avoid this problem, we consider the cosine similarity matrix (which is the cost matrix used in Salimans et al. (2018)) because its limited range made it less susceptible of exploding. Indeed, we noticed that the numerical instability occured very often when computing the Sinkhorn kernel $K = \exp(-C/\varepsilon)$, which often diverged to 0. The cosine similarity matrix is given by

$$K(x, y) = \frac{xy^\top}{\|x\|_2 \|y\|_2}$$

Where in practice we replace the denominator by $\max(\|x\|_2 \|y\|_2, eps)$ in order to avoid divisions by 0. The $(i, j)^{th}$ entry of this matrix corresponds to the cosine similarity between an observation from batches i and j.
The drafts of our implementations are contained in the folder `drafts` available here. In these notebooks we display our trials (and errors !) as we initially computed the cost matrix between the "averaged" images over the batch. Each entry therefore corresponded to the distance between two pixels across the batches. It turns out that with such a cost matrix, the model seems to learn a single pattern, such as for instance the pattern depicted in figure 13.

However a new problem arised : it turned out that our model was indeed learning but at some point the loss eventually "converged" to a certain value. However, the digits generated were not mimicking MNIST digits, as it can be seen from figure 4. . On figure 4a the evolution of the generated samples after 0, 10 and 20 epochs is displayed. Performances did not really improved beyon this point. We can see that the model indeed generates images but they remain far from MNIST digits. On figure 4b are displayed our "best" results with the cosine distance matrix. Again a pattern appears but it is not very readable. Nevertheless, we can highlight the fact that the model doesn't seem to generate an average number.


(a) Evolution during traning

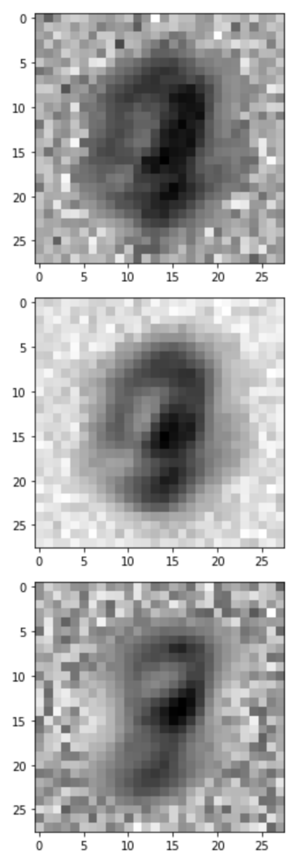

(b) Best outcome

Figure 4: Generated samples with the Sinkhorn-GAN

We tried numerous combinations of learning rates and batch sizes and it turned out that the behaviour of the loss function was following the same pattern, as depicted on figure 5. More example of losses are displayed on figure 12.

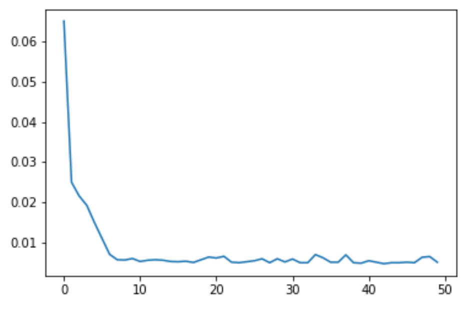


Figure 5: Evolution of the loss

During our experiments in order to understand what was hindering our replication, we managed to rule out the following possibilities :

- The network architecture. Even with two layers, the behavior was the same and the network itself is working as expected
- The activation function : we tried with ReLU and Sigmoid activation functions and in both cases the results

were similar. The images are "smoother" with the sigmoid activation (see for instance figure 14).

- The hyperparameters : the batch size, learning rate, number of Sinkhorn iterations as well as the regularization parameter did not seem to have an impact on the overall outcome
- The computation of the transport plan seemed to be correct and was tested against the exact LP formulation of the `POT` package

However, the cause of our failure unfortunately remains unknown.

References

Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. arxiv e-prints, art. *arXiv preprint arXiv:1701.04862*.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300.

Genevay, A., Peyré, G., and Cuturi, M. (2017a). Gan and vae from an optimal transport point of view. *arXiv preprint arXiv:1706.01807*.

Genevay, A., Peyré, G., and Cuturi, M. (2017b). Learning generative models with sinkhorn divergences. *arXiv preprint arXiv:1706.00292*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Salimans, T., Zhang, H., Radford, A., and Metaxas, D. (2018). Improving gans using optimal transport. *arXiv preprint arXiv:1803.05573*.

Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*.

A Flow diagram of the Sinkhorn AutoDiff algorithm

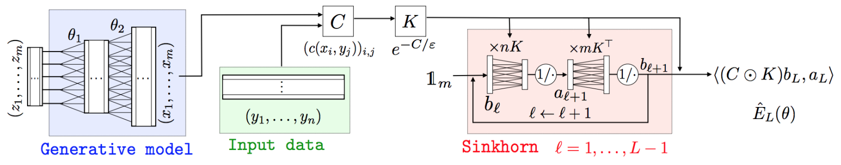


Figure 1: For a given fixed set of samples $(z_1, \dots, z_m)$, and input data $(y_1, \dots, y_n)$, flow diagram for the computation of Sinkhorn loss function $\theta \mapsto \hat{E}_\varepsilon^{(L)}(\theta)$. This function is the one on which automatic differentiation is applied to perform parameter learning. The display shows a simple 2-layer neural network $g_\theta : z \mapsto x$, but this applies to any generative model.

Figure 6: Flow diagram of the Sinkhorn-GAN. Source : Genevay et al. (2017b)

B Architecture of our networks

B.1 Generative network

Figure 7 displays the architecture of our generative network. The generative network is comprised of 3 hidden layers with ReLU activation functions and an output layer with an hyperbolic tangent activation function. The layer's sizes are 100 for the input layer (meaning that the dimension of the latent space is 100), 32, 64, 128 and 784 for the output layer. Dropout is applied during the forward pass.

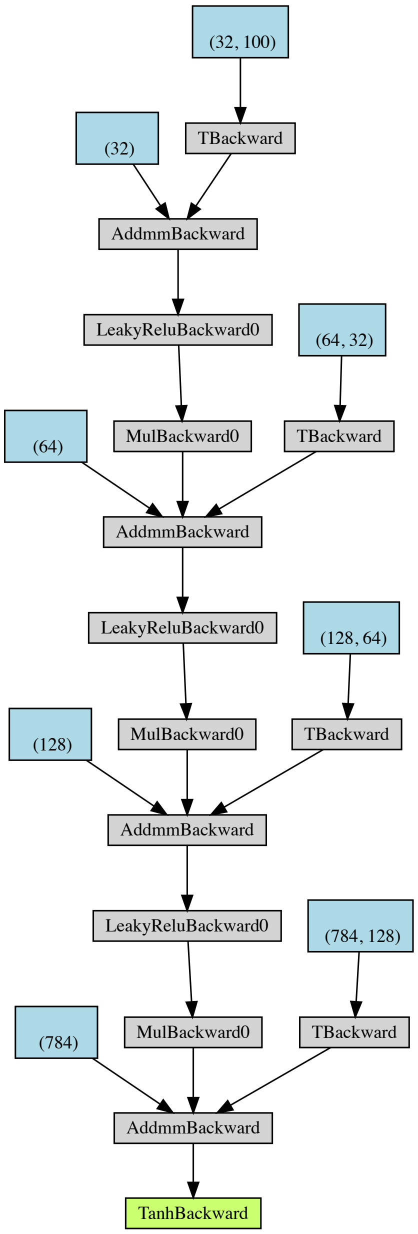


Figure 7: Architecture of the generator

B.2 Discriminator

The discriminator is symmetric and has the architecture displayed on figure 8.

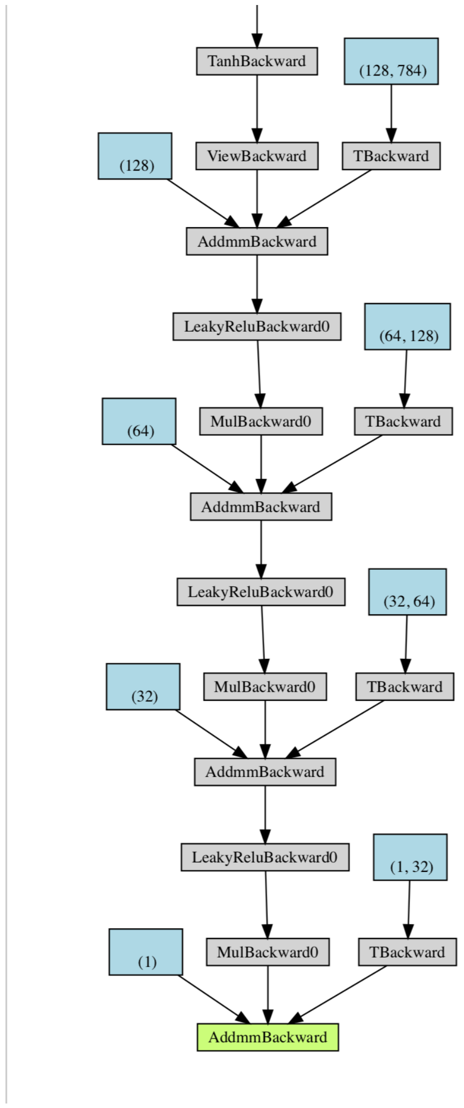


Figure 8: Architecture of the discriminator

B.3 Multilayer perceptron

Our MLP has the architecture depicted on figure 9. We also tried an alternative architecture with two fully connected hidden layers, suspecting that this may cause the bad performance of our implementation. It turned out that this was not the cause, so we stick with this architecture in our experiments.

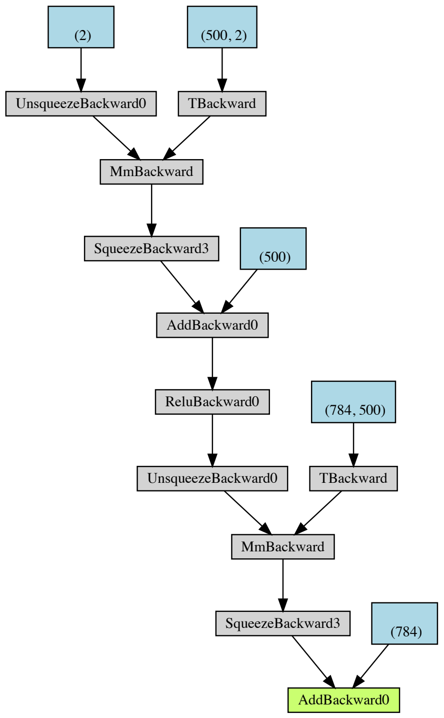


Figure 9: Architecture of the MLP

C Traning and sanity checks

In this section we provide some evidence that out networks are working properly. We first illustrate the evolution of the samples generated by our baseline GAN and then we turn to our custom made multilayer Perceptron, showing that when trained with a standard MSE loss it indeed works properly.

C.1 Baseline GAN

Figure 10 shows the improvements of our network over the 100 epochs. Row 1 presents the first epoch, where generated images are still mostly noise and each line represents a subset of generated samples every 10 epochs.

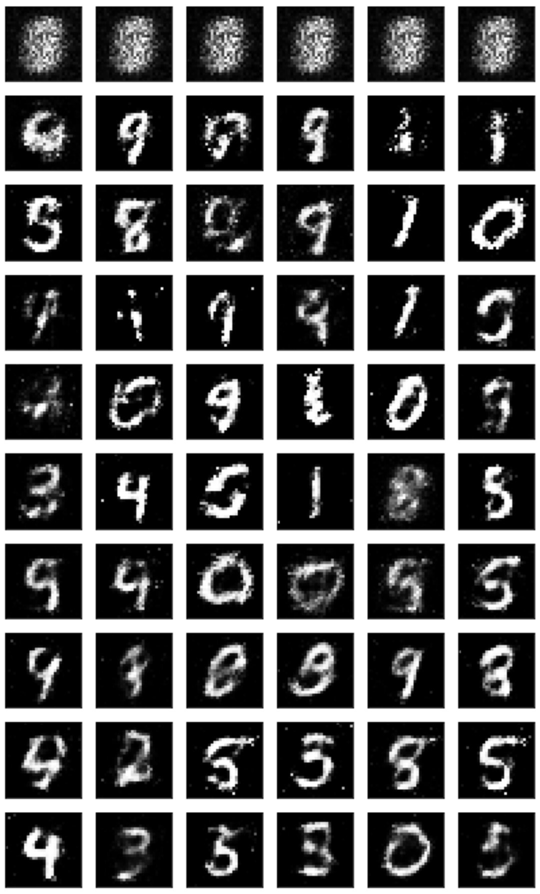
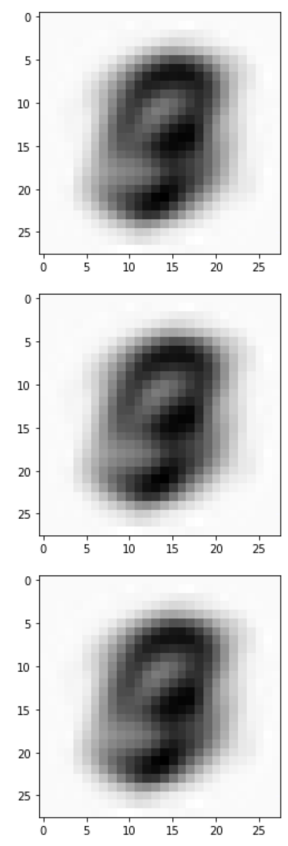


Figure 10: Generated images over the epochs

C.2 Checks on the multilayer Perceptron

In order to see whether our Perceptron was working as expected, we first trained it with a MSE loss. The results are displayed on figure 11, showing that indeed the network is working properly. The goal was therefore to replace this loss with a Sinkhorn loss that we've defined manually.
Figure 11a shows that the model converged as expected to an average image or a single image (see fig. 11b), whether it was trained on batches or on a single image. Note that each image is generated from a random point in the latent space. We can see that all points map to the same image in the image space.


(a) Traning with MSE critic on the whole dataset

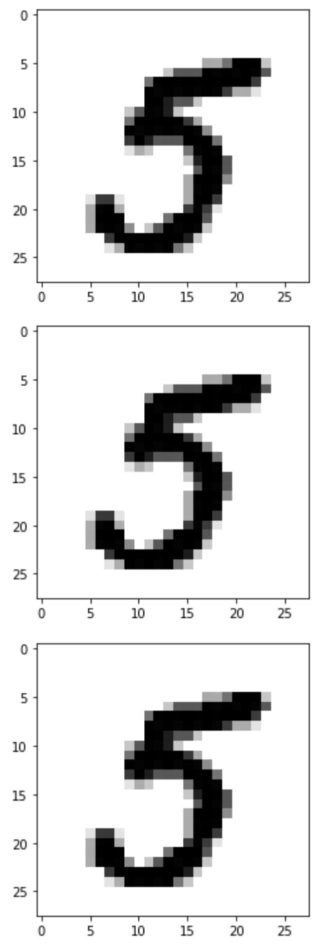

(b) Traning with MSE critic on a single image

Figure 11: Sanity checks on our architecture

D Outcomes over the experiments

this section gather figures of some outcomes obtained during our trials of replication of the results.

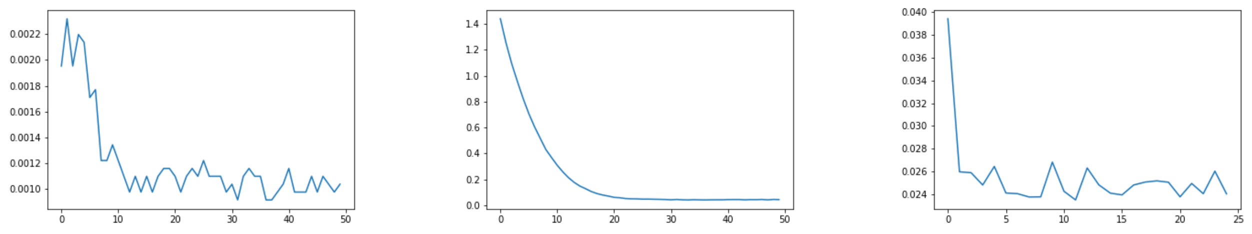


Figure 12: Shape of various losses obtained during the trials for different parametrizations

On figure 12 the losses are obtained either for various learning rates or batch sizes but they all display the same shape.

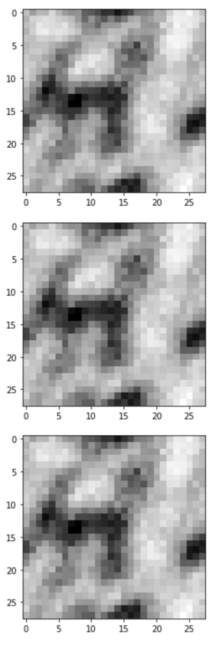
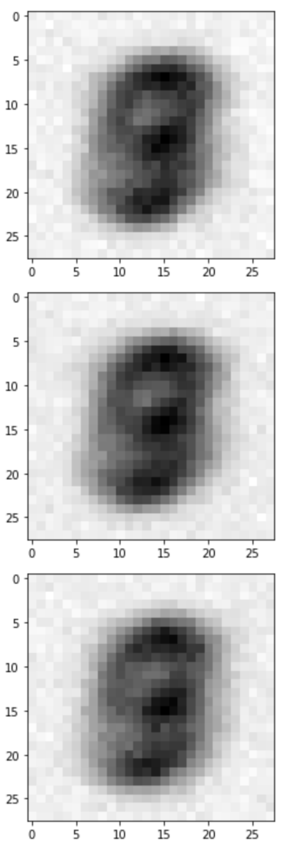
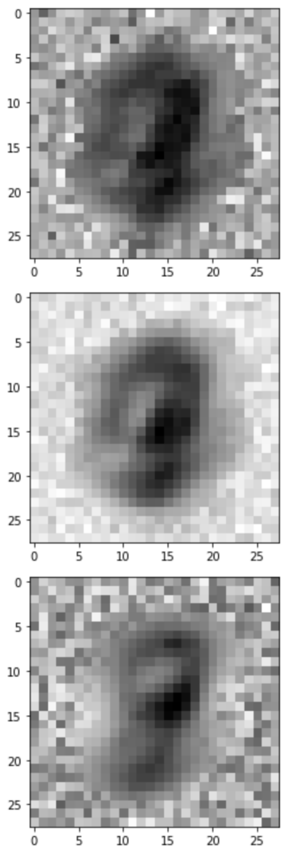


Figure 13: Patterns with an incorrect loss

Figure 13 is an example of patterns generated after training with a wrong cost matrix, computed over the size of the images (28*28 $\times$ 28*28) rather than over the batch size.



(a) Outcome with a sigmoid activation function



(b) Best outcome (with ReLU)

Figure 14: Generated samples with the Sinkhorn-GAN

We can see that the images are "smoother" with the sigmoid. However, for this parametrization, it also seems that the network generates an average image.