Generative Adversarial Networks

Generative Adversarial Networks (GANs) [1] are neural network architectures trained to produce a generative model faking real data.

In this project, I implemented several GANs architecture:

- Deep Convolutional GANs (DC-GAN) [2].
- Wassertstein GAN (WGAN) [3]
- Optimal Transport GAN (OT-GAN) [4]
- Cycle-GAN [5]

and tested it on MNIST. In this file, I give some explanations about these models.

1 Introduction

The idea of GANs goes back to [1], and consists in the training of two neural networks at the same time in order to generate data (mostly images) according to the distribution of a real dataset (for instance, MNIST).

GANs consist of a generator, g, which has to capture the data distribution, and a discriminator (or critic) d, which must estimate the probability that the data at hand comes from the training dataset (real images) or has been simulated by the generator.

In order to keep randomness in the generation, g will be a parametrized function (here a neural network) going from a **latent space** Z (of dimension usually 100) to the data space. One data point is therefore generated as $g(z)$ where z follows a standard normal distribution in the latent space Z.

During the training of the GAN architecture, g tries to fool the d, which tries not to be fooled. This game can therefore be seen as a two-player minimax problem, where:

- d tries to maximize the probability of assigning the correct label to each data point:

$$E_{x\sim p_{data}}(log(d(x)) + E_{z\sim p_z}(log(1-d(g(z))))$$

- g tries to minimize this probability, which amounts to minimizing

$$E_{z\sim p_z}(log(1-d(g(z))))$$

This gives the following objective:

$$\min_g \max_d E_{x\sim p_{data}}(log(d(x)) + E_{z\sim p_z}(log(1-d(g(z))))$$

2 Deep Convolutional GANs

In Deep Convolutional GANs [2], the main point concerns the type of architectures used for g and d. In fact, the authors advise taking Deep Convolutional Networks with a few tricks:

- no pooling layers
- use of Batchnormalization for both g and d (in my implementation, Batchnormalization in the discriminator seemed to give worse results, so I keep such layers only in the generator)
- no dense layer for deep architectures
- ReLU activation for g, leaky ReLU for d

The dataset used for my implementation was MNIST, which is a large database of handwritten digits that is commonly used for training various image processing systems. Like in the article, there was no preprocessing of the data (except that I re-normalized them to [-1, 1]), and I used the Adam optimizer of learning rate 1e-4 to train our model.

Concerning the exact architecture, I tried several variants, and kept the following one. For the Generator:

- A dense layer to go from the 100 latent space to 256 channels of 7 by 7, followed by a Batchnormalization and a LeakyReLU activation
- A Conv2DTranspose with 128 kernels of size (5, 5) and with strides of (1, 1), allowing to go to a space of size (7, 7, 128) ; with then apply a Batchnormalization layer and a LeakyReLU activation function.
- I repeat the last step 2 times with the same size of kernels, but with strides (2, 2) and with a final hyperbolic tangent activation function replacing the last LeakyReLU.

For the discriminator:

- A first convolution layer with 64 kernels of size (5, 5) with strides (2, 2) and a Leakyrelu activation function
- A second convolution layer with 128 kernels of size (5, 5) with strides (2, 2) and a Leakyrelu activation function
- A single output (Dense layer with 1 neuron) with no activation

As you can see, I did not followed exactly the recommendations of [2]. This choice was made because the recommendations of the authors of [2] gave worse results in this case.

As you can see in Figure 1, the results are quite interesting for the simple case of MNIST, but required a lot of epochs to be as performing.

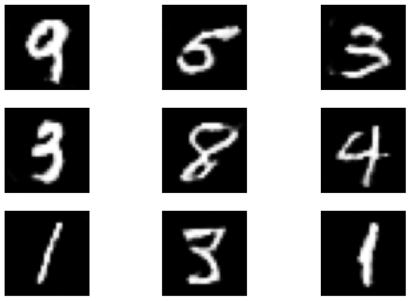

Figure 1: Images after 100 epochs (approximately 100,000 iterations) of DC-GAN

3 Wasserstein GAN

3.1 Presentation of the method

The article ”Wasserstein GAN” [3] focuses on the various ways to measure how close the model distribution and the real distribution are, or equivalently, on the various ways to define a distance or divergence on distributions. The paper makes the following contributions :

- They provide a comprehensive theoretical analysis of how the Earth Mover (EM) distance behaves in comparison to popular probability distances and divergences.
- They define a form of GAN called Wasserstein-GAN that minimizes a reasonable and efficient approximation of the EM distance.
- They empirically showe that Wasserstein-GANs cure the main training problem of GANs : the training instability.

First, the article defined a few distances including the one used for the Wasserstein GAN, called the Earth Mover (EM) distance. Let $\mathcal{X}$ be a compact metric set and let Σ denote the set of all the Borel subsets of $\mathcal{X}$. Let $\text{Prob}(\mathcal{X})$ denote the space of probability measures defined on $\mathcal{X}$. We can now define the Earth Mover distance between two distributions $\mathbb{P}_r$, $\mathbb{P}_g \in \text{Prob}(\mathcal{X})$:

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

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. The EM distance is viewed as the "cost" of the optimal transport plan.

According to the Kantorovich-Rubinstein duality [?] :

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

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$.

To find the optimal function f the neural networkis trained with with weights w lying in a compact space W and then back-propagation through $\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))]$, as we would do with a typical GAN. In order to have parameters w lie in a compact space, they propose to clip the weights to a fixed box (taken as [-0.01, 0.01] after each gradient update. Therefore, Momemtum based optimizers like Adam cannot be used, and the authors of [3] chose RMSPRop.

The fact that the EM distance is continuous and differentiable means that we can theoreticaly train the critic until optimality. This ability is one of the most compelling practical benefits of WGANs since it can continuously estimate the EM distance by training the discriminator to optimality.

In practice nonetheless, we will simply define a number of iterations to perform.

3.2 Implementation

Concerning the exact architecture of the discriminator and the generator, I kept approximately the same as for the DCGAN (adding just one convolutional layer in the discriminator since, its weights being clipped, it needs somehow to have more capacity).

I show the results in Figure 2. As you can see, we clearly recognize the different digits : two 3, one 4, one 6, one 8, one 7 and one 9. Two images could be interpreted as different numbers. Overall, the quality is satisfying, and seems better than for the DCGAN even if it trained twice as fast (I did the equivalent of 10 epochs against 100 for DCGAN, but the iterations are slower with WGAN).

4 Optimal Transport - GAN

4.1 Notation

In the following, g will be used alternatively for the generator and for the distribution of the data it generates, p will be used for the distribution of the real data, and d will still be the discriminator (concerning this particular point, my notation with the authors diverge, since they use the term critic, and write it v_η).

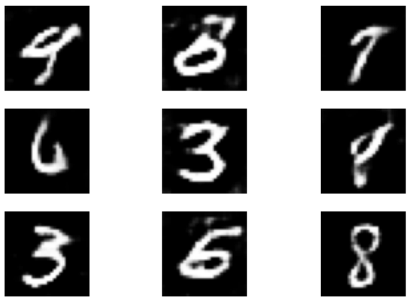

Figure 2: Images after 10,000 iterations of the WGAN, an iteration being one update on a mini-batch of 64 images (therefore, 1,000 iterations is approximately 1 epoch over the whole training dataset)

.

4.2 Motivation

The main idea motivating the use of Optimal Transport for Generative Adversarial Networks is that a good objective for the generator would be to minimize the distance between the distribution of the real data, p, and the distribution g. Defining a good distance between distributions is not easy, and has been the major concern of Optimal Transport in the past years. In fact, [3] proved that the Earth Mover Distance (EMD), or Wasserstein-1 distance, was a very good candidate for this task. This distance reads as follows:

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

where $\Pi(p,g)$ is the set of all joint distributions with marginals p and g, and c is some cost function.

Unfortunately, this distance is not so useful in practice for computational reasons, so that an approximation of it is preferred. To do this, we change the constraint to $\gamma \in \Pi_\beta(p,g)$, where $\Pi_\beta(p,g)$ is the set of joint distributions with entropy of at least β. This gives the Sinkhorn distance:

$$D_{Sinkhorn}(p,g) = \inf_{\gamma \in \Pi_\beta(p,g)} \mathbb{E}_{x,y \sim \gamma} c(x,y)$$

The question could now be concerning the calculation of the cost function. In practice, the authors advise to approximate it on mini-batches of data of K vectors for data X and Y. Denote $x_1, ..., x_K$ and $y_1, ..., y_K$ these vectors. One starts by building a cost matrix C of size $K \times K$ where :

$$\forall\ i,\ j = 1, ..., K, \quad C_{i,j} = c(x_i, y_j)$$

The joint distribution γ is replaced by a matrix M of size $K \times K$ of soft matching, ie it matches points from X to Y. It is restricted to the set $\mathcal{M}$ of matrices with positive entries, and with rows and columns summing to 1, and such that:

$$-Tr[M \log(M^T)] \geq \alpha$$

The final distance evaluated on the mini-batch in then:

$$\mathcal{W}_c(X,Y) = \inf_{M \in \mathcal{M}} Tr(MC^T)$$

In practice, M can be found efficiently using the Sinkhorn algorithm.

Based on the distance between mini-batches presented above, several losses have been studied in order to obtain the best possible gradients. The authors propose to use the so-called *Mini-batch Energy Distance* :

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

where $X,\ X'$ and $Y,\ Y'$ are independent samples from distribution p and g respectively. The authors argue that adding the repulsive term (in the sense that it forces g to be well spread) $-\mathbb{E}[\mathcal{W}_c(Y,Y')]$

to the attractive term (in the sense that it forces g and p to be similar) $\mathbb{E}[\mathcal{W}_c(X, Y)]$ allows to obtain unbiased gradients and a more stable optimization.

It remains now to define the cost function c that will be used by the authors in their algorithm. They propose, instead of using the classical euclidean distance which performs poorly in high dimensions, to have a cost function being learned during training. This is when the discriminator d (or **critic** v_η as chosen by the authors) comes into consideration. In fact, they propose to use the following cost function:

$$c(x, y) = 1 - \frac{d(x)d(y)}{||d(x)||_2||d(y)||_2}$$

where d is a neural network (also learned during training) that will map the images to a latent space where the cosine similarity will be taken.

4.3 Algorithm

I describe here some details about the algorithm used to train OT-GAN.

- Firstly: let us describe the loss function $\mathcal{L}$ that will be used in practice:

  $$\mathcal{L} = \mathcal{W}_c(X, Y) + \mathcal{W}_c(X, Y') + \mathcal{W}_c(X', Y) + \mathcal{W}_c(X', Y') - 2\mathcal{W}_c(X, X') - 2\mathcal{W}_c(Y, Y')$$

  as you can see, it is, in expectation, twice the Mini-batch Energy Distance. The generator will try to **minimize** this loss function, while the discriminator will try to **maximize** it.

- Secondly: The authors advise to train several times the generator for each update of the discrimininator. This is quite contrary to common practices, but is motivated by the fact that as soon as the cost function is not degenerate (meaning that two different points have a strictly positive cost), the goal of the generator is well defined. In the contrary, if d is trained several times without any update of g, the discriminator d could "over-fit" the distribution space and assign a null cost to some vectors in order to minimize $\mathcal{W}_c(X, X')$ and $\mathcal{W}_c(Y, Y')$ (to maximize $\mathcal{L}$). This would be problematic because then the generator could take advantage of this and in his turn overfit, which we don't want since it would probably result in mode-collapse.

- Thirdly: as the authors advise, I do not backpropagate through the Sinkhorn algorithm.

4.4 Results

I trained two different OT-GANs. The first with $n_{gen} = 5$ like the authors advise, the second with $n_{gen} = 2$ (ie alternatively updating the generator and the discriminator), the idea being that in our simple case, maybe that updating many times the generator for each discriminator update would be unnecessary.

As the results are quite similar, I present them in the notebook but give here only the results obtained for $n_{gen} = 5$. The best images I obtained are the ones proposed in Figure 3. They were obtained after only 6,000 iterations. The images I obtained after 10,000 epochs tend to be less good, but I think that it is due to the randomness of the samples, and so that it could have been otherwise.

Overall, the quality of the images obtained is satisfying, even if I trained only for 40 minutes approximately. In fact, the authors argue that the training can be very slow, and might take days for the most complicated tasks. Here, for MNIST, an optimized code could maybe be even faster than our implementation, that gives still good results.

Qualitatively speaking, the two OT-GAN I tried (with $n_{gen} = 2$ and $n_{gen} = 5$) seem relatively equal as you can see in the notebook. Nonetheless, if I had to choose, I think that I would keep $n_{gen} = 5$. In fact, the images for $n_{gen} = 5$ are more precise at the end of the training, even if the generator was trained less than for $n_{gen} = 2$ (because we do not train the discriminator and the generator at the same time).

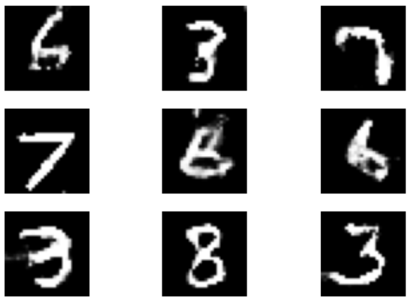

Figure 3: Images after 6,000 iterations of the OT-GAN

.

5 CycleGAN

5.1 Presentation of the method

Here, I consider a more advanced architecture that combines two GANs and which can be used to "translate" images with different characteristics, the cycleGAN. Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. The article [5] presents an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. Given training samples $\{x_i\}_{i=1}^N$ where $x_i \in X$ and $\{y_j\}_{j=1}^M$ where $y_j \in Y$, their objective was to learn a mapping $G : X \to Y$, such that $\hat{y} = G(x)$, $x \in X$, is indistinguishable from images $y \in Y$ by an adversary trained to classify $\hat{y}$ apart from y.

In addition, they introduce two adversarial discriminators D_X, D_Y and a second mapping $F : Y \to X$, where D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; and D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. They use two types of losses:

- **Adversarial loss :** used for matching the distribution of generated images to the data distribution in the target domain. They apply it to both mapping functions. For the mapping function G and its discriminator D_Y , we express the objective as:

$$\mathcal{L}_{\mathrm{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))] .$$

- **Cycle consistency loss :** used to prevent the learned mappings G and F from contradicting each other. The adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. To further reduce the space of possible mapping functions, they consider that the learned mapping functions should be cycle consistent. For each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \to G(x) \to F(G(x)) \approx x$.

$$\mathcal{L}_{\mathrm{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

The full objective is the following :

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\mathrm{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\mathrm{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\mathrm{cyc}}(G, F)$$

where λ controls the relative importance of the two objectives. The goal is then to solve :

$$G^*, F^* = \arg \min_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

5.2 Implementation

Once again, I tried several architectures for the generators. In our first implementation, I adopted the architectures described in [5], that would for instance go from a (28, 28) space to a (5, 5) one, to go back to (14, 14) space in the case of the generator from MNIST to USPS. Nonetheless, I found this method to be longer to train. The final approach I had was to do only convolutions to go from a (28, 28) space directly to a (14, 14) one. I did the reverse to go from the (14, 14) space of USPS directly to the (28, 28) space of MNIST (using only transposed convolution).

Apart from this small change (and the fact that I use only 1 residual block in each architecture), I implemented almost exactly the architectures given in [5], and I was surprised by how fast this algorithm trains. In fact, in less that 10 minutes (1000 iterations on Colab Pro) perfect images were obtained (see Figure 4).

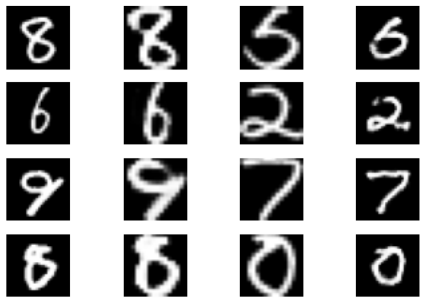

Figure 4: Images after 100 epochs of the CycleGAN.

On the first column you can see some images from the MNIST dataset, and on the second column their adaptation to USPS style by our algorithm. Similarly, the third column consists of images from the USPS dataset, and the last column is the translation of the third column to MNIST's style.

6 Conclusion

When comparing the different GANs, the improvement in terms of quality seems interesting quite stable across the different architectures.

Overall, even if MNIST allows to see some differences between them, I think that for the choice between the GANs presented here would require a more complex task.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Alec Radford and Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[4] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport, 2018.

[5] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.