Data Science Lab

Thomas Boudras, Augustin Gervreau, Medhi Inane

GAN Project

Report of Assignment 2
Team God

November, 14, 2023

This is a report detailing our work for a project aiming to implement Generative Adversarial Networks (GANs) for image generation, with training on the MNIST dataset. The following will explain our experimental approach to this problem.

1 Context of Generative Adversarial Network

We briefly recall the context of the assignment and set the notations for the future explanations.
We find ourselves in an image generation problem, for which we set up a generative adversarial network. The aim of this type of algorithm is to pit two neural networks against each other. The generator aims to generate sufficiently realistic images, while the discriminator aims to detect the difference between a real image and a generated one. This is summarized in the following min-max problem :

$$\min_G \max_D \mathbb{E}_{x\sim P}[\log D(x)] + \mathbb{E}_{x\sim \hat{P}_G}[\log(1 - D(x))] \tag{1}$$

where P is the data distribution, $\hat{P}_G$ the model distribution induced by the generator function G and D the discriminator function.

The constraint of this project relied on the fact that the architecture of the generator, which consisted of a simple Feed Forward Neural network of 4 Linear layers, is fixed. Thus, we focused our work on investigating improvements for this GAN, namely :

- Improving the performance of the VanillaGAN.
- Changing the loss function of the problem to a smoother one, based on the Earth Mover's distance. This is refered to as Wasserstein GAN (W-GAN)[1], constraining the discriminator to the space of 1-Lipschitz functions.
- Visualizing the performance of the discriminator of the W-GAN to check if the discriminator remains 1-Lipschitz, by using an upper bound based on its spectral norm
- Adding convolutional layers to the discriminator, showing that a better performing discriminator significantly improves the training of the generator in W-GANs.

In the experiments we did with the different methods, we compared only with the FID and did not look much into precision and recall. Thus every improvement have been aimed at reducing FID.

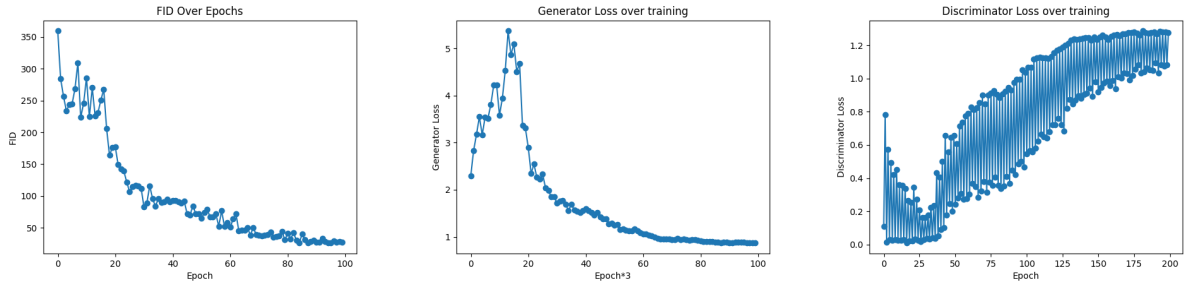
2 Vanilla GAN

We refer to VanillaGAN as the provided architecture, consisting of two competing feed forward neural networks of 4 linear layers, with LeakyReLU activations and a sigmoid output for the discriminator. The loss function for both is the binary cross-entropy loss, with Adam optimizer. As a first investigation step, we wanted to tune the discriminator and the training to ensure convergence for the VanillaGAN, before investigating other techniques. Thus, we tried the following changes:

- To counter collapse mode and prevent overfitting from the discriminator, we integrated dropouts into the discriminator architecture after each layer. This was an effective improvement of the model.
- To counter the relative simplicity of neural networks and try to avoid the mode collapse, we decided to train the generator model only every n epochs. With vanilla GAN, we obtain a better FID score when training the generator every two epochs.
- Adding batch normalization did not help improve the performance.

- The most critical improvement made to our model was differentiable data augmentation [4]. The augmentations consisted in modifying the images inputted to the discriminator, in both generator and discriminator training. We reused the code from [4][1], opting for translation and cutout.

This has allowed our FID to go from 385 with the base architecture to an FID of 42 without augmentation, to an FID of 27 with augmentation. Although significant improvements were made, the discriminator displays instability, as shown in figure 3c. The code for the best version of VanillaGAN can be found in the branch vanilla_data_augment vanilla_data_augment.



(a) FID score (b) Generator loss (c) Discriminator loss

Figure 1: Training progression for improved WGAN

3 Wasserstein GAN

The main difference between the Vanilla GAN and the Wasserstein GAN (WGAN) is the loss function used.

Indeed, WGAN make use of the earth's movers distance in the loss to encourage the distribution of generated images ($x \sim \hat{P}_G$) to approximate the true distribution.

Thus, the loss from (1) becomes the Wasserstein-1 GAN objective function [2]:

$$min_G max_{D,Lip(D)<1} \mathbf{E}_{x\sim P_r}[D(x)] + \mathbf{E}_{\hat{x}-\sim P_g}[D(\hat{x})] \tag{2}$$

The code of a WGAN with no normalization is present in branch LIP_noth.

Note we tried every implementation presented with some dropout and never got any improvement. This may be because dropout leads to slower training of the discriminator which did not help us in our case. To test this for general WGANS we would have had to exit the scope of mnist.

In practice, what changes from vanilla GAN is the output of D which is linear rather than sigmoïd. Thus the output is a score, the discriminator is therefore called the *critic*, and the expectation is approximated as the mean of the outputs from the discriminator (no issues with backpropagation). Furhermore, the new optimization objective (2) restrains the discriminator to smoother 1-Lipschitz functions. Keeping a neural network to act as *critic*, which in practice is not necessary since we only need to explore the family of 1-Lipschitz functions, is rather good in the sense that they can approximate many functions. Therefore the appropriate *critic* at each training iteration can be approximated as a neural network (Universal approximation theorem, even with such a restricted model).

However, the difficulty lies in constraining the model to choose only among the set of 1-Lipschitz functions, and several methods more or less successful have been proposed. We can see on figure 2a that the unconstrained problem leads to bad functions in terms of lipschitzness, since we get exponential increase

[1] `https://github.com/mit-han-lab/data-efficient-gans`

in the Lipschitz constant and training does not happen. This is expected since the wasserstein distance is not supposed to work with arbitrary functions.

3.1 Regularization

Weight clipping The initial papers came with the idea of weight clipping: all parameters are constrained to $[-c, c]$[1]. This is interesting as clipping the weights of the parameters to a compact set effectively bounds the Lipschitz constant. We visualize it in figure 2b where we can see it remains constant to a certain extent.

A main drawback of this method is the loss of expressivity of D which, is too constrained. Furthermore, backpropagating then clipping the weights does not yield good results as it may not be possible to do a good step in the direction looked at. This leads to difficulties in training, with the hyperparameter c which is very difficult to tune in practise.

In theory, weight clipping has also high chances of leading to vanishing or exploding gradients [2], experimentally, we often had some mode collapse using this method.

Code for WGAN with weight clipping can be found in branch LIP_clip

Gradient penalty A new way to enforce the constraint was proposed: Gradient Penalty[2]. The idea is to switch to a soft constrain by adding a regularization term to the gradient instead of clipping the weights. The loss for D has the added term $\mathbf{E}_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2]$ (where $P_{\hat{x}}$ the uniform distribution on the straight line between samples, meaning $\hat{x} = \epsilon x + (1-\epsilon)\widetilde{x}$). The goal is therefore to penalize the *critic* on the support of both the distributions. We can see in figures 2c, 2d and 2e that the bounding is not as efficient as the bounding of the weight clipping but since we are actually able to train and get a good FID (as low as 8 on our machine), we understand that this method constrains sufficiently while letting training happen. The code for gradient penalty with normalized can be found in LIP_grad_penalty and the one without normalized weights is in LIP_normal. We have used the gradient penalty function from [2]

3.2 Tracking the Lipschitz continuity during the optimization

During training, we have also tested the efficiency of the regularization techniques in terms of constraining the critic to 1-Lipschitz functions. To do so, we track the following upper-bound on the Lipschitz constant of D. The proof can be found in the appendix.

Proposition 1. *Let $M_1, M_2 ... M_k$ a sequence of layers in a neural network f, such that $M_i \in \mathbb{R}^{m*d}$. We have that $\forall x, y \in \mathbb{R}^d$, $\frac{||f(x)-f(y)||}{||x-y||} \leq \prod_{i=1}^{k} ||M_i||_2$, where $||.||_2$ denotes the spectral norm of a matrix.*

A way for us to approximate the Lipschitz constant is to mimic the monitoring of [3] for the spectral bounding regularization. Ideally, we would like to have $\prod_{i=1}^{k} ||M_i||_2 \leq 1$, enforcing the 1-Lipschitz constraint on D. This upper bound is not always smaller than the Lipschitz constant, and serves primarily as a visualization tool for Lipschiz continuity for us.

Proof. We first denote the discriminator expression : For an input image x, let

$$D(x) = \sigma_k(M_k(...\sigma_2(M_2(\sigma_1(M_1 x))) \tag{3}$$

where each σ_i denotes an activation function, in our case a leakyRELU. Note that this proof is general and considers both cases where our discriminator consists of convolutional or linear layers. Note that leakyRELU is already a Lipschitz function, thus we can disregard it without loss of generality. Thus, let $D(x) = M_k ... M_1 x$. We have that :

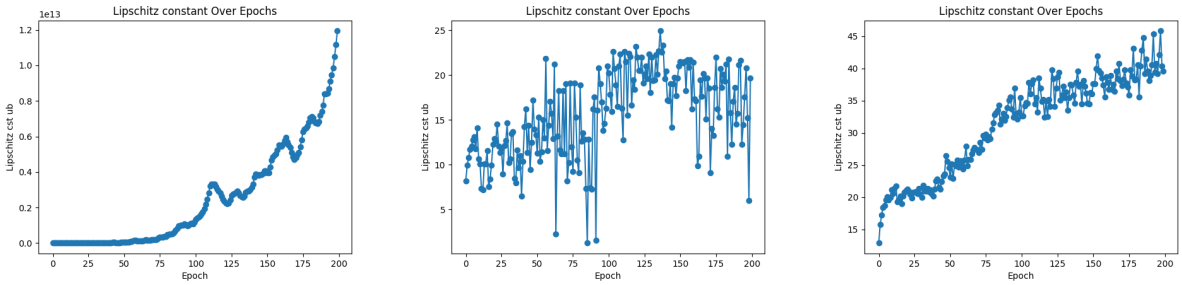
$$||A||_2 = max_{||x||_2=1} ||Ax||_2 = max_i \lambda i \tag{4}$$

[2] https://github.com/igul222/improved_wgan_training

where λ is a singular value of A Now, consider two images x and y :

$$||D(x) - D(y)||_2 = ||\prod_{i=1}^{k} M_i(x-y)||_2 \leq \prod_{i=1}^{k} ||M_i||_2 ||x-y||_2$$

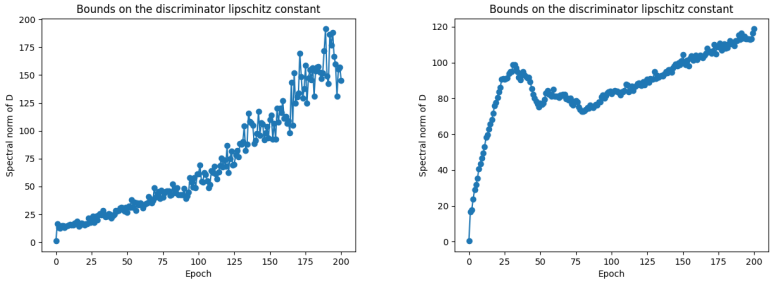
Thus, $\frac{||D(x)-D(y)||_2}{||x-y||_2} \leq \prod_{i=1}^{k} ||M_i||_2$ Now, we would like to check if this upper bound to be always less than the Lipschitz constant, in this case 1. This is what we do in our code by computing the spectral norm of each layer, then computing the product. We can visualize the results in figure 2. Note that this upper bound is not always less than the Lipschitz constant, and only serves us visually to track the regularity of the discriminator. [3] used a similar approach to normalize the weights by an upper bound of a spectral norm during the training steps, which gave a better convergence than traditional WGAN. Finally, we reshape the convolutional layers : if C has dimensions (out,in,h,w), then it is reshaped to (out,in*h*w) □



(a) Lipschitzness for WGAN without regularization

(b) Lipschitzness for WGAN with weight clipping

(c) Lipschitzness for WGAN with gradient penalty



(d) Lipschitzness for WGAN with gradient penalty - weights normalized in initialisation

(e) Lipschitzness for convolution + WGAN with gradient penalty - weights normalized in initialisation

Figure 2: Lipschitz constant for WGAN

In this case, we can see that the proposed upper bound does not go below one, which was expected as no normalization is done during the training, as opposed to spectral bounding [3]. This could be a further improvement to our project. However, We observe slight changes in this upper bound as the models approach convergence. Lastly, we have also tried to normalize the weights in the initialization to invetsigate if the gradient penalty keeps the upper bound below one, but after one epoch it went to be greater than 1.

3.3 Convolutional discriminator

Now that we have an efficient Wasserstein GAN, it would be interesting to set up a more powerful discriminator than the one initially provided. This is a general statement of WGANs that better discriminators

leed to better training of the generator, however training was way too long if we implemented this by training D more epochs than G (as we have to double or triple training time to get the same number of generator updates), so we decide to set up a convolutional discriminator that will be more powerful than the linear one we had until now. Code for this implementation is situated in branch main.
The network is composed of four successive blocks, each comprising a convolution layer followed by a LeakyReLU activation with a slope coefficient of 0.2. Each convolution layer reduces the spatial resolution of the input while increasing the number of channels. The first block has a single convolution layer with 16 filters, while subsequent blocks progressively increase the number of filters convolution layer.
We had initially tried with dropouts at the end of each block, but we've noticed that it's more efficient without dropouts.

This type of discriminator is already widely used in auxiliary classifier GAN (AC-GAN) (SOURCE). We have derived and adapted this model from one of these GAN discriminators.

This discriminator, combined with WGAN and slightly optimized hyperparameters, gives us an **FID of 15.4** on the server (5 on our machine), with a **recall of 0.25** and an **precision of 0.51**. We have also tried to combine it with differentiable data augmentation, but the performance was quite similar. Therefore, we decided to opt for the model that gave us the best FID, which is the one without augmentation.

4 Results

(a) FID score (b) Generator loss (c) Discriminator loss

Figure 3: Training progression for improved WGAN

Our work thus corroborates our statement : without data augmentation, we obtain a better FID, while the training seems to be more stable from the Wasserstein distance and the gradient penalty.

5 Conclusion

After improving the GAN Vanilla model, we set up several alternatives to the Wassertsetin method. The latter, combined with a convolutional discriminator and data augmentation, enabled us to obtain the best results on all our tests. During this hands-on session, part of the group learned how to use LAMSADE's ssh servers and the tools that go with them (vim, nohup, etc.).

We were also able to gain an in-depth understanding of GAN architecture, and in particular of the phenomena that occur during training, which can affect performance.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[2] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[3] Zhihong Zhang et al. "Spectral bounding: Strictly satisfying the 1-Lipschitz property for generative adversarial networks". In: *Pattern Recognition* 105 (2020), p. 107179. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2019.107179. URL: https://www.sciencedirect.com/science/article/pii/S0031320319304790.

[4] Shengyu Zhao et al. *Differentiable Augmentation for Data-Efficient GAN Training*. 2020. arXiv: 2006.10738 [cs.CV].