Université de Genève

S411015: Multivariate Analysis

Understanding GANs and associated models

Authors

(20-336-988) González, Marc
(19-321-488) Muresan, Iulia-Maia

PROFESSOR

Sebastian Engelke

Research Center for Statistics
GSEM

ABSTRACT

We introduce the Generative Adversarial Network (GAN) framework and analyze its different components from a theoretical point of view. After understanding its limitations, we discuss Wasserstein GANs and explore how they improve over the base model. Then, some other modifications and applications of relevance are investigated. An experiment using WGAN over several image datasets is run to show its properties.

Geneva, 25.03.2021

Introduction

The advent of **neural networks** has revolutionized most of the fields within the discipline of statistical learning, and the approximation of probability density functions is no exception to this phenomenon. In particular, Goodfellow et al. [1] introduced in 2014 a class of models that demonstrated widespread success in performing this task: the Generative Adversarial Networks, often abbreviated as GANs. Their good performance extends to many fields such as *natural language processing* or *time series synthesizing*; however the field where they have displayed the most outstanding improvements is that of *computer vision*, where among other tasks they excel at image generation and image-to-image translation, for example [2].

This success has sparked a research race to explore how to modify this initial GAN formulation in order to get **more refined** or case-adaptive versions with an even better performance; or at least counting with some desirable properties. One of these offshoots was the **Wasserstein GAN** proposed by Arjovsky, Chintala and Bottou [3], an adaptation born from the careful study of how to improve the **stability** of the algorithm, and also the original inspiration source for this report. There have been however many others proposed, with varying degrees of innovation and performance.

This story has motivated us to take on a challenge: to study and expose these Generative Adversarial Networks, and unravel their mysteries. Our goal is to step into what we consider to be a current and interesting area of research and understand where it might be implemented. As the actor Anthony Hopkins says in his role of Robert Ford in the fictional series *Westworld*:

"Everything in this world is magic, except to the magician."

Thus, the objective of this report is nothing else than to give an overview of what this intriguing Generative Adversarial Network framework is and explore its ecosystem, with the particular aim of developing insights on the subject. In particular, we will:

1. **Formalize the problem** of maximizing the likelihood function estimation for some data.
2. **Define what is a GAN** and how it can address the previously introduced problem.
3. **Analyze theoretical properties** of GANs of particular interest.
4. **Introduce the two main issues** concerning GANs: vanishing gradients and mode dropping, as well as comprehend why do they emerge.
5. **Explain how changing the evaluation metric** through the EM algorithm / Wasserstein-1 distance may facilitate the convergence of the now WGAN model. Also find out the implications of this innovation. What advantages does it offer? What problems do still require of further work?
6. **Explore some developments** in the literature beyond the WGAN.
7. **Dive into specific applications** of GANs over various problems and disciplines.
8. **Discuss our own implementation** of the WGAN version of the algorithm.

Afterwards, a **conclusion** with the main takeaway points from the whole study will close the report.

1 Learning the probability distribution of data

As Arjovsky, Chintala and Bottou already introduce [3], one of the main problems concerning the field of unsupervised learning is that of explicitly stating the generating process for some given data. In particular, this is done by approximating the probability density function.

$$\widehat{F}(y) = \frac{1}{m}\sum_{i=1}^{m} 1\left\{x_{i1} \leq y_1, \dots x_{ip} \leq y_p\right\} \tag{1}$$

This apparently simple formulation leads to what in fact is a very complex and non-trivial problem: just as we can approximate the cumulative distribution function using the **empirical distribution function** (as in Equation (1)[1]), **there is no general method for estimating the probability density function** itself without further assumptions.

So, in short, what are these assumptions? As the authors mention, if the real data distribution admits a density and we can parametrize this density using any sort of function P_θ, the problem can be expressed in terms of:

$$\max_{\theta \in \mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log P_\theta\left(x_i\right) \tag{2}$$

Or in simple words, maximizing the (log) Likelihood of the data. Furthermore they mention that maximizing this likelihood corresponds, asymptotically, to **minimizing the Kullback-Leibler divergence** between the real generating process and whatever parametrization has been chosen. However, they do not describe how does this correspondence take place, so we derived it in the proof of Theorem (1.1)[2].

Theorem 1.1 (MLE and KLD correspondence)**.** *If the real data distribution $\mathbb{P}_{data}$ admits a parametrized density P_θ, then asymptotically for $m \to \infty$, solving the maximization problem defined in Equation (2) is the same as minimizing the Kullback-Leibler divergence $KL(\mathbb{P}_{data} \| \mathbb{P}_\theta)$.*

Proof. We can write:

$$\begin{aligned}
\lim_{m\to\infty} \max_{\theta \in \mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log P_\theta\left(x_i\right) = \max_{\theta \in \mathbb{R}^d} \lim_{m\to\infty} \frac{1}{m}\sum_{i=1}^{m} \log P_\theta\left(x_i\right) \implies & \text{ By the Law of Large numbers} \\
\implies \max_{\theta \in \mathbb{R}^d} \mathbb{E}_{\boldsymbol{x}\sim P_{data}}[\log(P_\theta(\boldsymbol{x}))] = \max_{\theta \in \mathbb{R}^d} & \int_x P_{data}(\boldsymbol{x}) \log\left(P_\theta(\boldsymbol{x})\right) dx \\
= \min_{\theta \in \mathbb{R}^d} - & \left(\int_x P_{data}(\boldsymbol{x}) \log(P_\theta(\boldsymbol{x})) dx\right)
\end{aligned}$$

Which is close to the form of the KLD. Given that we are minimizing over θ, the outcome of this operation is not affected if we add another term independent of θ. By doing so, and then applying the definition of the KLD, we obtain:

$$\begin{aligned}
\min_{\theta \in \mathbb{R}^d} - \left(\int_x P_{data}(\boldsymbol{x}) \log(P_\theta(\boldsymbol{x})) dx\right) &= \min_{\theta \in \mathbb{R}^d} \int_x \left(P_{data}(\boldsymbol{x}) \log(P_{data}(\boldsymbol{x})) - P_{data}(\boldsymbol{x}) \log(P_\theta(\boldsymbol{x}))\right) dx \\
&= \min_{\theta \in \mathbb{R}^d} \int_x P_{data}(\boldsymbol{x}) \log(\frac{P_{data}(\boldsymbol{x})}{P_\theta(\boldsymbol{x})}) dx \\
&= \min_{\theta \in \mathbb{R}^d} KL(\mathbb{P}_{data} \| \mathbb{P}_\theta)
\end{aligned} \tag{3}$$

Thus demonstrating the asymptotic equivalence between the two.

■

[1] Stated in Slide 18 of Module 1 from S411015: Multivariate Analysis.

[2] Just to note, this is not the first time this has been proved, but we have done so in terms of our own understanding.

2 The GAN framework

In the previous section, we exposed the problem of getting a good estimate of the data generating process and asserted that maximizing the likelihood of a parametric function P_θ is asimptotically equivalent to minimizing $KL(\mathbb{P}_{data} \| \mathbb{P}_\theta)$, yet provided no particular method to actually solve either process.

As the astute reader will already have deduced, this report focuses on one proposed method to tackle this: the Generative Adversarial Network.

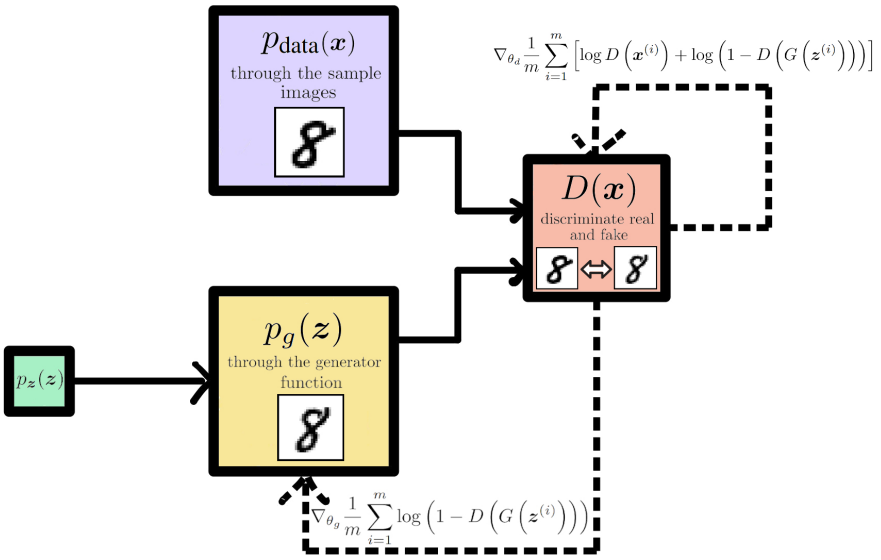


Figure 1: Pipeline of a classic Generative Adversarial Network

Before jumping into the mathematical details, we think convenient to have a **conceptual understanding** of what is a GAN doing and which are its core components. In Figure (1) we sketch, in a simplified way, the workflow of this base model using as an example some hand-written eights from the MNIST dataset, that we ran through the WGAN variation (further details provided in *Section 8*).

As can be seen, in a nutshell a Generative Adversarial Network is a framework where two Neural Networks compete in an **adversarial game**, or, against each other. These two Neural Networks are designed with very specific tasks in mind. They are usually labeled as:

- **Generator** · A **perceptron** that will accept a **random prior** $\boldsymbol{z} \sim p_{\boldsymbol{z}}$, usually drawn from either a uniform or gaussian distribution with number of dimensions corresponding to the latent space arbitrarily chosen. Its task will then be to **upscale** this input vector into a tensor containing information for every particular pixel of an image (several times if the image has color channels), in the case of *computer vision*, or any other data shape for the problem at hand.
- **Discriminator** · Also a **perceptron**, it will label an input depending on whether it comes from the real sample of images or from the data created by the generator. Conversely to the generator, it will do so by accepting the tensor containing information of the data in the original shape in which is generated (again, the pixels of an image for example), and then **downscale** them until they output a value assessing whether the input is *'real'* or *'fake'*.

Both components will **optimize each other**, as their loss functions operate under **zero-sum** game conditions: when the discriminator learns to better classify, the generator has a worse loss, and vice-versa. That is, both will be optimized in terms of how well is the other performing. Goodfellow ([1]) provides a very nice example using counterfeiters for the generator and the police for the discriminator in his original publication. The **outcome** of this should be, if the GAN is well stabilized, sample data following a distribution close to the real one in order to try to fool a very *'refined'* discriminator. In *computer vision*, this would translate to high-quality images.

3 Mathematical definition and properties

Generative Adversarial networks require inputs at two different steps of the process: firstly, $\boldsymbol{z} \sim p_{\boldsymbol{z}}$ which is then used as input in $G(\boldsymbol{z}; \theta_g)$, a differentiable function in the form of a multilayer perceptron which will map it to the data space.

Then, either the outcome of G or a real sample $\boldsymbol{x}$ is passed through a second multilayer perceptron $D(\boldsymbol{x}, \theta_d)$ that accepts input in the structure of the data and outputs a scalar, indicating whether D classifies $\boldsymbol{x}$ as coming from the real data and not from $G(\boldsymbol{z}; \theta_g)$.

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{4}$$

In the end it is possible to mathematically express the *'objective'* of this *'game'* between generator and discriminator in terms of the value function $V(G, D)$ stated in Equation (4). The authors, however, do not explain the origin of this value function at any moment and take it for granted. Therefore, in the upcoming **Section 3.1** we expose the rationale behind it.

3.1 Obtaining the GAN value function

For reasons hopefully clear at this point, the discriminator is designed to perform a binary classification between real and fake samples, for whatever distribution the generator is trying to grasp.

$$L(\hat{y}, y) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}); \qquad \hat{y} = \text{predicted value}, \quad y = \text{true sample label}$$

The GAN value function is tightly related to a particular loss function for classification: **Binary cross-entropy**, taking the form stated above. We distinguish between $D(G(\boldsymbol{z}))$, the output of the discriminator given a generator-produced sample $G(\boldsymbol{z})$; and $D(\boldsymbol{x})$, the output of the discriminator when the real data $\boldsymbol{x}$ is given. For **correctly classified** samples, the output should be $D(G(\boldsymbol{z})) = 0$ and $D(\boldsymbol{x}) = 1$.

To elaborate **on the discriminator side**, if it receives as an input some real data $\boldsymbol{x}$, then we can see from the binary cross-entropy function that what we want to predict is $\hat{y} = D(\boldsymbol{x})$ and the real value is $y = 1$. Then the loss-function for the discriminator becomes:

$$\begin{aligned} L(D(\boldsymbol{x}), 1) &= 1 \cdot \log(D(\boldsymbol{x})) + (1 - 1) \cdot \log(1 - D(\boldsymbol{x})) \\ &= \log(D(\boldsymbol{x})). \end{aligned}$$

On the other hand, if what the discriminator receives as as input is $G(\boldsymbol{z})$, the predicted value will be $\hat{y} = D(G(\boldsymbol{z}))$ and the label is $y = 0$, and thus

$$L(D(G(\boldsymbol{z})), 0) = \log(1 - D(G(\boldsymbol{z}))).$$

The discriminator needs to maximize the $\log(D(\boldsymbol{x}))$, and we know that the logarithm is a monotonic function, meaning that maximizing $D(\boldsymbol{x})$ will be sufficient to maximize the log. On the other hand, the discriminator will also optimize $\log(1 - D(G(\boldsymbol{z})))$. However, remembering that the logarithm is negative for values smaller than 1, the discriminator will need to minimize $(1 - D(G(\boldsymbol{z}))$ instead. The final loss function for the discriminator given a single sample is:

$$\max_D \log(D(\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z}))). \tag{5}$$

On **the side of the generator**, we remind that it is in competition against the discriminator, meaning that while the discriminator will benefit from scoring $D(G(\boldsymbol{z})) = 1$ the generator will in contrast seek $D(G(\boldsymbol{z})) = 0$. Analogously to the loss function of the discriminator, the generator will operate over

$$\min_G \log(D(\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z}))). \tag{6}$$

As a remark, the generator will minimize the loss function , meaning that it will maximize $D(G(\boldsymbol{z}))$, that is, trying to **worsen the score of the discriminator**. We also point out that the generator is not dealing with real data $\boldsymbol{x}$, so in Equation (6) the generator is not controlling the first term of the loss function.

Combining Equation (5) and Equation (6), we obtain the **value function for a single sample**:

$$\min_G \max_D \log(D(\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z}))) \tag{7}$$

However, we need to escalate this performance metric in Equation (7) to the whole sample space (the input data) at once. One of the most standard ways to do so is through the **expectation** operator, which will asymptotically correspond to the sample average (the number of times we measure this loss, for every element in the data). Therefore, we consider the loss function for the entire sample space by taking the expectation of Equation (7).

We denote here $v = \log(D(\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z})))$ for the loss function of Equation (7).

$$\mathbb{E}(v) = \mathbb{E}[log(D(\boldsymbol{x}))] + \mathbb{E}[log(1 - D(G(\boldsymbol{z}))]$$

Under the assumption of continuity of v, this translates to

$$\begin{aligned} V(G, D) = \mathbb{E}(v) &= \int_{\boldsymbol{x}} P_{data}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_z P_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(G(\boldsymbol{z})))dx \\ &= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data }(\boldsymbol{x})}}[\log(D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_{\boldsymbol{z}(\boldsymbol{z})}}[\log(1 - D(G(\boldsymbol{z})))] \end{aligned}$$

Which corresponds exactly to the value function that appears in the two-player mini-max game stated in Equation (4), proposed in [1].

3.2 Optimizing the neural networks

For the base GAN model, the overall **training process** is rather simple to understand.

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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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.

Algorithm (1), sourced from [1] illustrates how to train a GAN by using stochastic gradient descent. Just as a side comment, and a curiosity to the reader, this is close to the implication that the components of GANs behave approximately like a Kernel Machine under this base setting [3].

[3] To the interested reader, we encourage to check https://arxiv.org/abs/2012.00152 for the details of this statement.

Afterwards, they discuss the optimality of the discriminator D given a fixed G.

$$D_G^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

This optimal discriminator has a straightforward proof in the publication. The conclusion, and a point of interest for us, is that the generator loss in terms of an optimized discriminator becomes:

$$C(G) = \max_D V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log\left(\frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right)\right]. \quad (8)$$

This cost function based on $D_G^*(\boldsymbol{x})$ is then used to launch their main theorem, justifying the **global minimum** of $C(G)$. In what follows, we will pose *their* theorem along with an expanded proof covering on details that they decided to omit in their shortened version.

Theorem 3.1 (Global optimality of the generator)**.** *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $log(4)$.*

Proof. Knowing that $C(G) = \max_G V(G, D_G^*) = -log(4)$ when $p_g = p_{data}$, we replace $D(x)$ with $D_G^*(x) = \frac{1}{2}$ in the expression of the Generator's loss function.

$$\begin{aligned} V(G, D_G^*) &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log D_G^*(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log\left(1 - D_G^*(\boldsymbol{x})\right)\right] \\ &= \int_x p_{data}(\boldsymbol{x})\log(1/2) + p_g(\boldsymbol{x})\log(1/2)dx \\ &= -log(2)\int_x p_{data}(\boldsymbol{x})dx - log(2)\int_x p_g(\boldsymbol{x})dx \\ &= -2log(2) = -log(4) \end{aligned}$$

This value is a candidate for the global minimum, since it occurs when $p_{data} = p_g$. Then, we would like to show that it is the minimum value that $C(G)$ can take when fixing for the optimal discriminator. We will drop the assumption that $p_{data} = p_g$ in order to observe that for any G we can plug in the optimal value of the discriminator D_G^* into $C(G) = \max_G V(G, D)$. Also, in red there are highlighted some added terms that sum up to 0 and are useful for derivation's purposes:

$$\begin{aligned} C(G) = \int_x &(\log(2)p_{data}(\boldsymbol{x}) - \log(2)p_{data}(\boldsymbol{x})) + p_{data}(\boldsymbol{x})\log\left(\frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) \\ &+ p_g(\boldsymbol{x})\log\left(\frac{p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + (\log(2)p_g(\boldsymbol{x})\boldsymbol{x}\log(2)p_g(\boldsymbol{x}))\,dx \end{aligned}$$

By rearranging some terms:

$$\begin{aligned} C(G) = \int_x &p_{data}(\boldsymbol{x})\left(log(2) + \log(\frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) - \log(2)\left(p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})\right) \\ &+ p_{\boldsymbol{x}}(x)\left(log(2) + \log\left(\frac{p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right)\right)dx \end{aligned}$$

Finally, putting everything together,

$$\begin{aligned} C(G) &= -log(2) \int_x p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x}) dx + \int_x p_{data}(\boldsymbol{x}) \log \left(\frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) dx \\ &+ \int_x p_g(\boldsymbol{x}) \log \left(\frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) dx \\ &= -2log(2) + \int_x p_{data}(\boldsymbol{x}) \log \left(\frac{p_{data}(\boldsymbol{x})}{\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \right) dx \\ &+ \int_x p_g(x) \log \left(\frac{p_g(\boldsymbol{x})}{\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \right) dx, \end{aligned}$$

And then re-expressing in the form of the Kullback-Leibler divergence, we see that:

$$\begin{aligned} &= -log(4) + KL \left(p_{data} \| \frac{p_{data} + p_g}{2} \right) + KL \left(p_g \| \frac{p_{data} + p_g}{2} \right) \\ &= -log(4) + 2 \cdot JSD(p_{data} \| p_g) \end{aligned}$$

Both the KL and the JS divergence are always non-negative, so the lower bound and thus global minimum of C(G) will correspond to $-log(4)$, when both densities match and the corresponding divergences evaluate to 0. This can only happen when $p_g = p_{data}$ (The JS divergence is the square of the JS distance metric, which by definition has this property), meaning that there is a single parameter combination point that can calibrate the generator to reach its global optimum at $C^*(G) = -\log(4)$. ■

As a final remark on this, the paper states that for **enough capacity** of G and D, (and we understand by that a well suited specification of both models' architecture over the data), and by allowing the discriminator to be sufficiently trained to reach the optimum $D^*_G(\boldsymbol{x})$, then p_g converges to p_{data}. Their demonstration of this proof is comprehensive and thus we do not discuss it here.

4 Shortcomings of GANs

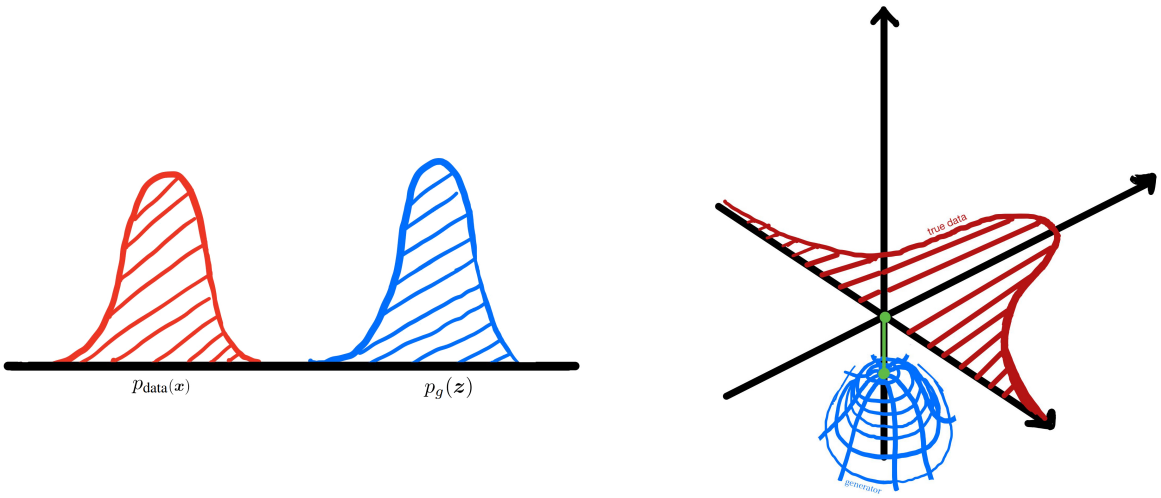
In Goodfellow et Al.'s publication [1], the authors briefly gloss over the issues that this base framework may pose, but without delving further into details. However, as it has been later demonstrated in many other sources [2], **GANs can be improved** to produce better and/or more stable results.

As mentioned in the introduction to this report, the original inspiration that motivated us to study this field was actually a later publication by Arjovsky, Chintala and Bottou [3] on a variation called Wasserstein GAN; that we will introduce in due time.

However, before that we consider highly relevant to get a real understanding of the **main flaws of GANs**, which will provide pivotal insights into why do we need to improve it, and more importantly on how to perform this improvement. Coincidentally, two of the aforementioned authors (Arjovsky and Bottou) have a previous publication also from 2017 [4] where they expose precisely these points, and on which they heavily rely to construct the WGANs, which we present now.

4.1 Perfect discrimination · Cause

When the supports of the distributions are disjoint or lie on low dimensional manifolds, there will always be a perfect discriminator between the real data distribution and generator distribution. This statement is described and proved in the Theorem 2.1 of [4].



(a) Disjoint supports (b) Low dimensional manifolds

Figure 2: Distance between densities

Despite having a reference for the mathematical understanding of these phenomena, we also consider that having some illustrative examples[4] will elucidate the reader on the problem at hand:

- By **disjoint supports** we understand what we see in Figure (2a): in practise the probability densities of the real data and of the generator are such that **the set of all points of their domains where the densities do not evaluate to 0 are mutually disjoint**. In simpler terms, the densities do not overlap at all. Therefore, it is way easier for a good classifier to find a decision rule to discern when the data is *'real'* or *'fake'*.
- The **low dimensional manifold** problem is more complex to describe. Take a look at Figure (2b): In this hypothetical case, p_g and p_{data} do not overlap either. However, considering only the dimensions where p_{data} has a non-zero measure (in red), and assuming for a moment that we are *'blind'* to the third dimension where p_g is also parametrized, we should in fact see an overlap. However, this is clearly not the case. The fact that both of them are defined using **different reference frames** makes it so that, by the normal evaluation metrics (KL and JS divergence), we automatically get the outcomes corresponding to the maximal divergence between the two densities.

However, coming back to Theorem 2.1 in [4]... what do we mean by a **perfect discriminator**? We label D as such when it can correctly separate the observations coming from p_g from those coming from p_{data}. From the proof of our **Theorem 3.1** we know that, for an optimal discriminator $D^*(\boldsymbol{x})$, the generator's objective function is:

$$V(D^*, G) = 2 \cdot JSD(\mathbb{P}_{data} \| \mathbb{P}_g) - 2\log(2) \tag{9}$$

Meaning that we have to optimize over the JS Divergence. Suppose $D^*(\boldsymbol{x})$ is the perfect discriminator, this means that it will always output: $D^*(\boldsymbol{x}) = 1$ when $x \sim p_{data}(\boldsymbol{x})$ and $D^*(\boldsymbol{x}) = 0$ when $x \sim p_g(\boldsymbol{x})$.

When this happens, the loss function $V(D^*, G)$ will go to zero and **there will not be any gradient to update the $C(G)$ during the training of the generator**, which we will detail in a moment.

Additionally, in [4] it is shown that (in practise) when we train D given G until it converges, **the discriminator error will go to zero**. Consequently the JSD between the distribution will be maximized. We know by the definition of the JSD that for log base e, or ln, **the upper bound is ln(2)**: $0 \leq JSD(\mathbb{P}_{data} \| \mathbb{P}_g) \leq \log(2)$. Replacing in equation (9) we will get $V(D^*, G) = 0$.

[4] And, as a GSEM professor likes to say in his lessons, *'...to have a visual break from all the equations...'*

4.2 Perfect discrimination · Vanishing gradient

We know that in practise we can not work with the true discriminator because it is not known. In this case we will use an approximation to the optimal discriminator. Using our definition of the value function in Equation (4) and assuming a parametric shape for the generator G as a function of θ, we note that:

$$\begin{aligned}\nabla_\theta V(D,G) &= \nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G_\theta(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\nabla_\theta \log(1-D(G_\theta(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\frac{\partial D(G_\theta(\boldsymbol{z}))}{\partial G_\theta(\boldsymbol{z})}\frac{\partial G_\theta}{\partial \theta}\frac{1}{D(G_\theta)-1}\right]\end{aligned}$$

For simplicity, we will denote: $\boldsymbol{x} = G(\boldsymbol{z})$ so that we have a mapping from $p_{\boldsymbol{z}}(\boldsymbol{z})$ to $p_g(\boldsymbol{x})$. We end up obtaining:

$$\nabla_\theta V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\left[\frac{1}{D(\boldsymbol{x})-1}\nabla_{\boldsymbol{x}} D(\boldsymbol{x})\nabla_\theta \boldsymbol{x}\right].$$

This describes the **backpropagation process** when the training on the discriminator is already done. If we already trained D to the point of *'perfection'*, what will happen for $D^*(\boldsymbol{x})$ and $\boldsymbol{x}\sim p_g(\boldsymbol{x})$ is the following:

$$\lim_{D\to D^*} D(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x}\sim p_g(\boldsymbol{x}) \Longrightarrow \lim_{D\to D^*} \nabla_{\boldsymbol{x}} D(\boldsymbol{x}) = 0$$

Finally, we conclude that:

$$\lim_{D\to D^*} \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\left[\frac{1}{D(\boldsymbol{x})-1}\nabla_{\boldsymbol{x}} D(\boldsymbol{x})\nabla_\theta \boldsymbol{x}\right] = 0$$

In [1] it was mentioned that in practise, optimizing Equation (4) for G will not provide a sufficient gradient for improvement, as we have shown above. It can so happen that in the very first learning, the generator is poor (it produces low-quality samples), while the discriminator is always strong enough to distinguish between real and fake sample.

Goodfellow proposed a variation of the value function to solve this, which was thought to make G perform better. However, it does not work: in [4] Arjovsky and Bottou show that:

$$\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[-\nabla_\theta \log D^*\left(g_\theta(\boldsymbol{z})\right)|_{\theta=\theta_0}\right] = \nabla_\theta\left[KL\left(\mathbb{P}_{g_\theta}\|\mathbb{P}_{data}\right) - 2JSD\left(\mathbb{P}_{g_\theta}\|\mathbb{P}_{data}\right)\right]|_{\theta=\theta_0}.$$

We draw attention to the fact that the KLD in this expression is not equivalent to the maximum likelihood equivalence from our **Theorem 1.1**. Moving on, Theorem 2.6 from [4] shows that the new gradient for the alternative cost function of the generator is very unstable, as the coordinates of $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[-\nabla_\theta \log D\left(g_\theta(\boldsymbol{z})\right)\right]$ follow a Cauchy distribution (with infinite expectation and variance).

The point here is that **the performance of GANs is very sensible to the cost function** assigned to G. In practise, when it is not correctly specified, either the gradient will vanish or it will be very unstable.

Arjovsky and Bottou then come with an alternative *ad-hoc* fix to the instability and the vanishing gradient issues. This is done by **adding continuous noise** to the inputs of the discriminator: knowing that the supports of the two distributions p_{data} and p_g are close to each other on low dimensional manifolds, this should increment the chances that p_{data} and p_g will match in a segment of the space. The JSD will then not be maximal between the distributions. However, this extra stability comes at a cost: adding this noise obviously worsens the final quality of the output. Luckily, we have better alternatives for this.

5 Wasserstein Generative Adversial Networks

We have seen already that GANs have some issues that make the training very difficult to implement. In this part we will direct our attention on another way to measure how close the model distribution and real data distribution are. In particular, we explore the definition of [3] on a new distance or divergence between the distributions p_g and p_{data}: the Wasserstein-1 or Earth Mover distance.

We will see that this new distance has better properties compared to the Jensen–Shannon and Kullback-Leibler divergence measures.

$$W\left(\mathbb{P}_{data}, \mathbb{P}_g\right) = \inf_{\gamma \in \Pi\left(\mathbb{P}_{data}, \mathbb{P}_g\right)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{10}$$

This distance is defined as in Equation (10). To elaborate, $\Pi\left(\mathbb{P}_{data}, \mathbb{P}_g\right)$ is the set of all possible joint probability distributions γ whose marginal distriubutions are respectively $\mathbb{P}_{data}$ and $\mathbb{P}_g$. As an intuition, γ can be perceived as the optimal *'transport plan'* to allocate mass from x to y in order to transform the $\mathbb{P}_{data}$ into $\mathbb{P}_g$.

Wasserstein distance compared

To better understand the implications of the W distance, we elaborate on the example discussed in [3], although we provide a more general proof in *Appendix A1*. Let $Z \sim U[0,1]$ and $\mathbb{P}_0$ be the distribution of $(0, Z) \in \mathbb{R}^2$. Now let $G_\theta(z) = (\theta, z)$ where θ is a single real parameter. How do the Kullbach-Leibler, Jensen-Sheannon and Wasserstein-1 distances perform in this simple setting? For simplicity, we note:

- $\mathbb{P}_0$ is a distribution where $\forall(x,z) \in \mathbb{P}_0$, we have $x = 0$ and $z \sim U[0,1]$.
- $\mathbb{P}_\theta$ is a distribution where $\forall(x,z) \in \mathbb{P}_\theta$, we have $x = \theta, 0 \leq \theta \leq 1$ and $z \sim U[0,1]$.

Kullback-Leibler divergence

$$KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \int p_0(x) \log\left(\frac{p_0(x)}{p_\theta(x)}\right)$$
$$KL(\mathbb{P}_\theta \| \mathbb{P}_0) = \int p_\theta(x) \log\left(\frac{p_\theta(x)}{p_0(x)}\right)$$

We define:

$$p_0(x) = P_0(X = x) \text{ and } p_\theta(x) = P_\theta(X = x)$$

When $\theta \neq 0$ and $x = 0$, we have the following results:

$$p_0(0) = P_0(X = 0) = 1 \text{ and } p_\theta(0) = P_\theta(X = 0) = 0$$

$$KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \int p_0(0) \log\left(\frac{p_0(0)}{p_\theta(0)}\right) = \int 1 \cdot log\left(\frac{1}{0}\right) dx = +\infty$$

Conversely, for$\theta \neq 0$ and $x = \theta$:

$$p_0(\theta) = P_0(X = \theta) = 0 \text{ and } p_\theta(\theta) = P_{theta}(X = \theta) =$$

$$KL(\mathbb{P}_\theta \| \mathbb{P}_0) = \int 1 \cdot \log\left(\frac{1}{0}\right) = +\infty$$

Only in the case where $\theta = 0$ the two distributions will overlap:

$$KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = KL(\mathbb{P}_\theta \| \mathbb{P}_0) = 0$$

Jensen-Shannon divergence

$$JSD\left(\mathbb{P}_0 \| \mathbb{P}_\theta\right) = \frac{1}{2}\int_x p_0(x)\log\left(\frac{p_0(x)}{\frac{p_\theta(x)+p_0(x)}{2}}\right)dx + \frac{1}{2}\int_x p_\theta(x)\log\left(\frac{p_\theta(x)}{\frac{p_0(x)+p_\theta(x)}{2}}\right)dx$$

When $\theta \neq 0$ we have the fallowing result:

$$\begin{aligned} JSD\left(\mathbb{P}_0 \| \mathbb{P}_\theta\right) &= \frac{1}{2}\int_x p_0(0)\log\left(\frac{p_0(0)}{\frac{p_\theta(0)+p_0(0)}{2}}\right)dx + \frac{1}{2}\int_x p_\theta(\theta)\log\left(\frac{p_\theta(\theta)}{\frac{p_0(\theta)+p_\theta(\theta)}{2}}\right)dx \\ &= \frac{1}{2}\int_x 1\cdot\log(2)dx + \frac{1}{2}\int_x 1\cdot\log(2)dx \\ &= \log(2) \end{aligned}$$

We know that JSD is symmetrical and :

$$JSD\left(\mathbb{P}_0 \| \mathbb{P}_\theta\right) = JSD\left(\mathbb{P}_\theta \| \mathbb{P}_0\right) = \log(2)$$

Wasserstein-1 Distance

When $\theta \neq 0$:

$$W(\mathbb{P}_0 \| \mathbb{P}_\theta) = |0 \cdot p_0(0) - \theta \cdot p_\theta(\theta)| = |\theta|$$

When $\theta = 0$:

$$W(\mathbb{P}_0 \| \mathbb{P}_\theta) = 0$$

We can conclude from this example that **only the Wasserstein-1 distance can guarantee continuity** almost everywhere, and it is a smooth measure very helpful for stabilizing the gradient descent. As the authors of [3] mention in the paper, we need to find a loss function that is continuous to define the divergence between distributions, which this achieves more nicely: Even if the distributions are far away, the distance in Equation (10) will not blow up i.e it will always have a stable value, solving the main problem that the KL and JS divergences present, as it is experimentally proven in [3].

5.1 WGAN explained

The main change of WGANs with respect to GANs is that the generator's loss function becomes:

$$\min_G W(\mathbb{P}_{data} \| \mathbb{P}_g)$$

The problem is that the original definition of the EM distance is intractable, because of the $infinimum$ in Equation (10). To be able to do the minimization, the authors of WGAN introduce the **Kantorovisch-Rubinstein duality principle**, in order to obtain an approximation of the Wasserstein distance that is tractable.

$$W\left(\mathbb{P}_{data}, \mathbb{P}_g\right) = \sup_{\|f\|_L \leq K} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_g}[f(\boldsymbol{x})]$$

In the new form of EM-distance, the *supremum* is the maximum over all 1-Lipschitz functions. It is specified that changing the parametrization of the function f into a family of functions $(f_w)_{w\in\mathcal{W}}$ that are K-Lipschitz continuous, where $\mathcal{W}$ is a compact set of all possible weights w the new problem to solves will be:

$$\max_{w\in\mathcal{W}} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}}\left[f_w(\boldsymbol{x})\right] - \mathbb{E}_{z\sim p(\boldsymbol{z})}\left[f_w\left(g_\theta(\boldsymbol{z})\right)\right] \tag{11}$$

This function can be used as the loss for the same Neural Networks than before, with some adaptations. Analogous to the GAN framework, we fix the generator and train the f_w to an optimal solution. The author is calling the neural network f, a **critic** and not a discriminator with the scope to better clarify that here **the critic does not function as a classifier**, as it will help to estimate based on Equation (10) the realness of the input. After we obtain the optimal critic we can compute the gradient of the generator:

$$\nabla_G W(P_{data}, P_g) = -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[f_w(g_\theta(\boldsymbol{z}))\right]$$

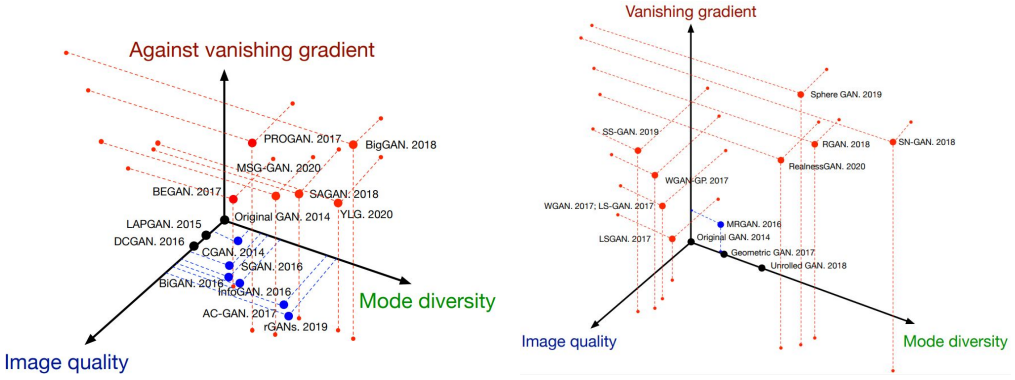
To ensure the K-Lipschitz continuity condition, the authors of [3] come with a practical trick, which consists on constraining the maximal influence of every weight in the Neural Networks after each step of backpropagation of the critic. They label this process as *'weight clipping'*, and it ensures that the gradient is bounded at each upgrade so that it cannot suddenly diverge explosively.

Also if we compare with the value function of the base GAN in Equation (4), this new formulation does not make use of the log. We have already seen that the GAN can be understood as a min-max adversarial game, where we need to alternate gradient updates between the discriminator and the generator, while being careful not to have a perfect discriminator that can cause the vanishing gradient problem. In the case of the WGAN, we can train the critic as optimally as we want before updating the generator, as the nice properties of the Wasserstein-1 distance ensure that even then there exists a gradient for the generator to work upon.

6 The GAN ecosystem

Generative Adversarial Networks have heralded a revolution within unsupervised learning: they are useful, cutting-edge, and perhaps more important, very cool. For reasons that we hope are clear at this point, they are tremendously efficient in learning from the data. For other reasons also apparent though, the base GAN is far from perfect and presents ample room for improvement, of which WGANs are just one (but remarkable) example.

Trying to even cover the variations proposed over all possible tasks would probably require us to escalate the length of this report by orders of magnitude. However, what we can do is to expose some of the most promising alternatives and briefly discuss what makes them so compelling. As mentioned in the introduction, *computer vision* is the discipline where the GAN framework has caused the deepest impact [2], so we will narrow down our perspective to this field.



(a) Architecture variant GANs

(b) Loss variant GANs

Figure 3: From [2], GAN developments within the field of *computer vision*.

Even when focusing on this environment, we see several important models pop up. In general, we can distinguish between those that present **architecture improvements** and those that are fitted through **loss function modifications**. Figure (3) summarizes both sub-families of GANs and assigns them based on how well they solve the vanishing gradient and mode dropping problems mentioned in *Section 4*, and then on how good of an image they output.

6.1 Architecture-variant GANs

In brief, they focus on introducing **new elements within the neural network structure** of the GAN, and are usually designed with the aim of fulfilling a particular task (such as image-to-image transfer or image completion, for example). Given the heuristic nature of modelling the architecture of Neural Networks, these models tend to benefit from improvements introduced in previous iterations. Two cases within this group caught our attention, which we now explore.

6.1.1 Self-Attention GANs

Also known as SAGAN, it was first proposed in [5], and modifies the base GAN architecture by introducing two important innovations:

- **Self-attention maps**: Where the image features $\boldsymbol{x} \in \mathbb{R}^{C^{(i)} \times N^{(i)}}$ between hidden layers (i) and $(i+1)$ undergo an additional step, the **attention layer**, for which they are mapped to three feature spaces $f(\boldsymbol{x}) = W_f \boldsymbol{x}$, $g(\boldsymbol{x}) = W_g \boldsymbol{x}$ and $h(\boldsymbol{x}) = W_h \boldsymbol{x}$. Subsequently, f and g undergo matrix multiplication to obtain an attention map, labelled $\boldsymbol{S}$.

  This map is then passed through

  $$\beta_{i,j} = \frac{exp\{\boldsymbol{S}_{i,j}\}}{\sum_{i=1}^{N} exp\{\boldsymbol{S}_{i,j}\}},$$

  which intuitively corresponds to the level of dependence between two different regions, i and j. Finally, to obtain the self-attention mechanism $\boldsymbol{o} \in \mathbb{R}^{C^{(i)} \times N^{(i)}}$, we do

  $$\boldsymbol{o_j} = \boldsymbol{v}\left(\sum_{i=1}^{N} \beta_{i,j} \boldsymbol{h}\left(\boldsymbol{x_i}\right)\right); \quad \boldsymbol{v}\left(\boldsymbol{x_i}\right) = \boldsymbol{W_v x_i}.$$

  The output of this self-attention layer will correspond to $\boldsymbol{y}_j = \gamma \boldsymbol{o}_j + \boldsymbol{x}_j$, where γ is a scalar that is originally set to 0, to allow the neural network to first learn from its local inputs and then automatically adjust through the usual backpropagation mechanisms the introduction of $\boldsymbol{o}_j$.

  In more plain terms, what this achieves is to incorporate non-local feature information within the convolutional layer structure, which is of course more specialized on capturing local features due to the mechanisms through which convolutions work. This extra layer between convolutions is applied to both G and D.

- **Spectral normalization**: By implementing it instead of the normal batch normalization, the discriminator does not require of hyperparameter tuning while in the generator it can help in preventing over weighting some indicators, which stabilizes the gradient and facilitates convergence.

  $$\sigma(W^{(i)}) := \max_{\boldsymbol{h}:\boldsymbol{h}\neq\boldsymbol{0}} \frac{\|W^{(i)}\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = \max_{\|\boldsymbol{h}\|_2 \leq 1} \|W^{(i)}\boldsymbol{h}\|_2 \tag{12}$$

  For a layer $g(\boldsymbol{h}) = W^{(i)}\boldsymbol{h}$, spectral normalization is defined as in Equation (12) in [6], where $W^{(i)}$ is the matrix of weights of layer i and $\boldsymbol{h}$, in our *computer vision* premise, would correspond to the output of applying a convolution to the previous layer, which we naturally then normalize, using this spectral method. Note that in application, the normalization of the weights would be done by

  $$\bar{W}_{\mathrm{SN}}^{(i)}\left(W^{(i)}\right) := \frac{W^{(i)}}{\sigma(W^{(i)})}$$

  And in practise $\sigma(W)$ corresponds to the **largest singular value** of W.

In the end, with the use of those two additional inclusions within the GAN architecture, the outcome is stable and relatively high-quality images with increased mode variety as a consequence of incorporating information on non-adjacent features, suggesting that **self-attention and convolutional layers are complementary**, especially for large-scale complex images.

6.1.2 BigGANs

This model inherits from the innovations introduced in the SAGAN publication and introduces some of their own, discussed in [7], namely:

- Training the generator over a multivariate standard normal prior $\boldsymbol{z} \sim \mathcal{N}(0, I)$, but then when deploying it only use a **truncated** version, where priors falling outside of a given bound are resampled until they satisfy the constraint. This constraint is an extra hyperparameter that allos to control some trade-off between image quality and mode diversity.
- **Scaling the architecture** of GANs to sizes never before explored. By the sheer fact of increasing batch size, incrementing the number of channels in each layer by 50% and adding some extra features that help in improving regularization, the variety and quality of the outcome sees substantial improvements

As a result, BigGANs achieve **state-of-the-art performance** in high-resolution imaging, reason for which we wanted to expose both the model itself and SAGAN as its predecessor in this part of the report.

6.2 Loss-variant GANs

The focal problems of GANs, those of vanishing gradients and mode dropping, **cannot be solved by adjusting the architecture** of the models, as the same underlying issues with the Kullback-Leibler divergence will prevail. Therefore, modifications on the optimization problem and procedure have to be introduced that help in circumventing these issues, which is the main focus within this branch.

In contrast to architecture-variant propositions, loss-variant GANs (such as the WGAN) do not tend to work so much over previous improvements (except some minor cases or adaptations to particular geometries), for the simple reason that (save for some exceptions) they already optimize the loss function variant that they are proposing. We briefly cover two examples:

6.2.1 WGANs with Gradient Penalty

The authors in [8] argue that even though Wasserstein GANs have indeed better theoretical properties than regular GANs, the method of **weight clipping handicaps the performance** of the model, as it leads to optimization issues and worse quality than possible if the model converges.

In particular, by weight clipping to small values as it is suggested in [3], the GAN is limited to learning simple functions, while if the clipping is too generous it will not be able to attack the vanishing gradient problem. As it will be shown later on in this report through the `SOCOFing` example (see *Section 8*), this leads to the WGAN failing to converge if the data has local but subtle patterns.

As an alternative to weight clipping, which requires of fine-tunning to work as intended, the authors propose to **add a penalty term to the loss function of the critic**. Let $\hat{\boldsymbol{x}} \sim p_{\hat{\boldsymbol{x}}}$ be a second input prior for the critic. Then, define the loss of the critic as

$$\begin{aligned}\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \\ + \lambda \cdot \mathbb{E}_{\hat{\boldsymbol{x}}\sim p_{\text{data}}(\hat{\boldsymbol{x}})}\left[\log\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)\right],\end{aligned}$$

Where the last term corresponds to the incorporated loss[5]. The prior $\hat{\boldsymbol{x}} \sim p_{\hat{\boldsymbol{x}}}$ is sampled uniformly from any coordinate coming from the interpolation between random pairs of $(x_i \sim p_{data(x_i)}, y_i \sim p_{g(y_i)})$, as an approximate to make the unit gradient norm rule work more or less as intended.

The outcome of this is a WGAN that when optimized through stochastic gradient descent behaves better and **converges more nicely** than the WGAN with weight clipping, which increases the quality and mode variety of the output.

[5] With respect to the one shown in the paper, we have adapted this term to a shape resembling more that of Equation (4). The optimality conditions and outcome are exactly the same, however.

6.2.2 Realness GANs

One of the latest propositions [9], they consist on replacing the outcome of D by a distribution, $p_{realness}$, so that $D(\boldsymbol{x}) = \{p_{realness}(\boldsymbol{x}, u);\ u \in \Omega\})$ with Ω the outcome space, which can be arbitrarily adjusted, where every u would correspond to some interpretation of *'realness'* (similarly to the WGAN critic). Thus, instead of modelling whether we have a ground truth of 0 or 1 in the discriminator, we optimize based on the *'ground truth distributions'* $\mathcal{A}_0$ and $\mathcal{A}_1$, corresponding to the fake's and the real's distributions. The value function then becomes:

$$\max_G \min_D V(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \left[KL\left(\mathcal{A}_1 \| D(\boldsymbol{x})\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g} \left[KL\left(\mathcal{A}_0 \| D(\boldsymbol{x})\right)\right] \tag{13}$$

Leading to the optimal discriminator

$$D^\star_G(\boldsymbol{x}, u) = \frac{\mathcal{A}_1(u) p_{\text{data}}\left(\boldsymbol{x}\right) + \mathcal{A}_0(u) p_g(\boldsymbol{x})}{p_{\text{data}}\left(\boldsymbol{x}\right) + p_g(\boldsymbol{x})}$$

Which then also maximizes the value in Equation (13) when $p_g = p_{data}$. Experimentally, this has led to the creation of a very balanced model that achieves good results with respect to all the axes of Figure (3b).

7 Applications

GANs research in areas other than *computer vision* still has some way to go, given the domain specific requirements that they need to overcome. For example, and as an illustration of this, in natural language processing the underlying distribution p_{data} has a more **discretized nature** than in the case of image generation (where we can model each pixel as approximately continuous). Therefore, many of the most impressive innovations are related to imaging, although we will see other success stories.

Originally, we started exploring the implementations discussed in [2], however we consider that instead of summarizing information that is already available in a publication, it is more exciting to *'go into the wild'* and see what interesting applications have people come up with. This will also add a more light-hearted section to what has shaped up to be a technical report.

7.1 DNA sequencing

In [10], we see that the authors first use a GAN to generate DNA sequences contained within the encoding of certain proteins. They do so by applying a base GAN model where furthermore they implement a so-called **feedback loop** where after some training epochs, the most promising outcomes of the GAN are re-labelled as real and used to increase sample size (which for real proteins we assume that is both hard and expensive to obtain).

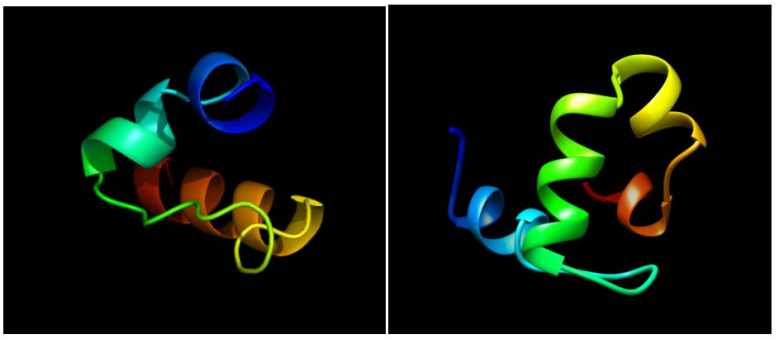


Figure 4: From [10], two examples of GAN-designed proteins

We can see the results of this implementation in Figure (4), where after several epochs the GAN starts to synthesize protein structures that could as well be observed in nature as can clearly be seen.

7.2 3D object modelling

The authors of [11] propose two interesting variations of GANs, which output 3D objects but with different structure and input allowance:

- **3D-GAN**: Which trains over 3-dimensional spatial data. This model is, in broad terms, simply a generalization of the 2D image processing GANs that are more common in literature: the model learns to map inputs $\boldsymbol{z}$ from the latent space to the desired shape, and then uses the usual discriminator to classify reals and fakes. The outcome allows to **obtain structures** like furniture, clothing, accessories, etc. even though the computational complexity of adding a third dimension escalates the training time dramatically.
- **3D-VAE-GAN**: The existence of a latent space gives room for us to, once the GAN is trained, choose an arbitrary coordinate set within it and obtain a 3D representation of an object. If the training has been stable, and the coordinate choice sensible, then this object will look like a realistic variation to that of one of the classes encoded in the latent space.

Figure 5: From [11], some 3D projections on the content of 2D images

Furthermore, if there exists a sensible way to project a 2D image of an object into its pertinent coordinates within the latent space, then it is possible to obtain what the trained GAN would allocate to be its 3D representation, which is exactly what this variation does, as in Figure (5).

In brief, this is done by adding a **variational autoencoder** that will project the data $\boldsymbol{x}$ into the latent space $\boldsymbol{z}$. Then, the generator G acts as a decoder that maps these images to 3D.

As can be seen, these approximations are still rough, but already suggest some interesting ideas. Being creative, we could think of applying these models (or future improvements) through an **image to 3D printing** process, for example.

7.3 Font generation

We found a publication [12] that proposes a model, **Attribute2font**, which allows to create interpolations between different font types of text.

Figure 6: From [12], fonts as a function of attributes

Figure (6) shows what this model can achieve, namely the automatic design of fonts as a function of certain attributes, which are arbitrarily chosen. Following the philosophy of encoders, this is done through exploring different coordinates within a latent space. In this case these coordinates are defined by attributes such as *gentle*, *strong*, or *legible*, which are labels within the original data but only used as complementary information, and not for direct prediction (and hence the **semi-supervised learning** nature of this work).

7.4 Art

As already discussed, GANs excel in producing quality results for image related tasks. These results, as we hope to have illustrated by this point, can extend to a myriad of fields and produce useful contributions to society.

However, they can also be used to enhance the **creativity** present in our human species, an area often overlooked when thinking about high-complexity models.

Figure 7: GANs learning from classical art

The artist Mario Klingemann is specialised in producing AI-related art by training Machine Learning algorithms with diverse creative goals in mind. One of his projects, *'Memories of passerby I'*, consisted on training some sort of GAN variant over a data set of classical art, containing portraits from the 17^{th} up to the 19^{th} century.

Some sort of stochastic algorithm is run over the latent space. As far as we understand, the random path that it navigates generates a **stream of transitioning images** that do never quite look the same. Figure (7) illustrates some publicly available examples, but in the site of the project many more can be found.

7.5 Fun

ETH Zürich investigator and occasional youtuber Yannic Kilcher shared very recently a video where he combines BigGAN, from *Section 6.1.2* with a supervised learning algorithm, **CLIP** [13], to produce a music video that is *'synchronized'* with a song he composed.

The role of BigGAN is to train on a set of images and define an image output based on selected coordinates from a latent space, as usual. Then, CLIP is used sort of as a loss metric: for a given piece of the lyrics of the song, it will assess the "goodness of fit" of the associated image that BigGAN generates for it. As Yannic Kilcher mentions, the good thing about this process is that it is fully differentiable, meaning that it is possible to **backpropagate from CLIP to the GAN** until the generated image fits the input piece of lyrics well, which is what he does in his video. Then, as usual, he interpolates the images for each consecutive piece of lyrics to create the transitions that can be seen.

The potential applicability of this particular project may be less ambitious than others discussed before, however it serves its purpose very well: to show to the world, in a simpler manner, what can possibly be achieved by GANs. We personally found it to be a very interesting watch.

8 Implementation

As a final contribution, we trained our own model over three sets of images: The handwritten numbers of `MNIST`, the clothing images of `fashion_MNIST` and a set of 6.000 fingerprint images from 600 African subjects publicly available on Kaggle as the SOCOFing dataset. As mentioned, the publication that originally inspired this report was the paper on WGANs [3], so we have decided to go for its application.

Along with this report, there will be attached two `.ipynb` files:

- `image_processing.ipynb` will detail how we learned about the structure of image data, and how to process a multi-channel $m \times n$ image into a shape that will then be used as an input to train the WGAN.
- `WGAN.ipynb` contains the code that we employed to train the WGAN for one of the datasets that we implemented, the `SOCOFing`, in particular. The code is for all three applications quite similar, only adjusting the data entry pipelines and scaling the architecture of the WGAN.

Also, to give credit where credit is due, we have based our code of `WGAN.ipynb` on the same philosophy followed in Jason Brownlee's book: *'Generative Adversarial Networks with Python'* [14]. This code has of course been adapted to our own necessities and cases, which we have accompanied with commentary between the kernels to further clarify what every building block is contributing to the execution. Thus, here we **focus on inspecting the results**, while we leave the code, as well as our comments on that regard, to the Jupyter notebooks.

What follows is the qualitative analysis for each of the data sets. Afterwards, we close with some remarks based on our insights from the different applications. We have included some figures related to the image generation process of each data set, however for spacing purposes they have been mostly relegated to the *Appendix A2-A6*[6].

8.1 Hand-written digits

More of a toy model, it is the case where the WGAN more easily converges.

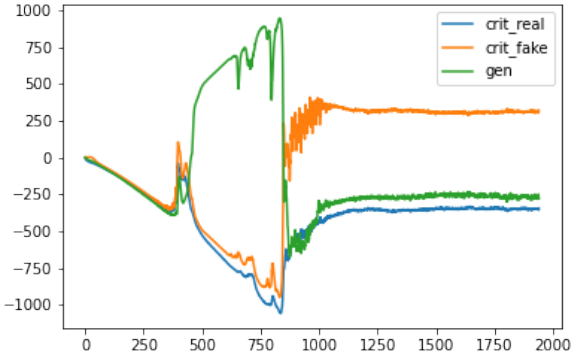


Figure 8: WGAN optimization for `MNIST`, loss functions.

Figure (8) and *Appendix A2* show the evolution of the critic (only in the plot) for fake and real data and of the generator as the WGAN learns in subsequent training iterations. There is a more or less recurrent pattern, not only for this example but for all the others that we illustrate:

- First, the WGAN appears to learn about the overall structure of the corresponding class, in this case how do *'standard'* handwritten eights look like, which we can see in Figure (8) as a downwards line at the left of the plot. There appears to be some sort of **initial mode dropping** related to this, as the epoch 2 plot highlights (most of the eights are quite similar).
- Next, it tries to introduce variation so that, in practise, we see different types of hand-written eights when performing visual inspection. However, as it of course has no intuitive knowledge of what the concept of *handwritten eight* is, it does so at random, to the point where nothing is distinguishable and there is a

[6]The full set of images is available on request, or alternatively it can be replicated by adjusting the WGAN architecture and data entry pipeline in the `.ipynb` notebook to each particular case.

high amount of noise for some time, as in the epoch 5 plot in *Appendix A2*. In Figure (8), this relates to the moment where the generator and critic losses diverge the most.

- Afterwards, the WGAN appears to slowly find the correct parameters and transformations that allow for the minimization of the Generator's loss. This translates to **visual improvements** that increasingly resemble the real images. The epoch 15 plot is illustrative of this, as the eights are still not perfect but look already convincing.
- Finally, **the model stabilizes**. It has learnt a sufficiently good set of parameters and does not improve anymore. The differences between the real and handwritten digits are minimal (epoch 30, the final one, is a good example of this).

Despite `MNIST` being a classical and overused case, we wanted to train a WGAN on it given that it nicely converges to the desired outcome, thus we can present it as a success case and a benchmark to compare against.

8.2 Low-resolution clothing pictures

We attempt the same architecture used in *Section 8.1* for the pictures in the `fasion_MNIST` dataset.

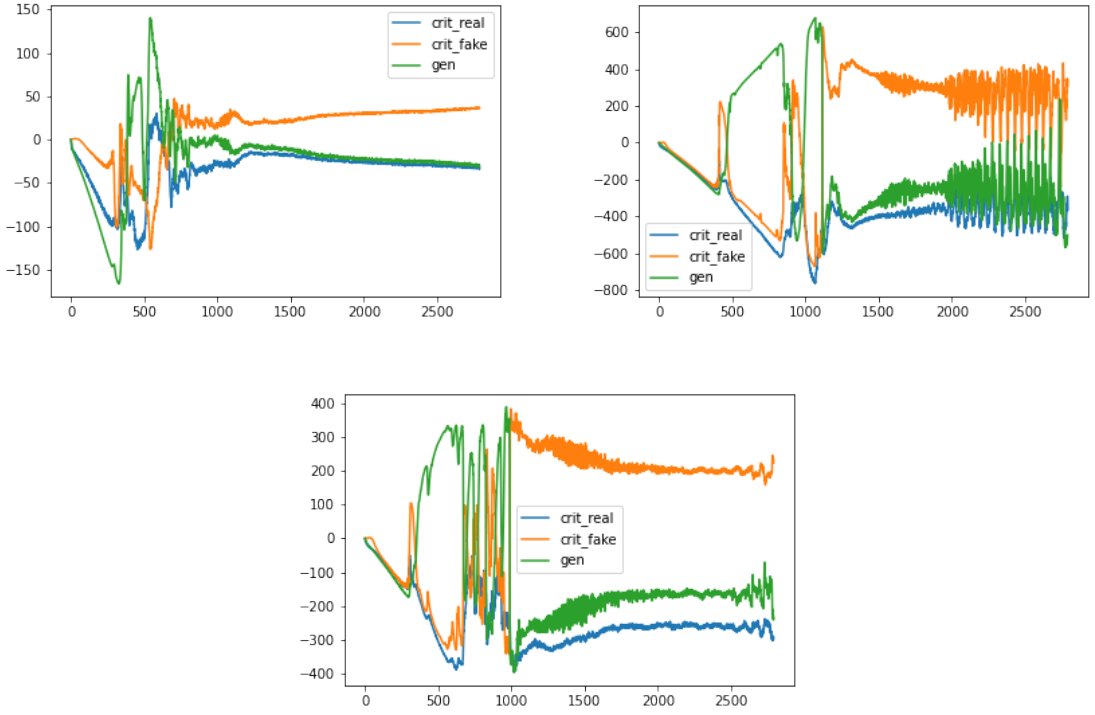


Figure 9: WGAN optimization for `fashion_MNIST`. Bags (top left), dresses (top right) and sandals (bottom).

However, as it can be observed in *Appendix A3-A5*, the results are remarkably less convincing than in the previous case. A **similar process** than that described before occurs: the WGAN first learns the overall structure but without any variety, then it introduces a high amount of noise and finally it slowly converges to something. This something, however, is not so nice as with the `MNIST` case. Some observations:

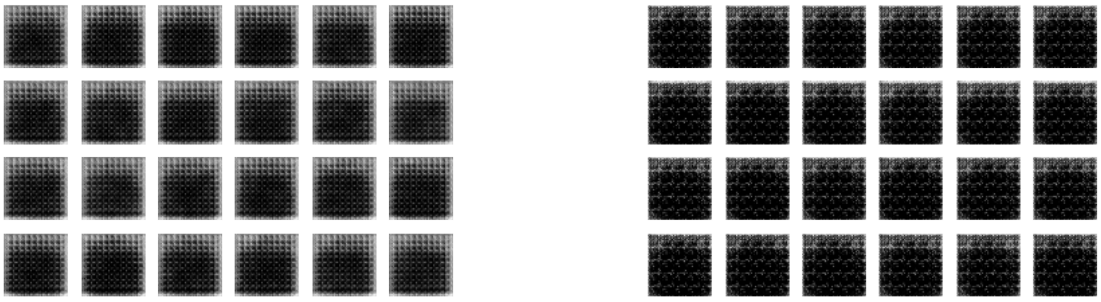
- As mentioned, the WGAN does not perform so well, in terms of the visual quality of the output, in this case. We suspect that this might be related to an extra layer of **complexity** with the `fashion_MNIST` data: for the hand-written digits, the WGAN only has to learn about the structure and its variations. However, in our current case, the pictures have an added **grayscale gradient** to them which makes the learning much more difficult. The result is then images reminiscent of the intended objects but that fall short from capturing the specific details.

- In the **bags** case, we see that the loss function evolution appears to suggest that the images get visually worse as time goes by. However, that is not the case in reality, where the latest epochs produce the highest quality output.
- For **dresses**, we observe a somewhat exotic behavior: after striking an acceptable optimization path, all the losses escalate their oscillations significantly during the latest epochs. We suspect that this might be related to the WGAN trying to find out more **extreme alternatives** that might lead to a better Wasserstein-1 evaluation. In the full set of dress images, we visually detect an oscillation in the quality of the images that would correspond to this. A similar process is suspected to start to occur at the end of the **sandals** process.

To summarize, in this second *'level of complexity'* for the data, we could still argue that there is some degree of success: the WGANs *'converge'* to some images, and those images could be roughly identifiable as their correspondent classes in most cases. This begs the question, however, of what happens when the complexity is escalated even further.

8.3 Fingerprint images

In this final case, we comment on the outcome of applying WGANs to the `SOCOFing` data. The architecture and specifications for this model are slightly different: the images contain (after processing) 80×80 pixels compared to 28×28 for the other data sets, thus the WGAN architecture has been adjusted accordingly.



(a) Fingerprint Generator, 11×11 variant epoch 4 (b) Fingerprint Generator, 5×5 variant epoch 4

Figure 10: Dense layer patterns

The presentation for this last case will be different, as we never finished any of the trainings (once it was clear that the results were not good we stopped the runs, as the computations were *very* time consuming) and thus we do not have available the loss function evolution plots from the previous sections. In brief:

- We tried several architectures respecting more or less the original image shape and with different layer specifications, but **none converged**. In *Appendix A6* it can be seen that after somewhat learning the rough shape of fingerprints, the images vanish into the dark never to come back. Roughly, this corresponds to the diverging part of Figures (8) and (9), though in this case the model is unable of reverting the process. This can be verified by running the `WGAN.ipynb` code, where the output of the `train(...)` function indicates the Wasserstein-1 loss at every iteration.
- We think that the available sample size and quality of the data, as well as the complexity and **non-linearities** present in human fingerprints, may be the main factor behind this outcome. A quick inspection to the real fingerprint images verifies that there may be too much diversity: they are not centered, in some cases the image is cut in the corners, etc. which no doubt imply additional complications.
- Interestingly, for all of the architectures that we tried, we see that **the dense layer has a huge influence**, at least on the intermediate outcomes. In Figure (10) we see the effect of an 11×11 initial dense layer (a) and of a 5×5 case (b). In both of them, the structure is clearly conditioned on the cells it generates.

We present this as a case where the WGAN fails to give any reasonable approximate of the class. As any other model, they are limited to their specifications and to the richness and availability of information, after all.

Conclusion

In this report we investigated what characterizes a GAN and how it works in practise. We analized its properties as well as its inherent issues, and we explored how WGANs solve them, at least in theory. Additionally, we took a wider view into the GAN family environment and focused onto some of the most promising applications, before discussing the outcomes of our own experiments over different data sets.

We conclude that GANs in general are tools with a tremendous potential over a vast variety of tasks. However, they are not perfect, neither in theory nor in practise, and indeed it is likely that some other class of models will overtake over them eventually, if not already[7]. This however cannot deny the enormous advances that they have brought over the field of unsupervised learning.

[7]For example, last week some investigators from OpenAI asserted in https://arxiv.org/abs/2105.05233 that Diffusion Models can beat the best GANs on `ImageNet`, at the expense of more computational power.

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.* arXiv, 2014.

[2] Zhengwei Wang, Qi She, and Tomas E. Ward. *Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy.* arXiv, 2020.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN.* arXiv, 2017.

[4] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks.* arXiv, 2017.

[5] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. *Self-Attention Generative Adversarial Networks.* arXiv, 2019.

[6] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. *Spectral Normalization for Generative Adversarial Networks.* arXiv, 2018.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis.* arXiv, 2019.

[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. *Improved Training of Wasserstein GANs.* arXiv, 2017.

[9] Yuanbo Xiangli, Yubin Deng, Bo Dai, Chen Change Loy, and Dahua Lin. *Real or Not Real, that is the Question.* arXiv, 2020.

[10] Anvita Gupta and James Zou. *Feedback GAN (FBGAN) for DNA: a Novel Feedback-Loop Architecture for Optimizing Protein Functions.* arXiv, 2018.

[11] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling.* arXiv, 2017.

[12] Yizhi Wang, Yue Gao, and Zhouhui Lian. *Attribute2Font: Creating Fonts You Want From Attributes.* arXiv, 2020.

[13] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. *Learning Transferable Visual Models From Natural Language Supervision.* arXiv, 2021.

[14] Jason Brownlee. *Generative Adversarial Networks with Python.* Self-Published, Melbourne, Australia, 2019.

Appendix

A1 · General problem for KL and JS divergences

Based on Theorem 2.3 from [4], we understand that when p_g and p_{data} suffer totally or partially from the problems illustrated in Figure (2), the divergences will be maximal. To see it, we apply the definition of the JSD:

$$\begin{aligned}
JSD\left(\mathbb{P}_{\mathrm{g}} \| \mathbb{P}_{\mathrm{data}}\right) &= \frac{1}{2}\int_x P_{data}(x)\log\frac{P_{data}(x)}{\frac{P_{data}(x)+P_g(x)}{2}}dx + \frac{1}{2}\int_x P_g(x)\log\frac{P_g(x)}{\frac{P_{data}(x)+P_g(x)}{2}}dx \\
&= \frac{1}{2}\int_x P_{data}(x)\left(\log(2) + \log\left(\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right)\right)dx \\
&+ \frac{1}{2}\int_x P_g(x)\left(\log(2) + \log\left(\frac{P_g(x)}{P_{data}(x)+P_g(x)}\right)\right)dx \\
&= \frac{1}{2}\int_x P_{data}(x)\log(2)dx + \frac{1}{2}\int_x P_g(x)\log(2)dx \\
&+ \frac{1}{2}\int_x \left(P_{data}(x)\log\left(\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right) + P_g(x)\log\left(\frac{P_g(x)}{P_{data}(x)+P_g(x)}\right)\right)dx
\end{aligned}$$

By the fact that we have $P_g(x) = 0$, the last integral will evaluate to 0. Therefore, we get that:

$$\begin{aligned}
JSD\left(\mathbb{P}_{\mathrm{g}} \| \mathbb{P}_{\mathrm{data}}\right) &= \frac{1}{2}\log(2)\int_x P_{data}(x)dx + \frac{1}{2}\log(2)\int_x P(x)dx \\
&= \log(2)
\end{aligned}$$

As JSD is symmetric, the same proof applies to $JSD(\mathbb{P}_{data}\|\mathbb{P}_g) = \log(2)$. For KL, we observe:

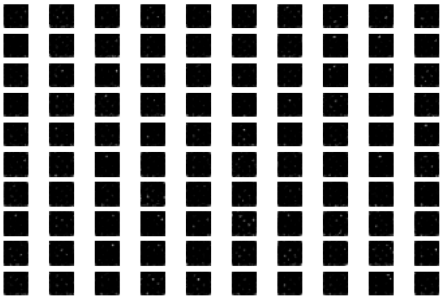
$$\begin{aligned}
KL(\mathbb{P}_{data}\|\mathbb{P}_g) &= \int_x P_{data}(x)\log\left(\frac{P_{data}(x)}{P_g(x)}\right)dx \\
&= +\infty,
\end{aligned}$$

Since $P_g(x) = 0$ so its $-\log$ evaluates to $+\infty$.

A2 · MNIST hand-written 8 images

(a) MNIST Generator, epoch 1

(b) MNIST Generator, epoch 2

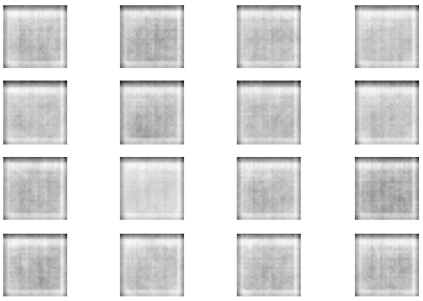


(a) MNIST Generator, epoch 5

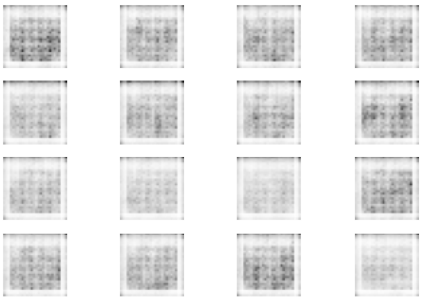
(b) MNIST Generator, epoch 15

Figure 13: MNIST Generator, epoch 30

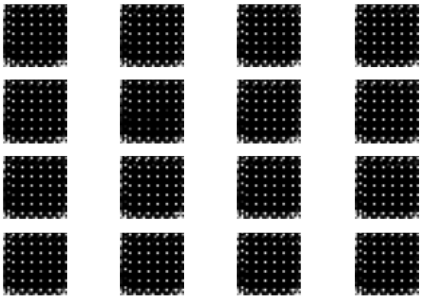
A3 · Low-resolution clothing: Bags



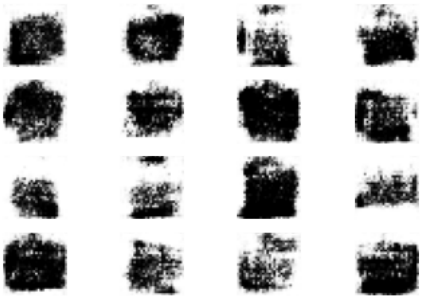
(a) Bags Generator, epoch 1



(b) Bags Generator, epoch 2



(a) Bags Generator, epoch 5



(b) Bags Generator, epoch 15

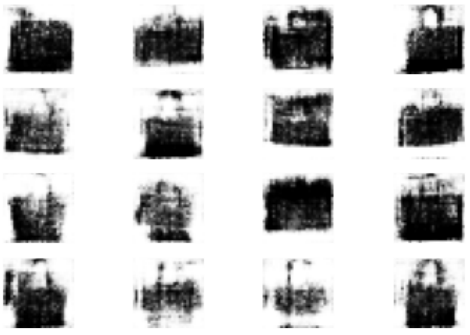
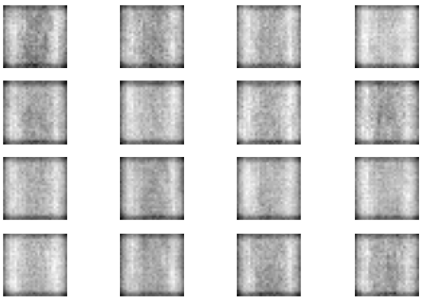
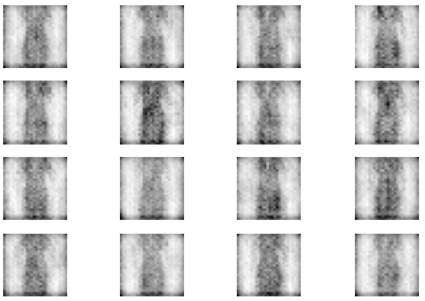


Figure 16: Bags Generator, epoch 30

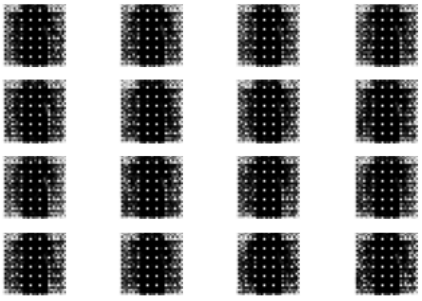
A4 · Low-resolution clothing: Dresses



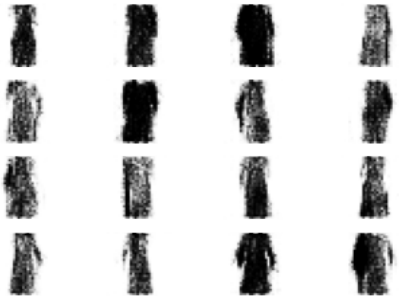
(a) Dresses Generator, epoch 1



(b) Dresses Generator, epoch 2



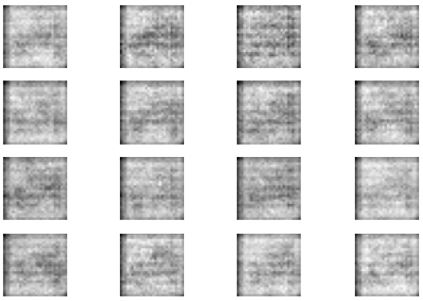
(a) Dresses Generator, epoch 5



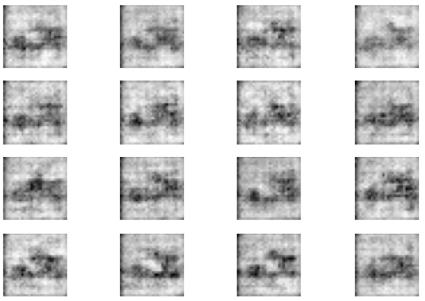
(b) Dresses Generator, epoch 15

Figure 19: Dresses Generator, epoch 30

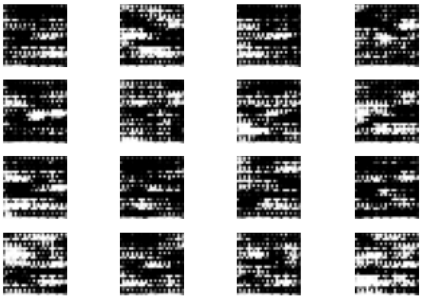
A5 · Low-resolution clothing: Sandals



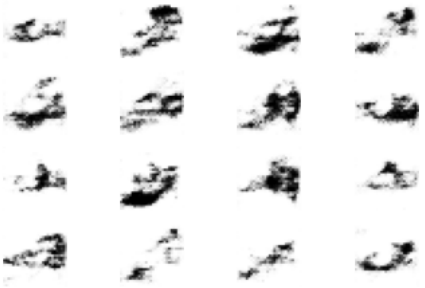
(a) Sandals Generator, epoch 1



(b) Sandals Generator, epoch 2



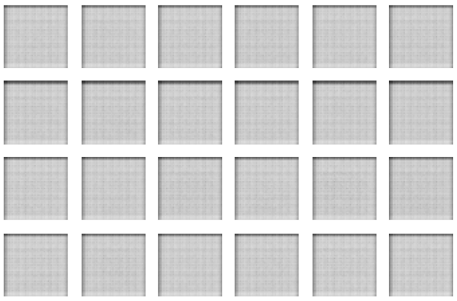
(a) Sandals Generator, epoch 5



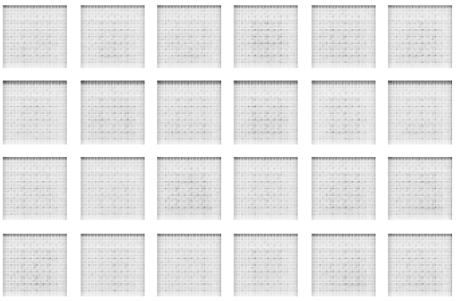
(b) Sandals Generator, epoch 15

Figure 22: Sandals Generator, epoch 30

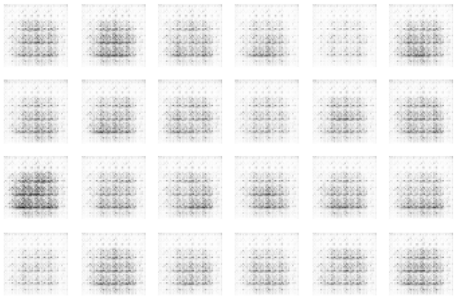
A6 · SOCOFing data



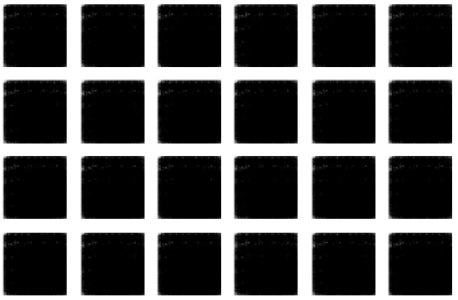
(a) Fingerprint Generator, epoch 1



(b) Fingerprint Generator, epoch 2



(a) Fingerprint Generator, epoch 3



(b) Fingerprint Generator, epoch 5

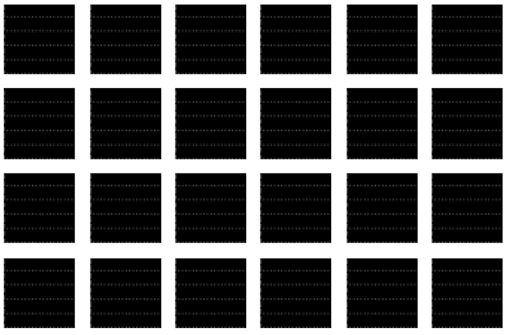


Figure 25: Fingerprint Generator, epoch 15