Master 2 in Mathematics and Applications (Sorbonne Université)

With specialization in Mathematics of modelization, Control, Optimization and Calculus of Variations

Master thesis

SiGANtures: Generating Times Series Using Wasserstein-Generative Adversarial Nets and the Signature Transform

Author: Linus BLEISTEIN

Supervisors: Gérard BIAU, Claire BOYER and Adeline FERMANIAN

Contents

List of Figures

Acknowledgments

I am deeply indebted to my supervisors Gérard Biau, Claire Boyer and Adeline Fermanian for accepting to oversee my work. Gérard provided excellent advice and great ideas for deepening my understanding of the subject. Claire gave me the opportunity to start working on GANs and Signature in the midst of the sanitary crisis, and gave me numerous and brilliant ideas and intuitions on the subject. Adeline helped me a lot in writing code and understanding everything there is to understand about signatures and research in general. Even if short, this experience has been a fantastic way for me to learn a lot about machine learning and statistics.

I want to thank Pierre Gaillard, Emmanuel Trélat, Francis Bach, Guillaume Carlier and Gaspar Rochette for offering me advice about research and mathematics in general and helping me out in difficult times. I also wish to thank Ugo Tanielian who made WGANs less mysterious to me, and the administrative staff of the LPSM for offering me excellent working conditions.

My warmest thanks go to my confinement partners at Montfort. Hugo, Lucas, Tim, Manon, Colin, Éloïse and Isabelle: thank you so much, you offered me a fantastic place to stay and amazing company during the lockdown.

Special thanks go to all the people and teams working on open source software I heavily relied on during my work. Python, Numpy, PyTorch and many other softwares are key tools of modern mathematics and their creators deserve, in my opinion, absolute gratitude from mathematicians and engineers using them.

Finally, I want to thank all friend, family members or teachers, who directly or indirectly made working and learning mathematics a great pleasure: Nicolas Brochier and Basile Morcrette for instilling me the joy of mathematics (and probability theory in particular), Helen, Ulrike, Werner and Marianne Bleistein, Léonore, Yannaï, Marie, Daniela, Marie, Victor, Antoine, Charlotte, Nicolas, Juliette, Arthur, Jules, Chiara, Agathe, Léo, Solène, Anouk, Lucille, Camille, Benjamin, Leila, Fiene, Estelle, Hugues, Emma, Benoît, everybody at La Clef Revival (for cheering me up and showing me movies to clear my head), and all others I cannot mention - acknowledgments must have an end, unfortunately, even though one never says «thank you» enough.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. I have documented all methods, data and processes truthfully, and mentioned all persons who were significant facilitators of my work. I have not manipulated data in any malicious way. All modifications (normalization, rescaling, etc.) of used data are mentioned. I thank the creators of the PenDigits data set, Alimoglu and Alpaydin 1996, which I used extensively.

1 Introduction

New statistical methods grouped under the name of "machine learning" are surprisingly effective for tasks as various as economic growth forecasting, image classification and music generation. They do use various tools, ranging from tools of optimal transport, a field of mathematics created by the French mathematician Gaspard Monge at the end of the 18th century, to theorems and objects coming from the much younger fields of functional analysis, Fourier analysis, wavelets and measure theory. To be effective, they do however often rely on massive amounts of observations: samples need to be big - hence the name "big data" - to produce effective models. Data availability and collection is thus a necessary condition of functional machine learning models, and has become an economic sector in itself. However, in some special cases, useful data cannot be published or even sold, because the data is sensitive, proprietary or private (Koenecke and Varian 2020). One could think of Google protecting its fine granular search data (but publishing aggregates through Google Trends), data from a clinical study that contains private information about patients, or fiscal records that feature information that make taxpayers identifiable. In all these cases, one might want to generate fake data that captures the true distribution of the real data, thus making use of the structure and information contained in the real data. Put mathematically, one wants to approximate the probability distribution of the true data to draw artifical, harmless samples from it.

They are other reasons than sensitivity for generating data. A simple straightforward reason is sample size enhancement: if one can double or triple the size of a dataset without duplicating existing observations, one can compute more precise estimators (Xie et al. 2018). Another one is simulation of extreme values: some industries, as for instance electricity production, have to account for worst case scenarios when designing models - think of an electricity consumption spike that happens because of an unexpected heat or cold wave (Pesch et al. 2015). In certain industries, data might also be very expensive to collect (Kegel, Hahmann, and Lehner 2018), because of costly collection devices or the high amount of noise. Finally, generating complex data is a good proof of efficiency of a given technology or algorithm.

1.1 Related Work

Even though this master thesis only explores generation through GANs (Generative Adversarial Nets), there are numerous other technologies used for data generation, some relying on rather classical statistical approaches, some others leveraging the possibilities of deep learning.

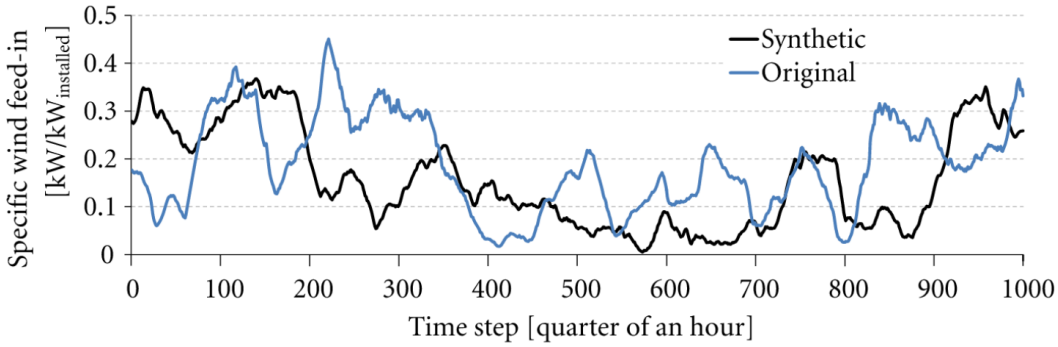


Figure 1: True and synthetic time series of wind power generated through a Hidden Markov Model of order 2. Source: Pesch et al. 2015.

Markov chain based methods. One of the most ancient methods to generate data relies on Hidden Markov Chain (HMC) models. To sample trajectories of a stochastic process $(Z_1, \dots, Z_T)$ given samples of real observations $(X_1, \dots, X_T)$, HMC works by estimating transition probabilities between subsets of the (discretized) state space. One thus needs to make an hypothesis on the order of the Markov process. Even though effective, HMC suffer from many limitations: they are only suited for low order models, as the size of the transition matrices and consequentially the amount of data need to estimate the transition probabilities accurately grows extremely fast; they assume that the underlying model generating the true data is in fact Markovian; they are ill suited to estimate non-homogeneous model, *i.e.* settings in which the transition probabilities vary with time. Applications include wind power time series generation (Pesch et al. 2015) and Chinese calligraphy generation (P. Liu et al. 2011).

Generative Moment Matching Networks. GMMNs were first proposed by Yujia Li, Swersky, and Zemel 2015. This method relies on a deep neural network through which random uniform vectors are propagated in order to create samples that match real data. The model is trained by backpropagation, the loss function computing the distance between different moments of the real and generated data. The model is thus trained to match high order moments of both distribution, resulting in high quality generation. GMMNs can generate speech parameters (Takamichi, Koriyama, and Saruwatari 2017) and faces (Ren et al. 2016), for instance.

Variational autoencoders. Variational autoencorders were introduced by Kingma and Welling 2013. They use an encoding / decoding neural network combined with random inputs to generate data, achieving excellent data generation. VAEs have been used for instance to generate sentences (Bowman et al. 2015), chinese poetry (J. Li et al. 2018) and images (Yan et al. 2016).

Examples of data generated by these different algorithms are shown in Figures 1, 2 and 3.

Data generating models have been widely used for generating text and images, two ubiquitous

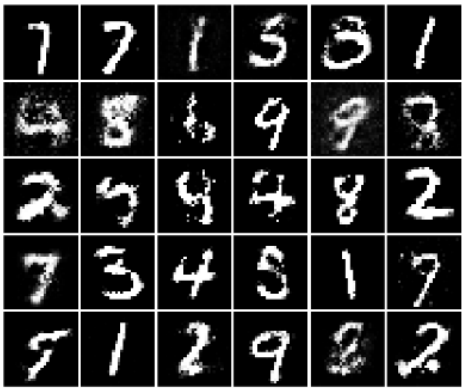


Figure 2: Fake MNIST digits generated by a Generative Moment Matching Network. Source: Yujia Li, Swersky, and Zemel 2015.

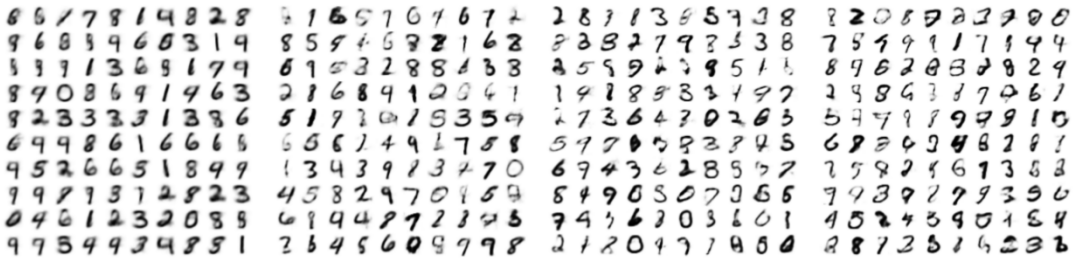


Figure 3: Fake MNIST digits generated by a Variational Auto-Encoder, for size of latent space 2,5,10 and 20 (from left to right). Source: Kingma and Welling 2013

data types in the machine learning literature because of their inherent complexity (which makes them great supports to test deep learning models) and to their omnipresence in modern culture (images and text can be found almost everywhere, and raise no particular problems as far as data collection is concerned).

1.2 Outline

This master thesis, however, deals with generation of time series, a complex data type of high practical importance. A time series simply is a collection of observations of a stochastic process at given sampling times. In practice, time series thus come in the form of a collection of vectors $(\mathbf{X}_1, \dots, \mathbf{X}_T)$, where every $\mathbf{X}_i$ is to be though of as an observation of the stochastic process at time i. Note that this process might very well be high dimensional. Time series are diverse and present in numerous fields, such as health (electrocardiogramms), economics (stock records), energy (energy production and consumption) and engineering. They basically occurs every time a collection of indicators is observed through time. Typical factors of complexity one encouters when dealing with time series are intertemporal dependencies, high amounts of noise, non homogenous data size (times series representing the same phenomenon have different lengths, such as for instance stock records of two compagnies listed on the stock exchange for different periods) and complexity due to high dimensionnality (different dimensions of the time series might interact, possibly through time).

This master thesis explores a combination of two tools in order to generate time series. The first one are (Wasserstein)-Generative Adversarial Networks, a particular class of generative models invented by Goodfellow et al. 2014. GANs rely on solving a minmax problem by combining two neural networks to approximate the density of the true data and create fake data by forwarding gaussian noise through the trained networks. They are one of the most popular and promising data generating technologies with those aforementioned. The second piece of the puzzle are signatures, a non linear transformation of paths and time series that has surprinsing statistical features. Signatures stem from the seminal work done by Chen (K.-T. Chen 1958) in the early 60's, and have been thoroughly analyzed by Lyons since the 90's as key elements of rough path theory. Recently, they have found promising applications in machine-learning-related topics (see for instance Fermanian 2019 and section 2.3.4).

This thesis describes the results of a research internship that concludes the graduate degree "Mathématiques de la modélisation - Spécialité COCV (Contrôle, Optimisation, Calcul des variations)" [1]

[1] Mathematics of Modelling, with specialization in control theory, optimization and calculus of variations.

(Sorbonne Université) and lasted from May 2020 until early August 2020. The main contributions of this work are the following.

1. In the first sections, we provided a global overview of GANs and signatures, from a theoretical and practical point of view.

2. Building upon theoretical work by Chang and T. Lyons 2019, we implement an algorithm to invert signatures, which is a challenging problem that has found very little effective solutions so far.

3. We combine GANs and signature inversion to generate fake time series. Due to computational limitations, the generation performance is modest, but nevertheless gives promising perspectives for data generation.

4. We make use of some specific features of the signature transform (namely the shuffle product and the particular structure of the signature of linear paths) to design some simple metrics for WGAN evaluation.

Our thesis is organized as follows.

- Section 2 provides an overview of GANs, WGANs and signatures from a theoretical and applied point of view. We focuse on properties used in this thesis, and present recent theoretical results as far as GANs and WGANs are concerned.

- In Section 3, we present our data generation pipeline, including the inversion of signatures through the insertion algorithm and signature generation by WGANs.

- Section 4 presents some experimental results.

2 Generative Adversarial Networks (GANs) and Signatures: Some Theoretical Elements

2.1 What Are GANs ?

Generative Adversarial Networks (GANs), sometimes called Generative Adversarial Nets, have been introduced by Goodfellow et al. 2014. GANs fall into the class of data generating models: the goal of a GAN is to learn a probability distribution from a sample of true data, and to learn to produce new samples that are drawn from the same distribution. GANs consists of a discriminator and a generator. Learning is done by adversarial training: while the discriminator learns to distinguish true from fake data, the generator learns to produce fake data. While GANs could theoretically be built with any kind of algorithm, it is common to choose neural networks for both the discriminator and the generator. Put otherwise, a GAN is best described as a technique to train a data generating model, rather than as generation model.

GANs have originally been used in Goodfellow et al. 2014 to generate fake images. Image generation is an important domain of application of GANs. They have been used for a large variety of tasks: they are now recognized as being able to generate images of faces and various other objects that look authentic to human observers. GANs have been used by Radford, Metz, and Chintala 2015 to generate fake altered images (for instance generate images of women with glasses by learning on images of men with and without glasses, and women without glasses). Y. Jin et al. 2017 generate fake anime characters. GANs also perform surprisingly well for different image transformation tasks, as for instance removing rain as in H. Zhang, Sindagi, and Patel 2019 or denoising pictures, as done by J. Chen et al. 2018. Recent applications also include the generation of high resolution versions of low resolution pictures (see for instance Vasu, Thekke Madam, and Rajagopalan 2018) and image reparation (for instance filling missing parts of pictures) as done by Yijun Li et al. 2017.

Apart from image generation and modification, GANs have been used on other types of data. The structure of the trained network is usually modified to best fit the data structure. For instance, Yu et al. 2017 use a RNN to generate Chinese poems and speeches by Barack Obama (an example is displayed in Figure 4). Mogren 2016 uses a RNN to generate music (one can listen to generated pieces here). L. Xu and Veeramachaneni 2018 train a GAN to generate tabular data, i.e. large datasets with many features, both categorical and continuous, including features with high dependencies.

Now we welcome the campaign as a fundamental training to destroy the principles of the bottom line, and they were seeing their own customers. And that's why we will not be able to get a claim of their own jobs. It will be a state of the United States of America. The President will help the party to work across our times, and here in the United States of uniform. But their relationship with the United States of America will include faith.

Thank you. God bless you. And May God loss man. Thank you very much. Thank you very much, everybody. Thank you. God bless the United States of America. God bless you. Here's President.

Figure 4: An excerpt of a fake Obama speech generated using a GAN. Source: Medium post by one of the authors of Yu et al. 2017.

As far as time series generation is concerned, there has been little application of GANs so far to our knowledge. Building upon RNN GANs, Simonetto 2018 generates spiking financial time series. Other applications have focused on generating biosignals (Haradal, Hayashi, and Uchida 2018), sensor data (Alzantot, Chakraborty, and Srivastava 2017) and smart grid data (C. Zhang et al. 2018). Yoon, Jarrett, and Schaar 2019 build a so-called TimeGAN that generates time series and captures temporal dynamics through a specific network architecture.

2.1.1 The Original Problem

The goal of a GAN is to learn a target probability measure μ^* supported on $\mathbb{R}^d$. This measure is assumed to be dominated by a known measure λ (this measure will be the Lebesgue measure most of the time). One should think of this measure as being very complex and not estimable by traditional means like maximum likelihood estimation. Conversely, d is meant to be very large, as the data often consists of images, videos or even music. We assume to have at our disposition a large sample $X_1, \dots, X_n \in \mathbb{R}^d$ draw from μ^*.

Let $G_\theta \in \{G_\theta\}_{\theta \in \Theta}$ and $D_\alpha \in \{D_\alpha\}_{\alpha \in \Delta}$ denote respectively the generator and the discriminator. Both the discriminator and the generator are parametric models (almost always neural networks); thus their parametric spaces should be considered to be very large.

The generator G_θ takes as an input called the noise, most of the time a random variable Z supported on $\mathbb{R}^k$, which will sometimes be called the latent space. Its output lies in $\mathbb{R}^d$. The parameter k is much smaller than d, which is precisely the point of GANs: they are able to "convert" low dimensional noise into high dimensional and diverse outputs. The generator is thus naturally linked to a family of probability measures $\{\mu_\theta\}_{\theta \in \Theta}$, defined through $G_\theta(Z) \overset{\mathcal{L}}{=} \mu_\theta \mathrm{d}\mu$. We will let p_θ denote the density of the measure μ_θ with respect to the measure λ, and p^* the density of the true distribution μ^*.

The discriminator takes as input observations in $\mathbb{R}^d$ and outputs a number in $[0,1]$ that corresponds to the probability that the observation comes from the true probability distribution μ^*. The discriminator is trained to become better and better at distinguishing true from fake data, while the generator's goal is to improve in data counterfeiting: both nets have opposite goals.
Given a sample of gaussian noise $Z_1, \ldots, Z_n$, the problem the GAN tries to solve is

$$\inf_{\theta\in\Theta} \sup_{\alpha\in\Delta} \left[\prod_{i=1}^{n} D_\alpha(X_i) \times \prod_{i=1}^{n} \left(1 - D_\alpha \circ G_\theta(Z_i)\right) \right]. \tag{1}$$

Let us examine this problem closely. The objective function corresponds to the likelihood, as calculated by a given discriminator D_α, that the true data comes from μ^* and that the fake observations $G_\theta(Z_1), \ldots, G_\theta(Z_n)$ do not come from μ^*. In a first time, the objective function is maximized in α given a certain θ: this means that one looks for the best discriminator given a certain generator. Once this discriminator is determined for every $\theta \in \Theta$, one choose the best generator in the sens that it minimizes the maximized objective function: by doing so, one chooses the generator that performs best in fooling the discriminator. This model will thus be able to generate fake observations hopefully indistinguishable from real ones.
Note that in practice, it is common to minimize the natural logarithm of the objective function. Problem (1) thus becomes

$$\inf_{\theta\in\Theta} \sup_{\alpha\in\Delta} \left[\sum_{i=1}^{n} \ln D_\alpha(X_i) + \sum_{i=1}^{n} \ln\left(1 - D_\alpha \circ G_\theta(Z_i)\right) \right]. \tag{2}$$

Note also that we do not assume that μ^* is in $\{\mu_\theta\}_{\theta\in\Theta}$.

2.1.2 Some Theoretical Properties

GANs suffer from little theoretical guarantees. As noted by Gérard Biau et al. 2020, "despite increasingly spectacular applications, little is known about the mathematical and statistical forces behind these algorithms [...] and, in fact, nearly nothing about the primary adversarial problem" (2). This section details some theoretical results on GANs coming from Gérard Biau et al. 2020 and Goodfellow et al. 2014.
A first important theoretical fact is the link between the problem (2) and the Jensen-Shannon divergence.

Definition 1. Let P, Q be two probability distributions which are absolutely continuous with

respect to a measure λ. Their densities are denoted p and q respectively. The Kullback-Leibler divergence from P to Q is defined as

$$D_{KL}(P||Q) := \int p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}\lambda(x).$$

Definition 2. The Jensen-Shannon divergence between P, Q is defined as

$$D_{JS}(P,Q) = D_{KL}\left(P \,||\, \frac{P+Q}{2}\right) + D_{KL}\left(Q||\frac{P+Q}{2}\right).$$

Note that problem (2) is the empirical counterpart of the minimization of the quantity

$$L(\theta, D) = \int \log(D(x)) p^*(x) \mathrm{d}\lambda(x) + \int \log(1 - D(x)) p_\theta(x) \mathrm{d}\lambda(x). \tag{3}$$

We will always consider in the following that the discriminator D is such that $L(\theta, D) > -\infty$ (this property is called θ-admissibility by Gérard Biau et al. 2020).

Goodfellow et al. 2014 show that if one takes D from the class $\mathcal{D}_\infty$ of all Borel functions from E to $[0,1]$, one has

$$\sup_{D \in \mathcal{D}_\infty} L(\theta, D) = L(\theta, D^*_\theta) = 2D_{JS}(\mu^*, \mu_\theta) - \log 4, \tag{4}$$

where $D^*_\theta(x) = \frac{\mu^*(x)}{\mu^*(x)+\mu_\theta(x)}$. Gérard Biau et al. 2020 improve upon this result by giving a uniqueness result.

Theorem 1. Let $\theta \in \Theta$ be such that $\mu_\theta > 0$ λ-ae. Then D^*_θ is the unique maximizer of

$$\sup_{D \in \mathcal{D}_\infty} L(\theta, D). \tag{5}$$

A natural question that derives from this result is the existence of an optimal θ: given μ^*, what is the optimal parameter $\theta \in \Theta$ that gives the best approximation of the unknown density p^* ? Recall that we do not assume that p^* can be written as $p_{\theta'}$ for a particular $\theta' \in \Theta$.

Theorem 2. Assume that the model $\{\mu_\theta\}$ is indentifiable, compact for the metric δ defined through $\delta(P,Q) := \sqrt{D_{JS}(P,Q)}$ and convex. Assume also that p^* is bounded ($m \leq p^* \leq M$) and that $\mu_\theta \leq M$ for all $\theta \in \Theta$. Then there exists θ^* such that

$$\{\theta^*\} = \arg\min_{\theta \in \Theta} L(\theta, D^*_\theta).$$

However, in practice, one always works with parametrized families of generators and discriminators. Letting $\mathcal{D}$ denote the parametrized class of discriminators, we therefor define $\bar{\theta}$ as

$$\bar{\theta} = \arg\min_{\theta \in \Theta} \left[\sup_{D \in \mathcal{D}} L(\theta, D) \right] \tag{6}$$

We thus finally state a result linking some properties of the discriminator class and the generator class to the precision of the approximation of the true distribution.

Theorem 3. Assume that

1. $(\mathbf{H_1})$: there exists $T \in]0, 1/2]$ such that

$$\min(D_\theta, 1 - D_\theta) \geq T, \ \forall \theta \in \Theta.$$

2. $(\mathbf{H_2})$: there exists $\varepsilon \in]0, T[$ and $D \in \mathcal{D}$ a $\bar{\theta}$-admissible discriminator such that

$$\left\| D - D^*_{\bar{\theta}} \right\|_\infty \leq \epsilon.$$

There exists a constant C, depending only upon T, such that

$$0 \leq D_{JS}(\mu^*, \mu_{\bar{\theta}}) - D_{JS}(\mu^*, \mu_{\theta^*}) \leq c\varepsilon^2. \tag{7}$$

Thus, if the discriminator class is big enough, one can approximate the true density in terms of Jensen-Shanon divergence.
Gérard Biau et al. 2020 also provide further properties of GANs, including convergence of $\bar{\theta}$ towards θ^* as the size of the generator class grows.

2.1.3 Taming and Training GANs

Training GANs is notoriously difficult as the model tries to solve a double "inf-sup" problem with a great number of parameters. While conventional neural networks used for instance for classification can produce good results after a few hundred iterations of mini-batch training, GANs require much more training. Following Goodfellow et al. 2014 and Gérard Biau et al. 2020, we suggest using the following training strategy for GANs:

1. The discriminator is trained by maximizing the objective function through one step of gradient ascent.

2. The generator is trained by minimizing the alternative objective function

$$\theta \mapsto -\sum_{i=1}^{n} \ln\left(D_\alpha \circ G_\theta\left(Z_i\right)\right) \tag{8}$$

through one step of gradient descent. While resulting in identical minimizers, this optimization step provides bigger gradients and thus ensures faster convergence of the model.

It is common to train one of parts of the GAN more than the other one (for instance, the discriminator is trained for ten steps and the generator only for one step, as in Gérard Biau et al. 2020): in the algorithm described below, this asymmetry is controlled by the hyperparameter u [2]. Algorithm 1 describes the training scheme with more details. Note that update is here described in terms of classic batch gradient descent for the sake of clarity and illustration of the general training principle; in practice, one often uses more efficient algorithms such as Adam (see Appendix A).

Algorithm 1 GAN training through mini-batch stochastic gradient descent.

1: **Inputs: $X_1, \ldots, X_n$: sample of true data; m: mini-batch size; K: number of training iterations; u: number of training steps for the discriminator for every generator step.**
2: **for** $u \in \{1, \ldots, K\}$ **do**
3: Sample $X_1, \ldots, X_m$ true observations.
4: Sample $Z_1, \ldots, Z_m$ noise observations from $\mathcal{N}(0, 1)$.
5: **Upgrade discriminator** through gradient ascent for u steps:

$$\alpha \leftarrow \alpha + \nabla_\alpha \frac{1}{m} \sum_{i=1}^{m} \left[\ln D_\alpha(X_i) + \ln\left(1 - D_\alpha \circ G_\theta(Z_i)\right)\right].$$

6: **Upgrade generator** through gradient descent:

$$\theta \leftarrow \theta + \nabla_\theta \frac{1}{m} \sum_{i=1}^{n} \ln\left(D_\alpha \circ G_\theta\left(Z_i\right)\right).$$

2.2 From GANs to Wasserstein-GANs with Gradient Penalty (WGAN-GP)

2.2.1 The Wasserstein Distance

The Wasserstein distance is a central object of optimal transport theory (Villani 2003). Informally, optimal transport can be though of as the analysis of optimal (with respect to a certain cost function) transportation plans between sources and objectives. The original problem was formulated by french mathematician Gaspard Monge (1746-1818) in his famous work *Mémoire sur la théorie*

[2] When training WGANs, we found this parameter to be very sensible to the datatype: some setups require large assymetries ($u \sim 10$), and in other cases, a perfectly symmetric training schema works best.

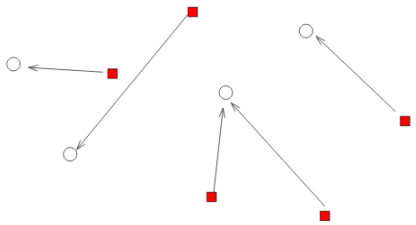


Figure 5: An illustration of the Monge coupling problem: squares stand for sources and circles for targets. Source: Villani 2008.

des déblais et des remblais (1781) (Villani 2008). Assume that one has piles of dirt and holes located in the plane, all of which have the same volume. Assume also that transporting dirt is costly, and that the cost of moving one pile of dirt to a hole is equal to the distance times the volume. What is the optimal transportation plan, i.e. the *coupling* between piles and holes that minimizes the transportation cost ?

In mathematical terms, this problem amounts to match two (probability) distributions while preserving mass. This last condition simply means that no dirt leaves or enters the framework. To fix ideas, consider an easy case where measures are discreet and with countable support. Let $\mu := \frac{1}{n}\sum_{i=1}^{n} \delta_{y_i}$ and $\eta := \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$ be the empirical distributions associated to two families of points, the sources (x_i) and the targets (y_i). Let $c : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^+$. be the cost function that associates to two points in the plan the cost to transport one unit of mass from one to the other (this function often is a distance). The Monge problem writes

$$\min_{\sigma \in S(n)} \sum_{i=1}^{n} c(x_i, y_{\sigma(i)}) \tag{9}$$

where $S(n)$ stands for the set of permutations of $\{1, \dots, n\}$.

In this discrete setting, mass is automatically preserved as one transports from points to points and all points have unit mass. Finding the optimal permutation amounts to finding an optimal pairing between sources and targets. Monge extended his seminal ideas to a 3-dimensionnal setting with continuous densities (see Villani 2008 for a detailed description of Monge's work).

The modern formulation of optimal transport, called the Monge-Kantorovitch problem, is due to a reformulation and relaxation of the Monge problem by sovietic mathematician Leonid Kantorovitch. Let $(\mathcal{X}, \mu)$ and $(\mathcal{Y}, \eta)$ be two probability spaces.

Definition 3. A coupling of μ and η is a couple of random variables (X, Y) such that $\text{Law}(X) = \mu$ and $\text{Law}(Y) = \eta$.

Remark. This amount exactly to building a density γ on $\mathcal{X} \times \mathcal{Y}$ such that its marginals are equal to μ and η. A coupling thus is a way to transport mass allocated according to μ to locations placed according to ν. The coupling condition can be seen as a mathematical formulation of mass conservation. Indeed, if γ couples μ and η, and one sees, for every measurable subsets $A \in \mathcal{X}$ and $B \in \mathcal{Y}$, $\gamma(A \times B)$ as the quantity of mass transported from A to B, the coupling condition reads

$$\gamma(A \times \mathcal{Y}) = \mu(A) \text{ and } \gamma(\mathcal{X} \times B) = \eta(B),$$

which precisely means that *all* mass coming from A must be allocated to some targets, and *all* mass allocated to a region of targets must come from some sources.

Let $\Gamma(\nu, \eta)$ denote the set of all couplings of μ and η. We can now state the optimal transport problem.

Definition 4. Let $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ be a cost function. The Monge-Kantorovitch problem consists in solving

$$\inf_{\gamma \in \Gamma(\mu,\eta)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y) \mathrm{d}\gamma(x, y). \tag{10}$$

Solving this problem means finding a transportation plan or coupling such that it minimizes the transportation cost between two measures among all transportation plans. Indeed, the objective function can be interpreted as the infinitesimal sum of transported quantities $\mathrm{d}\gamma(x, y)$ times the cost of transport between both locations.

As optimal transport is an exploding field, there are numerous properties and known facts about particular cases and settings of this problem. We will only state those needed to properly define the Wasserstein distance and refer the curious reader to Villani 2008 and Villani 2003 for an extensive literature review of the subject. For applications to machine learning and data science, and computational aspects, the reader is referred to the exhaustive book by Peyré, Cuturi, et al. 2019.

Let us now start our quick tour of optimal transport. To quote Villani 2003, "the first good thing about optimal couplings is that they exist".

Proposition 1. Assume that $c(\cdot, \cdot)$ is lower semi-continuous and that the spaces $\mathcal{X}$ and $\mathcal{Y}$ are compact. The problem (10) has a solution.

Remark. This means that problem (10) can be rewritten as

$$\min_{\gamma \in \Gamma(\mu,\eta)} \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\mathrm{d}\gamma(x,y). \tag{11}$$

The Wasserstein distance between two probability measures is defined as the minimal cost of transportation between those measures, when the associated cost is itself a distance.

Definition 5. Let $d(\cdot,\cdot)$ be a distance on $\mathcal{X}\times\mathcal{X}$. The Wasserstein distance $W_p(\cdot,\cdot)$ between two probability measures μ,η is defined as

$$W_p(\mu,\eta) := \left[\min_{\gamma\in\Gamma(\mu,\eta)} \int_{\mathcal{X}\times\mathcal{X}} d(x,y)^p \mathrm{d}\gamma(x,y)\right]^{1/p}. \tag{12}$$

Focusing on the distance W_1, we will state one last fact, known as Kantorovitch-Rubinstein duality. This reformulation of the problem through convex duality is a central element of the application of optimal transport to the training of GANs.

Proposition 2. One has

$$W_1(\mu,\eta) = \sup_{f:\mathcal{X}\to\mathbb{R}} \left\{ \int_{\mathcal{X}\times\mathcal{X}} f(z)\mathrm{d}(\mu-\eta)(z), \quad \mathrm{Lip}(f)\leq 1 \right\}. \tag{13}$$

Remark. Lip$(\cdot)$ stands for the Lipschitz constant of a function. Note that liptschitz functions are always continuous.

Alternatively, one can rewrite (13) in probabilistic terms as

$$\sup_{\mathrm{Lip}(f)\leq 1} \mathbb{E}_{X\sim\mu}\left[f(X)\right] - \mathbb{E}_{Y\sim\eta}\left[f(Y)\right]. \tag{14}$$

This last formulation of the Wasserstein distance is the central element of Wasserstein-GANs.

2.2.2 Introducing the Wasserstein Distance in GANs

The previous section alluded to the following fact, which must be stated clearly.

Proposition 3. $W_1(\cdot,\cdot)$ is a distance on $\mathcal{P}_1(\mathcal{X})$, the space of probability measures supported on $\mathcal{X}$ that have a finite first moment.

Proof. While symmetry and separation are easy to prove, the triangular inequality is a bit more tricky to get and relies on the so-called "gluing Lemma".
Let μ_1,μ_2,μ_3 be threee probability measures on $\mathbb{R}^d$. Let (X_1,X_2) be an optimal coupling of μ_1

and μ_2 and (Z_2, Z_3) be an optimal coupling of μ_2 and μ_3. The gluing Lemma states that one can glue together these two couplings, resulting in a collection (Y_1, Y_2, Y_3) such that $\text{law}(Y_1, Y_2) = \text{law}(X_1, X_2)$ and $\text{law}(Y_2, Y_3) = \text{law}(Z_2, Z_3)$. One thus has

$$W_1(\mu_1, \mu_3) \leq \mathbb{E}d(Y_1, Y_2)$$

since the Wasserstein distance is defined through optimality of transport plans, and

$$\begin{aligned}\mathbb{E}d(Y_1, Y_3) &\leq \mathbb{E}d(Y_1, Y_2) + \mathbb{E}d(Y_2, Y_3) \\ &= W_1(\mu_1, \mu_2) + W_1(\mu_2, \mu_3)\end{aligned}$$

by using the triangular inequality and by optimality of the transport plans defined above. When $p \geq 1$, one needs to use the Minkovski inequality for the last steps. □

The goal of a GAN is to approximate a probability measure, or, to put it otherwise, to reduce the distance between the unknown measure that characterizes the data and an artificial measure created by the generating mechanism. It is thus natural to introduce the Wasserstein distance to capture the similarity of those two measures. This was first done by Arjovsky, Chintala, and Bottou 2017. Stepping from a GAN to a W-GAN simply consists in changing the objective function: the network does not try to maximize-minimize the Jensen-Shanon divergence anymore, but tries to minimize the Wasserstein distance between two measures. However, since the Wasserstein distance is the solution of a complex optimization problem and does not have a close form solution most of the time, the problem also includes a maximization step to approximate W_1 through the reformulation (14). The goal of a WGAN is thus to solve the problem

$$\inf_{\theta \in \Theta} \sup_{\alpha \in \Delta} \left[\mathbb{E}_{X \sim \mu^*} \left[D_\alpha(X) \right] - \mathbb{E}_{Z \sim \mathcal{N}(0,1)} \left[D_\alpha \circ G_\theta(Z) \right] \right], \text{ s.t. } \text{Lip}(D_\alpha) \leq 1, \tag{15}$$

through solving its empirical counterpart

$$\inf_{\theta \in \Theta} \sup_{\alpha \in \Delta} \frac{1}{n} \left[\sum_{i=1}^{n} \left[D_\alpha(X_i) \right] - \sum_{i=1}^{n} \left[D_\alpha \circ G_\theta(Z_i) \right] \right], \text{ s.t. } \text{Lip}(D_\alpha) \leq 1, \tag{16}$$

where $X_1, \ldots, X_n$ is a sample of true data with density p^*, and $Z_1, \ldots, Z_n$ is a gaussian sample.

First, notice the change in nature of the discriminator. In a GAN, the discriminator is trained to produce a probability. In a WGAN, the discriminator no longer acts as a judge assigning probabilities to observations, but as a distance approximator. Secondly, one should notice that the optimization problem is now constraited by a Lipschitz constraint, which can be reformulated as

$$\left\|\nabla_x D_\alpha(x)\right\| \leq 1 \text{ a.e.} \tag{17}$$

This constraint introduces an additional difficulty in training the WGAN, which is usually addressed by clipping or gradient penalty.

Clipping is a somewhat brutal method to enforce the Lipschitz constraint. It consists in constraining all parameters $(\alpha_1, \ldots, \alpha_p) = \alpha$ of the discriminator to be in a small compact set, $[-0.01, 0.01]$ for instance (as in Arjovsky, Chintala, and Bottou 2017). This makes the parameter space compact and thus enforces Lipschitz constraint up to a multiplicative constant. At every training step, weights are checked and simply pushed to the closest threshold if they are to big. As Arjovsky, Chintala, and Bottou 2017 put it, "weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs). We experimented with simple variants (such as projecting the weights to a sphere) with little difference, and we stuck with weight clipping due to its simplicity and already good performance.However, we do leave the topic of enforcing Lipschitz constraints in a neural network setting for further investigation, and we actively encourage interested researchers to improve on this method".

A notable improvement on this technique was provided by Gulrajani et al. 2017. The new technique to enforce the Lipschitz constraint, called gradient penalty (GP), simply consists in relaxing the problem through a soft penalization of the constraint. The WGAN-GP problem thus writes

$$\inf_{\theta\in\Theta} \sup_{\alpha\in\Delta} \frac{1}{n}\left[\sum_{i=1}^{n}\left[D_\alpha(X_i)\right] - \sum_{i=1}^{n}\left[D_\alpha \circ G_\theta(Z_i)\right]\right] + \lambda\frac{1}{n}\sum_{i=1}^{n}\left(\left\|\nabla D_\alpha(Y_i)\right\| - 1\right)^2. \tag{18}$$

λ is a hyperparameter which controls the importance of the gradient penalty. Since enforcing the constraint almost everywhere is impossible, the constraint is only enforced on a sample of points $Y_1, \ldots, Y_n$. These points are sampled uniformly along lines connecting the real and fake

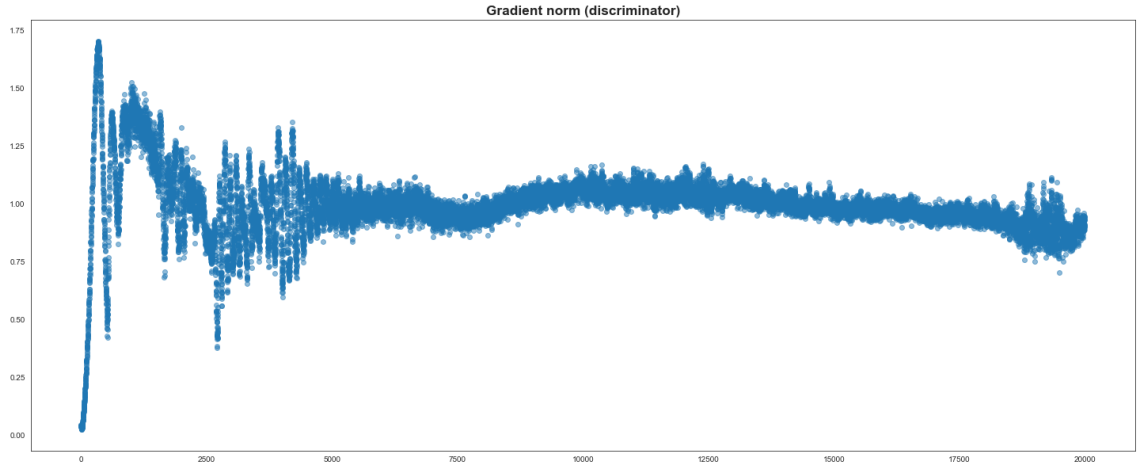


Figure 6: Evolution of the norm of the discriminator's gradient during the training of a WGAN with gradient penalty. As the training of the networks goes further, the discriminator's gradient norm tends toward 1, after oscillating in the first stages of training. The trained WGAN approximates a density in 2D and is fully described in section 4.2.

observations. Note that the discriminator's gradient is constrained not to be smaller than 1 but close to 1: this is motivated by empirical results and the fact; furthermore, according to the authors of the aforementioned paper, the gradient of the optimal critic is most of the time close to 1. Figure 6 shows the evolution during training of the discriminator's gradient norm of the WGAN described in section 4.2.

Algorithm 2 presents the training strategy with more details. Optimization is done with Adam optimizer. For further details on this optimization algorithm, see Appendix A.

2.2.3 Theoretical Properties of WGANs

Gérard Biau, Sangnier, and Tanielian 2020 recently gave some theoretical insights into WGANs, following the seminal work of Liang 2018 and P. Zhang et al. 2017. We here present a few results similar to those presented for GANs in section 2.1.2.

A first guarantee one wishes to have when working with WGANs is the existence of a maximizer among the class of discriminators (in our case, neural networks) one considers. The generator and discriminator can in this framework be written as

$$G_\theta(z) = U_p \quad \sigma\left(\quad U_{p-1} \quad \cdots \quad \cdots \quad U_2 \quad \sigma\left(U_1 z + b_1\right) + b_2 \quad \right) \cdots + b_{p-1} \Bigg) + b_p$$

and

Algorithm 2 WGAN training through mini-batch stochastic gradient descent with GP.

Inputs: $\mathbf{X_1}, \ldots, \mathbf{X_n}$**: sample of true data;** $\boldsymbol{m}$**: mini-batch size;** $\boldsymbol{K}$**: number of training iterations;** λ**: Gradient Penalty parameter;** $\alpha, \beta_\mathbf{1}, \beta_\mathbf{2}$**: Adam optimizer parameters.**
for $k \in \{1, \ldots, K\}$ **do**
  Sample $X_1, \ldots, X_m$ true observations.
  Sample $Z_1, \ldots, Z_m$ noise observations from $\mathcal{N}(0, 1)$.
  for $i \in \{1, \ldots, m\}$ **do**
    Sample $\varepsilon \sim \mathcal{U}[0, 1]$.
    $Y_i \leftarrow \varepsilon X_i + (1 - \varepsilon) G_\theta(Z_i)$
    $L^{(i)} \leftarrow D_\alpha \circ G_\theta(Z_i) - D_\alpha(X_i) + \lambda \left(\left\| \nabla D_\alpha(Y_i) \right\| - 1 \right)^2$
  Upgrade discriminator through gradient ascent:

$$\alpha \leftarrow \text{Adam}\left(\nabla_\alpha \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, \alpha, \beta_1, \beta_2 \right).$$

  Upgrade generator through gradient descent:

$$\theta \leftarrow \text{Adam}\left(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, \alpha, \beta_1, \beta_2 \right).$$

$$D_\alpha(x) = V_q \quad \tilde{\sigma} \Big(\quad V_{q-1} \quad \cdots \tilde{\sigma} \left(V_2 \tilde{\sigma} \left(V_1 x + c_1 \right) + c_2 \right) + \cdots + c_{q-1} \Big) + c_q.$$

σ and $\tilde{\sigma}$ are the network's activation functions, resp. rectifier and Groupsort. The matrices U_i and V_i are the matrices of weights of the generator and the discriminator respectively. The b_i's and c_i's are offset vectors.

We let $\mathcal{D}$ denote the parametrized class of discriminators.

The two following assumptions will always be assumed to hold true. $\|.\|_2$ denotes the operator norm for matrices derived from the 2-norm on vectors, i.e. $\|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2$. The matrice norm $\|.\|_\infty$ is defined in the same manner for the supremum vector norm.

1. First, we assume that for all $\theta = \left(U_1, \ldots, U_p, b_1, \ldots, b_p \right) \in \Theta$,

$$\max \left(\|U_i\|_2, \|b_i\|_2 : i = 1, \ldots, p \right) \leqslant K_1$$

   where $K_1 > 0$ is a constant.

2. Besides, we assume that for all $\alpha = \left(V_1, \ldots, V_q, c_1, \ldots, c_q \right) \in \Lambda$,

$$\|V_1\|_{2,\infty} \leqslant 1, \max \left(\|V_2\|_\infty, \ldots, \left\|V_q\right\|_\infty \right) \leqslant 1, \text{ and } \max \left(\|c_i\|_\infty : i = 1, \ldots, q \right) \leqslant K_2$$

   where $K_2 \geqslant 0$ is a constant.

These two assumptions ensure that all discriminators we consider are indeed 1-Lipschitz.

Theorem 4. For every $\theta \in \Theta$, there exists $\alpha \in \Lambda$ such that

$$|\mathbb{E}_{\mu^*}(D_\alpha) - \mathbb{E}_{\mu_\theta}(D_\alpha)| = \sup_{f \in \mathcal{D}} |\mathbb{E}_{\mu^*} f - \mathbb{E}_{\mu_\theta} f|. \tag{19}$$

In plain language, this theorem means that for every generator, there exists a perfect discriminator for every considered generator. Note that if one had access to all 1-Lipschitz functions and could choose f within this larger class, this theorem would simply be the existence of an optimal transport plan through Kantorovitch duality. Uniqueness, however, cannot be guaranteed without further assumptions.
Furthermore, one also has a guarantee of existence of a generator that minimizes this quantity.

Theorem 5. There exists at least one solution to the problem

$$\inf_{\theta \in \Theta} \sup_{f \in \mathcal{D}} |\mathbb{E}_{\mu^*} f - \mathbb{E}_{\mu_\theta} f|. \tag{20}$$

Note that this statement only relies on the two assumptions made earlier and does not require compactness of E.

2.2.4 Advantages and Drawbacks from Using WGANs

GANs notoriously suffer from numerous recurrent problems (Weng 2019). They include, but are not limited to:

1. **Mode collapse.** Mode collapse is said to occur when the generator fails to produce diverse observations and only produces one type of object. This is a problem as the goal of GANs is to produce samples as diverse as the original data.

2. **No convergence**. One can easily reformulate problem (1) in terms of game theory: the goal of training GANs is to attain a Nash equilibrium in a two-player non cooperative game. However, this can sometimes fail, resulting in oscillating models with no convergence.

3. **Low dimensional support.** This issue is specifically addressed by Arjovsky and Bottou 2017. In fact, even though real data (such as images) might look highly complex, its probability density often concentrates on a low-dimensional manifold. Since the generator also lies

in a low dimensional space, it is likely that their densities' supports do not overlap. A consequence is that the discriminator will become almost perfect quickly, resulting in vanishing gradients.

4. **Vanishing gradient.** As the discriminator gets better and better, and eventually perfect, the gradient of the loss function tends toward 0, thus making training impossible. This is precisely why the alternative loss function (8) is often used in practice.

WGAN were introduced as an attempt to solve some of these problems. As noticed by Arjovsky, Chintala, and Bottou 2017, WGANs are less prone to mode collapse and do generate diverse samples. They come, however, at the price of sacrificing the interpretability of the discriminator. When training a GAN, one can assess the quality of the model by testing the discriminator on generated samples: the output of the discriminator can be interpreted as the probability that the discriminator classifies the generated sample as a true sample. The generator is perfectly trained if the discriminator classifies its output to be true data with probability 0.5: this amounts to the discriminator flipping a coin. When training WGANs, the output of the discriminator corresponds to an approximation of the Wasserstein distance and is less interpretable.

2.3 Signatures

A recurrent problem in computer science and machine learning is dimension reduction: how can one summarize information contained in large and complex object in simpler objects, without loosing too much information, and ideally, while keeping the possibility to invert this compression process ? This fundamental question has motivated the use of Fourier basis and wavelets for image, sound and signal compression (see for instance Mallat 1999).

Signatures are among the tools initially designed by pure mathematicians for pure mathematics, and which later on found surprising applications in machine learning. Its origins trace back to the work of K.-T. Chen 1958. Lyons pioneered its use in rough path theory (see for instance T. J. Lyons, Caruana, and Lévy 2007b). They allow for path compression but without being a transformation based on a base of functions (as Fourier or wavelets).

Signatures can be seen as a highly non linear transformation that maps paths (functions from a real time interval $[0, T]$ into a vector space V) into tensor product of V. This spaces, even though quite complex and difficult to analyse, can be easily identified with real vector spaces like $\mathbb{R}^d$ as we will explain with greater detail later on. This makes it possible to map complex, high-dimensional paths into simpler vectors of $\mathbb{R}^d$ while conserving a great amount of statistical information. Another advantage of signatures, which is of great importance when dealing with time

series, is its invariance with respect to path length: the signature transform allows to compare and to use standard statistical techniques (classification, regression) on time series that are of different length, thus making techniques such as rescaling or path truncature unnecessary. Finally, the signature transform has been found to have surprising applications in and links with probability theory and statistics, in particular when applied to multimodal datastreams that are structured by complex inter-dimensional correlations, both in time and space. This section provides formal definition of the signature transform, fundamental properties and applications to machine learning problems.

2.3.1 Preliminaries

Signatures can be described in a very abstract and formal manner, using tensor products, tensor spaces and abstract rougth paths. We will, however, present this tool in a simplified framework, both for the sake of clarity and because it is well suited to our ends. Thus, in the following, we will always talk of paths as functions from $[0,1]$ to $\mathbb{R}^d$, but the reader seeking generalizations or familiar with rough path theory should keep in mind that things can be generalized to $(V,\|.\|)$, an arbitrary Banach space of finite dimension. Paths will be denoted by $X_t := X(t)$ to lighten notations. The subscript denotes time, while the superscript will denote a given dimension of the path : $X^i_{t_j}$ is the i-th component of the path, evaluated at the time moment t_j.
To make rough paths a little less wild, one must first equip the space of paths with a good norm. The bounded variation norm is well suited for this purpose.

Definition 6. The p-variation of X is defined as

$$\|X\|_{p-var} := \left[\sup_D \sum_{t_i \in D} \|X_{t_i} - X_{t_i - 1}\|^p \right]^{1/p}, \tag{21}$$

where the supremum is taken on all possible finite partitions $D = \{0 = t_1 < \cdots < t_k = 1\}$ of the interval $[0,1]$.

Note that $\|.\|_{p-var}$ is a semi-norm, since the p-variation of constant paths is 0. A simple way to turn this semi-norm into a full norm is to consider the so-called *bounded variation norm*

$$\|X\|_{BV^p} := \|X\|_{p-var} + \sup_{z \in [s,t]} \left\|X(z)\right\|. \tag{22}$$

The space of paths that have finite bounded variation norm is denoted $BV^p(V)$. $(BV^p(V),\|.\|_{BV^p})$ is a Banach space (T. J. Lyons, Caruana, and Lévy 2007a). One can think of this norm as the

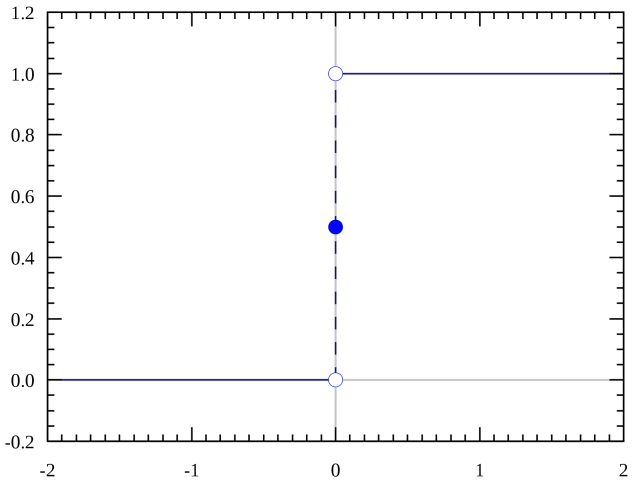


Figure 7: The Heaviside step function has a p-variation of 1 for any $p \geq 1$ on any interval that includes 0 in its iterior.

maximal sum of increments, or variations, that can occur along a path. For instance, a step function like the Heaviside function (Figure 7), has a p-variation of 1 on the interval $[-1, 1]$, and its BV^p norm is 2.

To define the signature transform, we also need to give some details about the Riemann-Stieljes integral, which is a generalization of the Rieman integral and is well suited for integration of rough paths.

Definition 7. Let $X, Y : [0, 1] \to \mathbb{R}^d$ be two paths, and $D_n = \left\{0 = t_1^n < \cdots < t_k^n = 1\right\}$ a sequence of partitions which mesh size $\|D_n\| = \sup_{1 \leq i \leq k} |t_i - t_{i-1}|$ tends to 0. For $s_i^n \in [t_i^n - t_{i-1}^n]$ for all i, n, if the sum

$$\sum_{t_i \in D_n} Y_{s_i} \left(X_{t_i} - X_{t_i - 1}\right) \in \mathbb{R}^d \tag{23}$$

converges to a limit independent from (s_i^n) and from D_n, this limit is called the Riemann-Stieljes integral of Y against X, and is denoted by

$$\int_s^t Y_t dX_t \in \mathbb{R}^d.$$

Intuitively, one can think of the classic Riemann integral as a integral where the size of increments is fixed (and uniformly equal to the length of the segment on which the function is integrated). The Riemann-Stieljes allows for varying increment size. Readers familiar with the Itô integral will also notice the strong resemblance between those two objects: the Riemann-Stieljes

integral can be seen as a deterministic version of the Itô integral.

2.3.2 First Definitions

A path $X_t \in \mathbb{R}^d$ can be though of a collection of d signals living in $\mathbb{R}$, that possibly interact through time. In a simplified way, the signature transform captures the variations of each signal with respect to every other through integration from one dimension against another. It also appears naturally when computing solutions to controlled differential equations (we do not discuss this topic, which is covered in Chevyrev and Andrey Kormilitzin 2016 for instance). To define it properly, we must first define the signature coefficient associated to a *word* of length m of the alphabet $\{1, \dots, d\}$, meaning a ordered collection of m elements from $\{1, \dots, d\}$ with allowed repetitions (i.e. an object exactly constructed as one constructs real words from the set of letters of the latin alphabet).

Definition 8. Let $K = (i_1, \dots, i_m)$ be a word of size m of the alphabet $\{1, \dots, d\}$. The signature coefficient associated with the word K, denoted by $S^K(X)$ is

$$S^K(X) := \int_{0<z_1<\cdots<z_m<1} dX_{z_1}^{i_1} \dots dX_{z_n}^{i_m} \in \mathbb{R}. \tag{24}$$

The signature is an iterated integral, and is thus very sensible to the order of integration : when permuting two letters within the word K, there is no reason for the two signatures to be equal. The number of words of length m one can form from the alphabet $\{1, \dots, d\}$ is finite and equal to m^d. We will write them $W_1^m, \dots, W_{d^m}^m$: the superscript denotes the length of the word, while the subscript stands for the number of the word (words can be ranked by lexicographical order, which is detailed in Appendix B.1). The signature of a path is a infinite series that collects, for every length $m \geq 1$, all possible signature coefficients that can be computed with words of length m.

Definition 9. Let $X : [0, 1] \to \mathbb{R}^d$ be of bounded variation. The signature of X is the infinite series equal to

$$S(X) := \left(1, \mathbf{S}^1(X) \dots, \mathbf{S}^m(X), \dots\right), \tag{25}$$

where the m-th coefficient of S, $\mathbf{X}_m$, is equal to

$$\mathbf{S}^m := \left(S^{W_1^m}, \dots, S^{W_{d^m}^m}\right) \in \mathbb{R}^{d^m}. \tag{26}$$

Formally speaking, the signature is an element of the infinite tensor product space. The inter-

ested reader is referred to Fermanian 2018 for further details. Since our goal is to use signatures within a machine-learning framework, in which a computer will perform vectorized operations on signatures, identifying tensor spaces with real vector spaces is a natural simplification.
Dealing with infinite objects like the signature is highly impractical. An object one will often consider when using the signature for machine learning is the truncated signature.

Definition 10. The truncated signature at order N is the finite collection of signature coefficients

$$S_N(X) := \left(1, \mathbf{S}^1, \dots, \mathbf{S}^N\right) \in \mathbb{R}^{\frac{d^{N+1}-1}{d-1}}. \tag{27}$$

The truncated signature of order N lives in the space $\mathbb{R}^{\frac{d^{N+1}-1}{d-1}}$: there are d words of length 1, d^2 words of length 2, etc. to which one generally adds the first coefficient equal to 1 (which is the word of length 0 in a certain sens), and finally, the number of words of length between 0 and N is $\sum_{i=0}^{N} d^i = \frac{d^{N+1}-1}{d-1}$. When using signatures for tasks like classification, this is how it will be concretely used. The object one will encounter in practice thus looks like

$$S_N(X) = \left(1, S^{(1)}(X), \dots, S^{(d)}(X), S^{(1,1)}(X), \dots, S^{(d,d)}(X), \dots, S^{(d,\dots,d)}(X)\right) \in \mathbb{R}^{\frac{d^{N+1}-1}{d-1}}.$$

The first coefficient 1 is always dropped, since it adds no information to the signature.

Remark. This means that the size of the signature grows exponentially fast. The signature taken at higher order contains more and more information, but is also bigger and bigger: this is a trade-off one needs to keep in mind when using the signature in machine learning.

2.3.3 Examples and properties

Signature are complex objects. This section gives a few simple examples to fix ideas.

Example 1. (Signature in 2 dimensions). Let $X = (X_1, X_2)$. The signature at order 2 thus is

$$\begin{pmatrix} \int_0^1 \int_0^{u_2} dX_1(u_1)dX_1(u_2) & \int_0^1 \int_0^{u_2} dX_1(u_1)dX_2(u_2) \\ \int_0^1 \int_0^{u_2} dX_2(u_1)dX_1(u_2) & \int_0^1 \int_0^{u_2} dX_2(u_1)dX_2(u_2) \end{pmatrix}.$$

Example 2. (Signature of a simple path). Consider the path $X(t) = (t, f(t))$, for t in $[0, 1]$, which is simply the representation of the graph of a function as a path. Assume that $f(0) = 0$. Simple calculations yield

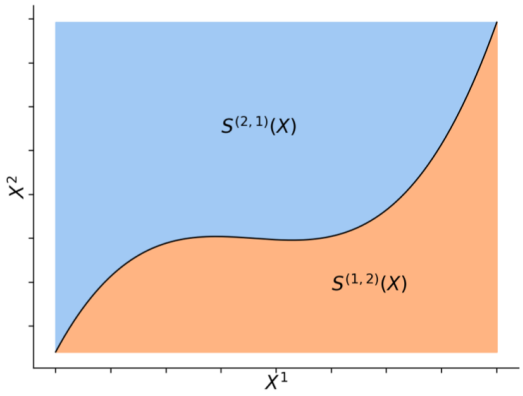


Figure 8: Graphical interpretation of signature coefficients. Source: Fermanian 2019.

$$S^{(2,1)}(X) = \int_{0<u_1<u_2<1} df(u_1)du_2 = \int_0^1 f(u_2)du_2,$$
$$S^{(1,2)}(X) = \int_{0<u_1<u_2<1} du_1 df(u_2) = f(1) - \int_0^1 f(u_2)du_2.$$

This allows for a graphical interpretation of the signature: the two calculated coefficients capture the area below and above the curve of f (see figure 8).

The following example is somewhat crucial when using the signature in machine learning. When using paths in practice, one can only sample them at precise time points: linking the values of the stochastic process at these time points thus yields a piece wise linear path. The good news is that it is very easy, from a computational point of view, to compute the signature of these kind of paths. Let us first compute the signature of a linear path.

Example 3. (Signature of linear paths). When paths are linear, i.e. when

$$X_t = X_0 + (X_1 - X_0)t = \begin{bmatrix} a_1 t + b_1 \\ \vdots \\ a_n t + b_n \end{bmatrix},$$

the signature is easy to compute. Indeed, for any word $K = (i_1, \dots, i_n)$, one has

$$S^K(X) = \frac{a_{i_1} \cdots a_{i_n}}{n!}.$$

To compute the signature of piece wise linear paths, we will need one small lemma, that makes

the link between those paths and simply linear paths.

Proposition 4. (Chen's identity). Let $a \in]0,1[$, and let $X : [0,a] \to \mathbb{R}^d$ and $Y : [a,1] \to \mathbb{R}^d$ be two continuous paths of bounded variation. Let $X * Y$ be the concatenated path obtained by concatenating X and Y, i.e. by gluing together the two paths. Then one has

$$S(X * Y)_{[0,1]} = S(X)_{[0,a]} \otimes S(Y)_{[a,1]}, \tag{28}$$

where $S(X)_{[x,y]}$ denotes the signature with integration on the time segment $[x,y]$, and $\otimes$ is the Kronecker product between two vectors.

This lemma is key to computing the signature of time series. Indeed, one can see that by iterating Chen's identity, it is easy to compute the signature of a piece wise linear path of multiple pieces. The full signature of such a path is simply equal to the Kronecker product of the signatures computed on all pieces: if $X : [0,1] \to \mathbb{R}^d$ is such a path, i.e. linear on all pieces of a partition of $[0,1]$ of the form $\big\{[t_i, t_{i+1}], 1 \leq i \leq N\big\}$, one has

$$S(X)_{[0,1]} = \bigotimes_{i=1}^{N} S(X)_{[t_i,t_{i+1}]}. \tag{29}$$

Last but not least, we will state two more properties of the signature: the first one is an interesting structure that characterises the signature and will help us determine later on if the vectors we generate are indeed signatures; the second one is a property about the approximation capacities of the signature transform.

Before going further, let us briefly recapitulate the notations used to avoid any confusion :

- $S(X)$ denotes the full signature (an infinite series) of a path X.
- For a word K, $S^K(X)$ stands for the signature coefficient associated with this word (a real number).
- $S_N(X)$ is the full signature truncated at order N.
- $\mathbf{S}^n(X)$ is the collection of all signature coefficients associated with words of length n.
- The notation $S_{[a,b]}(X)$, used to denote the full signature but with integration on the segment $[a,b]$, is specific to Proposition 4 and will not be used anymore.

The signature coefficients associated with two words are surprisingly directly linked to the coefficients associated with the elements of the shuffle product of these two words. The shuffle product

is a natural operation one can perform on words: it consists in forming all possible words by concatenating the two initial words and permuting the letters, but with the additional constraint that the initial letter order - the order within the words - must be preserved.

Definition 11. (Shuffle product) Let $I = (i_1, \dots, i_n)$ and $J = (j_1, \dots, j_m)$ be two words from the alphabet $\{1, \dots, d\}$ of length n and m. Their shuffle product, denoted $I \sqcup J$ is defined as the set of permutations of the letters of the word $(i_1, \dots, i_n, j_1, \dots, j_m)$ that preserves the order within the words I and J.

Example 4. Let $I = (1, 2)$ and $J = (3, 4)$. The word $(1, 3, 2, 4)$ is in $I \sqcup J$, but $(1, 4, 3, 2)$ is not, since the ordering of the word J imposes that the letter 3 must be placed before the letter 4.

Example 5. Using real words, the shuffle product of the words "yes" and "no" includes "yenos", "ynoes" and "yesnso", but not "yeons" or "yonse".

One can obtain the product of two signature coefficients associated to two words (which we recall are two real numbers) by shuffling the words and summing up all the signature coefficients.

Proposition 5. (Shuffle identity) Let I, J be two words from the alphabet $\{1, \dots, d\}$ of length n and m. One has

$$S^I(X)S^J(X) = \sum_{K \in I \sqcup J} S^K(X) \in \mathbb{R}. \tag{30}$$

Example 6. For instance, one has

$$S^{(1)}(X)S^{(2)}(X) = S^{(1,2)}(X) + S^{(2,1)}(X)$$

and also

$$\begin{aligned} S^{(1,2)}(X)S^{(3,4)}(X) =& S^{(1,2,3,4)}(X) + S^{(1,3,4,2)}(X) + S^{(1,3,2,4)}(X) \\ &+ S^{(3,4,1,2)}(X) + S^{(3,1,4,2)}(X) + S^{(3,1,2,4)}(X). \end{aligned}$$

This section ends with an approximation theorem that is central for the use of signature in Machine Learning. It uses the concept of tree-like paths, which for the sake of simplicity is not presented here but in Appendix B.2.

Proposition 6. Let D be a compact subset of $BV_1(\mathbb{R}^d)$ of paths that are not tree-like equivalent. Let f be a continuous function (for the 1-variation norm) over this subset, i.e. a function of paths,

that maps paths to $\mathbb{R}$. For every $\varepsilon > 0$, there exists a integer N and $w \in \mathbb{R}^N$ such that **for any** $X \in D$

$$|f(X) - w \cdot S_N(X)| \leq \varepsilon. \tag{31}$$

Remark. $\cdot$ denotes the usual scalar product. For a proof of this Proposition, the reader is refered to Király and Oberhauser 2016. This theorem crucially motives the use of signatures in machine learning: it states that any function that maps time series to the reals and is continuous can be approached by a linear combination of the path's signature coefficients.

2.3.4 Applications of the signature in machine learning

The signature transform is becoming an increasingly popular tool in the field of machine learning. This is mainly due to its capacity to handle time series of different length, to its statistical capacity to capture complex phenomena of dimensional correlation, to its ability to approach real functions of paths (see Proposition 6) and also to its simple form when paths are linear or piecewise linear. Indeed, as time series are samples from a possibly continuous stochastic process at given times, they always are piecewise linear paths. To compute their signature, one can use the Proposition 4. We give some examples of applications of the signature transform, but refer, for an exhaustive and regularly updated overview of applications, to the website DataSig which centralizes information, papers and upcoming events centered on the signature transform.

Yang, L. Jin, and M. Liu 2016 achieve state-of-the-art writer recognition on Chinese and English handwritten text by using the signature transform. They first enhance a database of online collected handwritten characters by dropping some parts, then compute signatures and finally classify by using a deep convolutionnal neural network.

Fermanian 2019 applies the signature to a classification problem for sound (UrbanSound dataset, Salamon, Jacoby, and Bello 2014), simple hand drawn objects (Quick!Draw dataset) and motion sensor data (Motionsens dataset, Malekzadeh et al. 2018). This article also addresses the highly important question of signature embedding, which is not discussed in this thesis. Paths are first embedded (the article features comparisons between classification performances as a function of embedding choice); signatures are then computed and used for classification with different algorithms (Neural Networks, XG-Boost, Random Forest, Nearest Neighboors). On the MotionSens dataset, the author for instance achieves 100 percent accuracy with XGBoost classification.

A.B. Kormilitzin et al. 2016 use the signature transform on medical data with very small sample size ($N = 28$) to predict bipolar disorder. Gyurkó et al. 2013 use the signature to classify financial

data streams.

Even if signatures have been mainly used for classification, some other applications exist. Fermanian 2020 extends signature usage to a linear regression framework, providing theoretical results and applications to weather prediction using Canadian weather data. We do not know of any work combining GANs and Signatures except from Ni et al. 2020, which was published during our work on the subject.

Recall the Kantorovich dual formulation of the optimal transport problem, which we described in Problem (14): we want to solve the optimization problem

$$\sup_{\mathrm{Lip}(f)\leq 1} \mathbb{E}_{X\sim\mu}\left[f(X)\right] - \mathbb{E}_{Y\sim\eta}\left[f(Y)\right]. \tag{32}$$

By reducing the class of Lipschitz functions to the class of linear functions of norm less or equal to 1 of the signature, i.e. by solving the problem

$$\sup_{L \text{ linear}, |L|\leq 1} L\left[\mathbb{E}_\mu S_M(X) - \mathbb{E}_\nu S_M(X)\right], \tag{33}$$

where the X's are paths in $\mathbb{R}^d$, they achieve computation of a closed form of the Wasserstein distance, and thus are able to train a WGAN without training the discriminator (since the explicit formulation of W_1 is known in their framework). Indeed, the solution to problem (33) is

$$\mathrm{Sig}-W_1^{(M)} := \left\|\mathbb{E}_\mu S_M(X) - \mathbb{E}_\nu S_M(X)\right\|_{\ell^2}. \tag{34}$$

The WGAN empirical problem the network tries to solve thus collapses to

$$\inf_{\theta\in\Theta} \frac{1}{n}\left\|\sum_{i=1}^n S_M(X) - \sum_{i=1}^n S_M\left(G_\theta(Z_i)\right)\right\|_{\ell^2}, \tag{35}$$

where the $(Z_i)_i$ are a gaussian sample. This trick tremendously simplifies the WGAN problem (16), since the discriminator is perfectly trained from the beginning. Note that the network does not generate signatures: it uses the signatures of generated paths to approximate the Wasserstein distance between the distributions of true and generated paths.

The authors use this framework for conditional time series generation. Starting from an initial point in time, the network uses the past of the series and some Gaussian noise to generate one further step in time, thus moving forward step by step by using previously generated values. The generated samples are of high quality and outperform other time series generation methods based on GANs and WGANs. However, we believe this method suffers from the high complexity of

the training algorithm and from the lack of analytical tractability of the approximation made in Equation (33).

3 Generating Paths Through GANs and Signatures

3.1 Inverting the Signature Through the Insertion Algorithm

Hambly and T. Lyons 2010 show that the signature characterizes its underlying path under certain conditions; namely, a sufficient condition is that the path has a monotonic coordinate (also see Appendix B.2). Necessary conditions can be found in this article. The problem of inverting a signature, i.e. transforming a given signature into a path (while knowing its truncation order and the dimension of the original path), which naturally follows from this property, has been recently tackled by different articles.

3.1.1 An Overview of Existing Methods

Chang, Duffield, et al. 2017 provide a probabilistic approach to this problem, using large deviations and building upon a probabilistic interpretation of the signature of a path. However, their inversion technique only works for completly monotonic paths, i.e. paths that are monotonic in every direction at every time. This makes it unpractical for applications to real data.

Kidger et al. 2019 use a neural network to approximate the underlying path of a given signature $S(X)$. For a given signature $S(X)$ of a path $X : [0,1] \to \mathbb{R}^d$ truncated at order n, the authors solve the minimization problem

$$\underset{Y:[0,1]\to\mathbb{R}^d}{\arg\min} \left\| S(X) - S(Y) \right\|_2$$

by purely computational means. The problem is not solved analytically: the loss function is simply minimized by gradient descent enabled by backpropagation (which is supported by the Python iisignature package) through a dense neural network. Results are shown in Figure 9. Even though this technique is very effective (signature are inverted almost perfectly), it is extremely costly and non-generic: for every signature one wishes to invert, a full neural network needs to be trained. For instance, inverting the digit 3 shown in 9 took 113 seconds (as shown in the GitHub repository that completes the article). Furthermore, it suffers from an absence of analytical tractability and theoretical guarantees.

We focused on the technique introduced by Chang and T. Lyons 2019, which provides a generic method to invert any given signature, provided that one knows the order at which it was taken and the dimension of the underlying path.

The following subsection describes the inversion algorithm in detail. The general idea of the algorithm is the following. Signatures of order n and $n+1$ are linked through the insertion of the

Figure 9: Approximation of the underlying path of PenDigits signatures trucated at order 12 through a feedforward neural network with ReLu activation trained with backpropagation. True digits are continuous lines, dotted line is the underlying path as reconstructed through signature inversion. The network was trained during 5000 epoches. Source: Kidger et al. 2019.

derivatives of the path into a tensor product that is then integrated. To retrieve the derivate of the path, one can solve a minimization problem that amounts to find a vector in $\mathbb{R}^d$ that will, once inserted into the tensor product of the signature of order n, minimize the distance between this modified signature and the signature at order $n+1$. For truncated signatures, this problem can be written as a constrained minimization problem in finite dimension and can be solved using Lagrande multipliers and Singular Value Decomposition (SVD). Solving this problem allows to retrieve the derivatives of the path on different intervals and to reconstruct the path by integration.

3.1.2 The Insertion Operator

Consider a piece wise linear path $\gamma : [0,1] \to \mathbb{R}^d$ with bounded variations. Define the normalized signature at order m as

$$\bar{\mathbf{S}}^m(\gamma) := m!\,\mathbf{S}^m.$$

We now define the insertion map $I_{p,n} : \mathbb{R}^d \to \mathbb{R}^{d^{n+1}}$. It inserts a vector x into the normalized signature at the spot p, i.e.

$$I_{p,n}(x) = \int_{0<u_1<\cdots<u_n<1} \gamma'(u_1) \otimes \cdots \otimes \gamma'(u_{p-1}) \otimes x \otimes \gamma'(u_p) \otimes \cdots \otimes \gamma'(u_n) du_1 \ldots du_n \in \mathbb{R}^{d^{n+1}},$$

where $\otimes$ denotes the usual Kronecker product between vectors.

Proposition 7. $I_{p,n}$ is linear and Lipschitz-continuous for any norm l^p ($p > 0$) on $\mathbb{R}^{d^m}$.

If inserted at the right spot in the signature of order n, one can expect that the obtained tensor will approximate the signature of order $n+1$.
In fact, the following theorem asserts that the insertion application with parameters n, p, applied to the derivative of the path a a given point, is a good approximation of the signature of order $n+1$. This theorem is crucial as it guarantees that we will be able to retrieve the derivative of the path by finding the vector that minimizes the distance between the insertion operator and the

signature at a order $n+1$.

Definition 12. We say a norm is *proper* if

1. $\left\|\sigma(v)\right\| = \|v\|$ for all $v \in \mathbb{R}^d$ and for every permutation σ of the coordinates of v,
2. For all $n, m \geq 1, \|v \otimes w\| = \|v\|\|w\|$ for all $v, w \in \mathbb{R}^d$.

Theorem 6. Let $\|.\|$ be a proper tensor norm. Assume $\gamma : [0,1] \to \mathbb{R}^d$ is not tree-like, continuous, of bounded variations and its derivative defined almost everywhere and of norm 1 at every time. If γ is linear on $]s,t[\subset [0,1]$, and for $\theta \in]s,t[$, and $p = \lfloor \theta(n+2) \rfloor$, then

$$\left\| I_{p,n}(\gamma'(\theta)) - \bar{S}_{n+1} \right\| \to 0 \quad \text{as} \quad n \to \infty.$$

3.1.3 The Minimization Problem

Using Theorem 6, one can derive the minimization problem

$$\arg\min_{\|x\|_2=1} \left\| I_{p,n}(x) - \bar{\mathbf{S}}^{n+1} \right\|_2 \tag{36}$$

to approximate $\gamma'(\theta)$. The constraint ensures that the path one recovers is parametrized at unit speed, which is a requirement for 6 to hold. Note that this constraint is not convex, and one thus cannot solve this problem by standard gradient descent. If the path is not parametrized at unit speed, the authors show that one can equivalently solve the minimization problem

$$\arg\min_{\|x\|_2=1} \left\| L I_{p,n}(x) - \bar{\mathbf{S}}^{n+1} \right\|_2, \tag{37}$$

where L is the length of the underlying path. It is conjectured by Chang, T. Lyons, and Ni 2018 that the length of the path can be approximated by $n!\|\mathbf{S}^n\|^{1/n}$. The implemented algorithm relies on this conjecture (and provides good inversion results, as will be shown latter on).

Proposition 8. Problem (37) has at least one solution.

Proof. Since $I_{p,n}$ is Lipschitz-continuous, one has for every x, y in $\mathbb{R}^d$

$$\left| \left\| L I_{p,n}(x) - \bar{\mathbf{S}}^{n+1} \right\| - \left\| L I_{p,n}(y) - \bar{\mathbf{S}}^{n+1} \right\| \right| \leq \left\| L I_{p,n}(x) - L I_{p,n}(y) \right\| \leq K \|x - y\|.$$

Thus the objectif function of (37) is continuous. The unit sphere of $\mathbb{R}^d$ is closed and bounded, and the problem has at least one solution. □

3.1.4 An Algorithm for Signature Inversion

The inversion algorithm builds upon the solution to (37) the authors derive by using Lagrange multipliers [3].

First, one can let A denote the matrix representing the linear application $I_{p,n}$ in the canonical basis of $\mathbb{R}^{d^n}$. To find an explicit solution to (37), one first needs the following Lemma.

Lemma. All the singular values of A are identical and equal to $\left\|\bar{\mathbf{S}}^n\right\|_2$.

Proposition 9. Problem (37) has a unique solution.

We give two proofs of this statement. The first one is the one presented by Chang and T. Lyons 2019 in the original paper. The second one was found by Adeline Fermanian and greatly simplifies the inversion of the signature, both from a theoretical and from a computational point of view.

Proof. Let $b := \bar{\mathbf{S}}^{n+1}$, where we indentify $\bar{\mathbf{S}}^{n+1}$ with a vector of $\mathbb{R}^{d^{n+1}}$. By applying singular value decomposition to matrix A (see Appendix B.4), one can write $A = U\Sigma V^T$, where U is orthogonal, Σ is diagonal and V is orthogonal. Since

$$\|Ax - b\|_2 = \left\|\Sigma V^T x - U^T b\right\|_2 = \left\| \begin{pmatrix} \lambda q_1 - y_1 \\ \vdots \\ \lambda q_d - y_d \\ -y_{d+1} \\ \vdots \\ -y_{d^{n+1}} \end{pmatrix} \right\|_2,$$

where $\lambda := \left\|\bar{\mathbf{S}}^n\right\|_2$, $V^T x = \begin{bmatrix} q_1 \\ \vdots \\ q_d \end{bmatrix}$ and $U^T b = \begin{bmatrix} y_1 \\ \vdots \\ y_{d^{n+1}} \end{bmatrix}$. The problem 37 is thus equivalent to

$$\min_{q \in \mathbb{R}^d} \sum_{i=1}^{d} (\lambda q_i - y_i)^2 \text{ subject to } \sum_{i=1}^{d} q_i^2 = 1, \tag{38}$$

since $\left\|V^T x\right\|_2 = \|x\|_2$. Using Lagrange multipliers, one can solve this problem and get the

[3] This algorithm has been jointly designed, coded and tested with Adeline Fermanian, whom I shall thank once again for her precious help. We both contributed equally to these tasks.

solution

$$x^* = \frac{1}{\sum_{i=1}^{d} y_i^2} V V^T \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix}.$$

□

Proof. Problem (37) can be seen as a constrained version of the least-square problem, whose solution is well known to be

$$x^* = \left(A^T A\right)^{-1} A^T b.$$

Projecting this solution onto the sphere $\left\{x \in \mathbb{R}^d \,|\, \|x\| = 1\right\}$, one gets the solution

$$x^* = \frac{\left(A^T A\right)^{-1} A^T b}{\left\|\left(A^T A\right)^{-1} A^T b\right\|}. \tag{39}$$

However, since the matrix $A^T A$ is diagonal, this computation drastically simplifies to

$$x^* = \frac{A^T b}{\left\|A^T b\right\|}. \tag{40}$$

□

In the first stages of our work, signature inversion was implemented through the steps of the first proof in algorithm 3. The algorithm solves the optimization problem for every interval $\left[\frac{p}{n+2}, \frac{p+1}{n+2}\right[$, for $p \in \{0, \dots, n+2\}$, thus recovering the derivative on every interval, and finally integrates the vectors to recreate the path.

However, this algorithm has an important computational bottleneck: the SVD decomposition is quite costly and slows the algorithm down. The second proof allows for a much less costly inversion through a straightforward algorithm that is not described here: one simply creates the matrices A and b and computes the solution using any linear algebra library.

Note that this inversion algorithm does not recover the exact path for two reasons. First, it only approximates the derivatives. Secondly, since the signature is invariant by translation, the starting point of the path is lost when transforming into signature and cannot be recovered; furthermore, the recovered path is always parametrized at unit speed, and will thus not have the same scale as the original path.

Using these algorithms allows for high quality path inversion. One can see some examples in Figure 10, which are commented in Section 4.1.

Algorithm 3 Inversion of a signature (the complicated way)

Inputs: S**: full signature to invert;** n**: truncation order of** S**;** d**: dimension of the underlying path;** a**: starting point of the reconstructed path.**
Get $\mathbf{S}^n$, the signature terms of order n.
$L \leftarrow n!||\mathbf{S}^n||_2^{1/n}$ ▷ Approximate path length
for $p \in \{1, \dots, n+1\}$ **do**
    for $i \in \{1, \dots, d\}$ **do** $\mathbf{A}[d^i : d^{i+1}] \leftarrow I_{p,n-1}(e_i)$
    $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V^T} \leftarrow \text{SVD}(L\mathbf{A})$ ▷ Singular value decomposition
    $\mathbf{y} \leftarrow U^T \mathbf{S}^n[1:d]$
    $\mathbf{x}^*_\mathbf{p} \leftarrow \mathbf{V^T} \frac{1}{||\mathbf{y}||_2} \mathbf{y}$
    Integrate $\mathbf{x}^*_\mathbf{p}$ on $\left[\frac{p}{n+2}, \frac{p+1}{n+2}\right[$
Reconstruct path by concatenation, starting from a**.**

3.1.5 Limits of the Insertion Algorithm

Even though algorithm 3 provides very effective inversion, it is limited by the underlying dependence between inversion precision and signature order. Indeed, when given a signature of order n, the algorithm reconstructs the path through derivative approximation on $n+1$ intervals. Thus, if one wants to reconstruct a Figure with high precision,i.e. by approximating the path's derivative on many intervals, one needs to use large signatures, which rapidly slows the algorithm down since the size of the matrix A grows exponentially with the signature order (recall A is of size $d^{n+1} \times d$).

4 Experimental Results

This section collects four different numerical experiments :

1. The first experiment simply consists in the inversion of some true signatures, including Pendigits but also some geometrical figures.

2. In the second experiment, a WGAN is trained to learn a complex 2D density. This experiment can be seen as a toy example to understand how a WGAN works and which parameters significantly affect the generation results. It has no connections with signatures, but has been a way for us to familiarize ourselves with deep learning, the PyTorch library and WGANs.

3. The third experiment dives into the experimental core idea of our thesis: generating artifical signatures with a WGAN, and then inverting them to obtain artificial time series. To tackle this difficult problem, we first start by trying our method on a simple toy example: we train a WGAN to generate fake signatures of straight lines, which combine the advantages of being simple objects and having a particular structure highlighted in Example 3.

4. Finally, we extend this procedure to the signatures of data coming from the PenDigits dataset (created by Alimoglu and Alpaydin 1996) and design some metrics to asses the generating performance of our WGAN. Experiments are designed in order to understand which parameters play an important role and significantly affect the generating performance, and which others are less important.

4.1 Inverting True Signatures Through the Insertion Algorithm

The insertion algorithm, despite its weaknesses described in Section 3.1.5, performs well for inverting signatures. It was tested on PenDigits and elementary geometrical Figures. One clear weakness one can observe when inverting the signature of a sinusoidal curve is its inability to reproduce small curvatures. Examples can be seen in the notebook provided on the GitHub repository; some are displayed in Figure 10. One can observe that the signature captures some deep information about the structure of the path, which is reconstituted by the insertion algorithm: looking at the inverted 8, one for instance sees that the algorithm adds points that are not present in the original digit but are fully coherent with the structure.

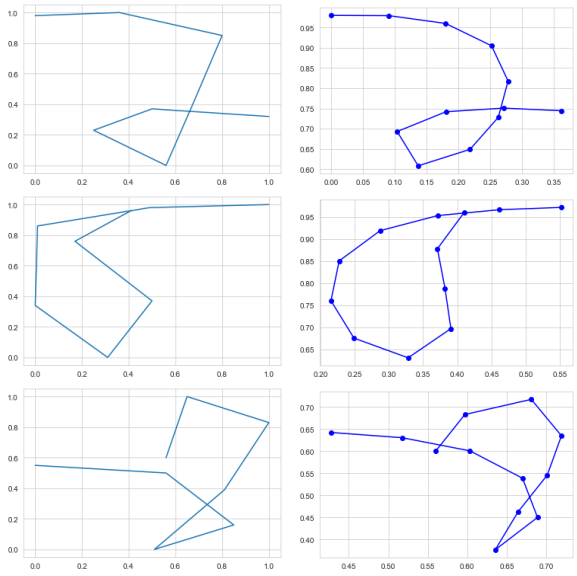


Figure 10: True (light blue) and inverted (dark blue) PenDigit.

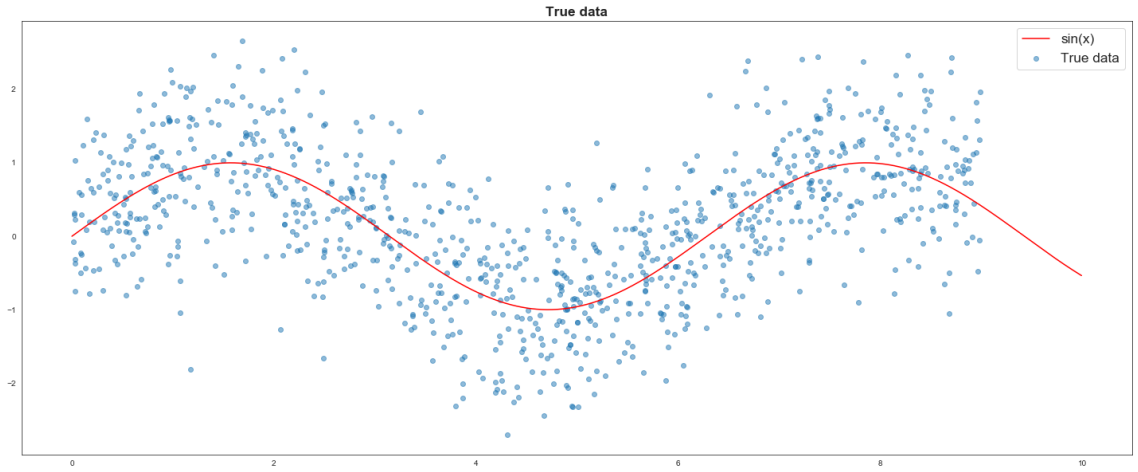


Figure 11: 1000 samples from the true data (in blue) from Experiment 2. The red curve is the function $x \mapsto \sin(x)$.

4.2 Learning a 2D Density with a WGAN

4.2.1 Experimental Setup

The goal of this experiment is to learn a 2D density with a WGAN to get used to WGANs, PyTorch and deep learning. No signatures are involved in this small experiment.

The WGAN tries to reconstruct the 2D density of a couple of random variables (X, Y), where $X \sim \mathcal{U}[0, 9]$, and $Y \sim \sin(X) + \varepsilon$. ε is a Gaussian noise, of variance σ and mean 0. True data is displayed in Figure 11. After training, the WGAN should be able to draw samples (i.e. points in the plan) from this distribution that are indistinguishable from the true points.

4.2.2 Some Results

As the experiment practitioner might surely know, an major problem when training WGANs (and really *any* deep learning model) is the sheer abundances of hyper parameters one might tune, making fine tuning an excruciating and difficult task. This is also a good reason, when trying to use WGANs, to start with very simple settings, that allow to rule out some parameter configurations that lead to aberrant generation results.

A first question that is natural to investigate is to check how much the training of the model influences the results. The WGAN was trained for 1000, 2000, 5000, 10 000 and 20 000 iterations. Note that as explained by Arjovsky, Chintala, and Bottou 2017, a WGAN is theoretically able to learn indefinitely, even if the generator is perfectly trained. For this investigation, the WGAN was trained with input noise size 3, gradient penalty parameter 1, 4 hidden layers of size 50 and Adam optimizer with parameters set as in Gulrajani et al. 2017 ($\beta_1 = 0.5, \beta_2 = 0.999, \alpha = 0.0001$). While we did not see any significant improvement of the WGAN between 1000 and 2000 observations, the generation process gradually starts to get better with 5000 iterations. 10 000 and 20 000 iterations both allow for highly resembling samples. Figure 12 shows some samples drawn from the generator after different amounts of training: one can see that the generated points are more and more diverse and fit the true density increasingly well.

Another parameter that seems of high importance at first glance is the size of what is usually the *latent space*: it is the dimension of the vector space $\mathbb{R}^k$ in which the noise - the initial data fed into the WGAN - lives. This noise, as said earlier, is usually drawn from a Gaussian distributions, even if uniform and more exotic distributions are sometimes used. The WGAN was trained for 10 000 iterations and all other parameters identical, but for the input noise size, which was set to 1,3,5,10 and 50. The input noise size has an important influence on the data generation quality. While a noise size set to 1 produces bad samples, results get very close to the original data for noise size 10 and 50 (one can see that the empirical CDF almost perfectly approximates the true CDF).

What did we learn from experimenting with this simple WGAN ?

- Trying to tune the parameters of the Adam algorithm, ore trying to use other optimizers than Adam, is most of the time unsuccessful. When using Gradient Penalty (GP), one can stick with the parameters given by Gulrajani et al. 2017.
- The size of the latent space plays an important role, but increasing it too much is unnecessary. A latent space of dimension 1 produces bad generation results, but we seen little difference between samples generated with noise size 10 and 50.
- While we did not extensively investigate the effects of the architecture of the generator and

the discriminator on generation quality, we found that small, simple feed-forward networks are able to generate high quality samples.

- As already suggested by some numerical experiments performed by Gérard Biau, Sangnier, and Tanielian 2020, it is best to keep the generator and the discriminator symmetric (same size, same activation functions). Some small tests performed with asymmetric networks showed deteriorated generating performance.

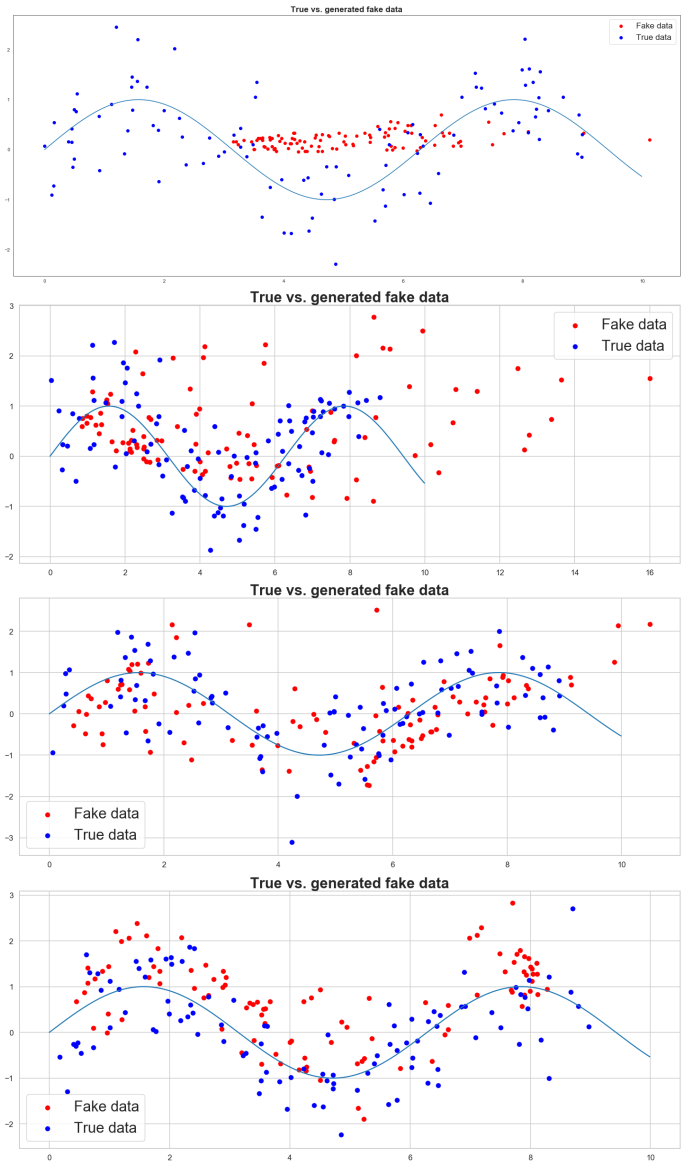


Figure 12: True (blue) and fake (red) datapoints generated with a WGAN-GP trained for 1 000, 5 000, 10 000 and 20 000 iterations (from top to bottom).

4.3 Generating Fake Straight Line Signatures

4.3.1 General Setup

In this experiment, we try to generalize our WGAN framework to generate signatures of straight lines in three dimensions with random slopes using a WGAN with gradient penalty. The true data consists in linear paths of length 5, in three dimensions, discretized over 10 points. The true samples thus looks like

$$\begin{bmatrix} x_{1,1} & \dots & x_{1,10} \\ x_{2,1} & \dots & x_{2,10} \\ x_{3,1} & \dots & x_{3,10} \end{bmatrix}$$

and are transformed into signatures truncated at order 4 that thus have the shape

$$\left(S^{(1)}, \dots, S^{(3,3,3,3)}\right) \in \mathbb{R}^{120}.$$

The slope, in every dimension, is drawn at random from a uniform distribution on $[0, 1]$. Signatures are then computed for these paths at truncation order 4. For every training iteration of the WGAN, new data is generated, consisting in 100 signatures of true paths. Both the generator and the discriminator have 8 linear layers with Relu activation functions. Every layer has size 240, and input noise has size 10. After training the WGAN for 6000 iterations, we generate some signatures and invert them through the insertion algorithm.

4.3.2 Extrapolating Signatures

When generating signatures, it is hard to know if one has actually succeeded: how can one distinguish a signature from any other vector, and how can one tell how close the generated signature is to an actual signature ? This fundamental problem is specific to our case: when generating images, sound or text with a WGAN, one can rely on visual inspection of the generated samples to asses the network's quality. When generating abstract objects like signatures, such an inspection is much more limited - one might detect if the network is completely missing its target, but will not grasp small differences that become decisive when inverting the signature.

A special case in which one can easily know what the generated signature should look like is the case of signatures of linear paths. As one can see in example 3, the signature of a three dimensional linear path is fully determined by its three first coefficients. Indeed, if we consider a three

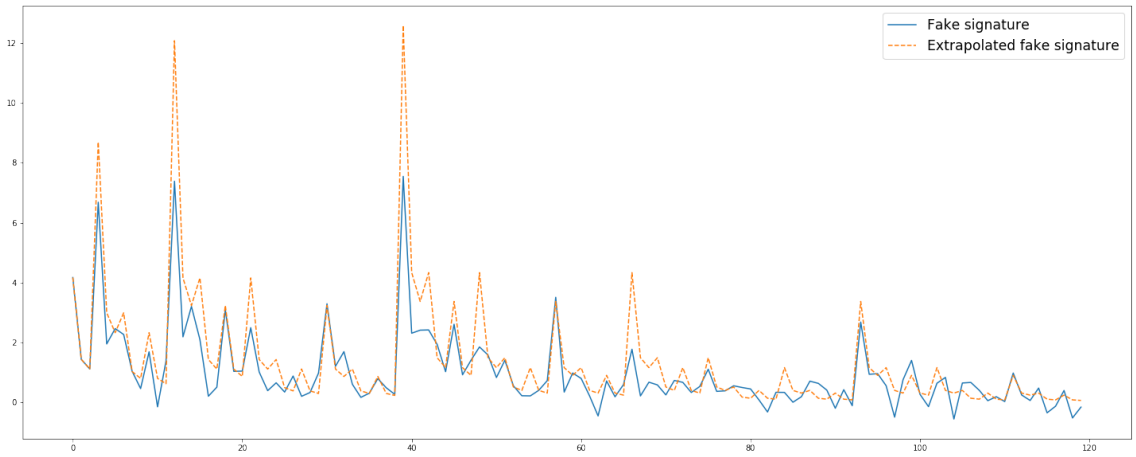


Figure 13: Plot of a full fake signature of a linear path generated by a WGAN and of the signature obtained by extrapolating its first coefficients.

dimensional linear path $X : [0, 1] \to \mathbb{R}^3$, meaning that

$$X_t = X_0 + t \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

the signature of the path can be immediately computed : for any word $K = (i_1, \dots, i_n)$, one has

$$S^K(X) = \frac{a_{i_1} \cdots a_{i_n}}{n!}. \tag{41}$$

Thus, one can check for internal coherence of generated signatures by checking if the last coefficients are coherent with the three first: if we suppose that the generated vector is indeed a signature, then the three first coefficients should correspond to a_1, a_2 and a_3 and all the following coefficients should be in line with these coefficients in the sens of Equality (41). We call this verification extrapolation of the signature. Note that this is generally impossible for non linear paths. Figure 13 displays a comparison between a fake signature and its extrapolated counterpart, generated by the aforementioned WGAN. The WGAN misses some of the spikes of the signature, but globally generates a coherent sample.

Going further, we compute the L_2-norm of the difference between the generated signature and the extrapolated version of a sample of paths for every training iteration. We eventually average this distance over multiple signatures. The computed extrapolation metric is thus equal to

$$\frac{1}{K} \sum_{i=1}^{K} \left\| S_{N,i} - \tilde{S}_{N,i} \right\|_2, \tag{42}$$

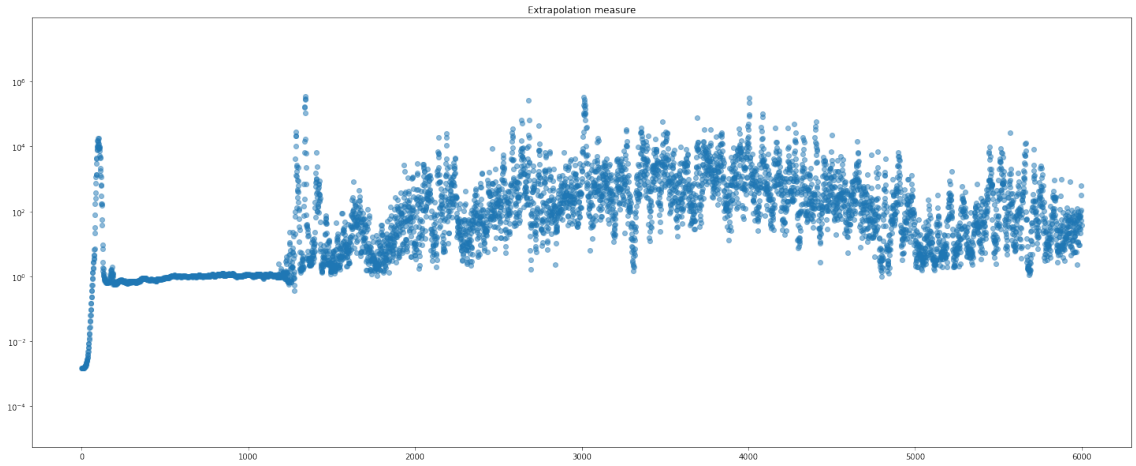


Figure 14: Plot of the extrapolation measure described in equation (42) for $K = 100$.

where $\left(S_{N,i}\right)_i$ are generated full signatures (truncated at order N), and $\left(\tilde{S}_{N,i}\right)_i$ are the signatures extrapolated from the first coefficients. Figure 14 displays the evolution of this quantity during the training of WGAN. The obtained plot is hard to interpret: it does not show clear convergence of the quantity towards a minimum, suggesting (1) that it does not capture well the increasing performance of an improving WGAN (2) that the WGAN is not trained enough for us to see a clear convergence of this metric. We favour the second interpretation of this graph: WGANs notoriously require great amounts of training, which we were not able to do due to computational limitations.

4.3.3 Results

One sees (Figure 22 in the Appendix) that a problem of our WGAN is that it fails to generate the more spiking signatures; however, zooming in on the signatures that have a smaller amplitude shows diverse generation at this scale. We see no evidence of mode collapse.

However, once inverted, these signatures do not generate perfect straight lines. This problem is caused by bad generation performance of the WGAN: tests concluded that when given the true signature of a straight line, the insertion algorithm is perfectly capable of reconstructing a straight line. Some plots of inverted lines can be seen in Figure 21. Consistently with results of the previous experiment, we found that the input noise size (the size of the latent space of the generator) improves generation quality up to a certain point (we set it to be 10). We did not see any improvement in performance beyond this noise size threshold.

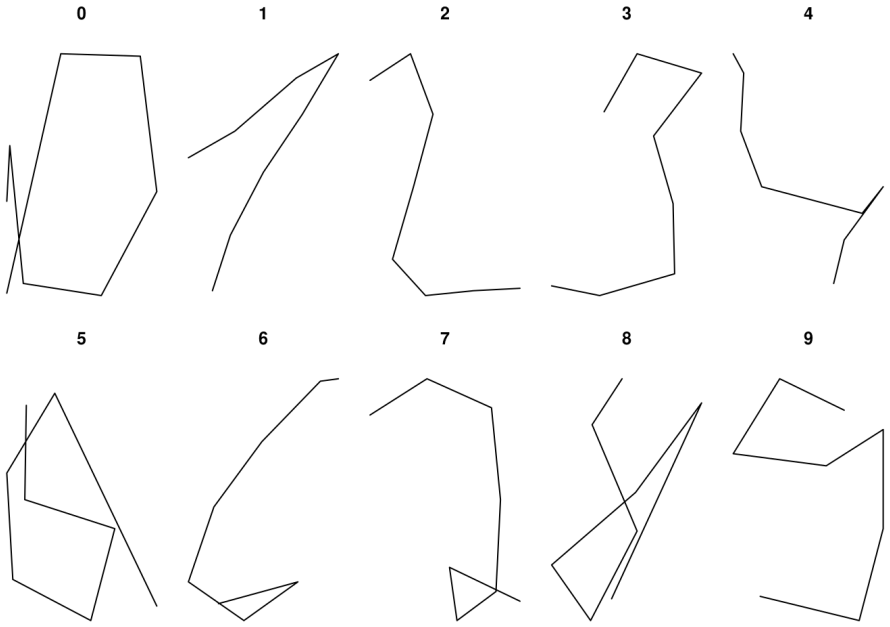


Figure 15: Some digits from the Pendigits dataset. Source: blogpost by F.X. Jallois.

4.4 Generating Fake Pendigits Signatures

4.4.1 The PenDigits dataset

The Pendigits dataset was created by Alimoglu and Alpaydin 1996. It is a multiclass classification set containing handwritten digits with 16 attributs, split into 10 different classes (the integers between 0 and 9). The set was created by asking 44 writers to write 250 random digits with a stylet on a 500×500 pixel screen. Every integer consists in a time series, i.e. a vector of position (x_i, y_i) in $\mathbb{R}^2$ that is indexed by eight points time.

Some digits from the dataset can be seen in Figure 15. One should keep in mind that they usually have a similar level of resemblance with actual digits throughout the whole dataset.

4.4.2 Evaluating the Performance of a Signature Generating WGAN

Evaluating performances of WGANs is a notoriously hard task. Metrics do exist, but are often designed for one specific kind of data (Shmelkov, Schmid, and Alahari 2018), and as said early, evaluating the performance of a signature generating WGAN is particularly hard. To evaluate the performances of our WGANs, since we cannot rely on signature extrapolation anymore, we use four different techniques.

First, as it is often done for evaluating WGANs, one can check for quality of generation by visually inspecting the generated samples. In our case, we can inspect the generated signatures to check if they approximately match the target signatures and are not completely off.

Secondly, we check for convergence in mean of the generated signatures towards the true signatures. More precisely, letting $\tilde{S}_N = (\tilde{S^{(1)}}, \dots, \tilde{S}^{(d,\dots,d)})$ be a generated signature, and $S_N = (S^{(1)} \dots, S^{(d,\dots,d)})$ be a true signature , consider a sample $(\tilde{S}_{N,k})_k$ of M generated signatures and a sample $(S_{N,k})_k$ of M signatures of real Pendigits for a given class of integers. The quantity we call the mean metric is equal

$$\left\| \frac{1}{M} \sum_{k=1}^{M} \tilde{S}_{N,k} - \frac{1}{M} \sum_{k=1}^{M} S_{N,k} \right\|_2 ,$$

where the $(\tilde{S}_{N,k})_k$'s and $(S_{N,k})$'s are sampled from the generator and the true data respectively. This quantity measures the convergence in norm L^1 of the random variable associated with the generator (the output of this network) towards the true distribution of pendigits.

Thirdly, using the shuffle identity (Proposition 5), we derive a metric to evaluate the coherence of generated signatures. Recall that the shuffle identity links two signature coefficients with the signature coefficients of the words of their shuffle product: let I, J be two words from the alphabet $\{1, \dots, d\}$ of length n and m; one then has

$$S^I(X)S^J(X) = \sum_{K \in I \sqcup J} S^K(X) \in \mathbb{R}. \tag{43}$$

Formally, the metric we use is equal to

$$\frac{1}{K} \sum_{I_k, J_k} \left(S^{I_k}(X)S^{J_k}(X) - \sum_{P \in I_k \sqcup J_k} S^P(X) \right)^2 , \tag{44}$$

where $(I_k, J_k)_{1 \leq k \leq K}$ are multi-indices sampled randomly from all possible compatible multi-indices that are used to compute the signature of the path. Compatible is here to be understood in the following sens: the shuffle product of two words I, J of length m and n creates words of length $n + m$; thus, to be able to compare the signature coefficients of words of the shuffle product with the product $S^I(X)S^J(X)$, $n + m$ must be lesser or equal to the truncation order of the signature. This metric is then eventually averaged over multiple signatures. It evaluates the coherence of generated signatures, as for real signatures, it is theoretically equal to zero (and very close to 0 in practice due to numerical approximations).

Finally, we introduce a third party classifier that tries to classify the generated signatures after being trained on true signatures. Fermanian 2019 uses random forests, nearest neighbors, neural networks and XGBoost to classify signatures and to investigate how the classifier's performances

are affected by signature order and specific embeddings. Building upon this idea, we evaluate the WGAN's performance with following algorithms:

1. **Logistic regression.** A standard way to classify data is to use logistic regression. We implement this method with Lasso regularization. Data is centered using SKlearn StandartScaler before training. The minimization problem is solved using L-BFGS. Maximum number of iterations is set to 1000.

2. **Random forest.** This classification algorithm was initially proposed by [Breiman 2001]. Using SKlearn, we implement random forest classification with 50 estimators and maximum depth 10.

3. **Nearest neighbors.** This popular non-linear classification method is implemented with 5 considered neighbors and using Minkowski distance.

4. **Neural network.** We build a neural network to classify generated signatures. The architecture of our network comes directly from [Fermanian 2019]: it consists of a dropout layer with rate 0.5, followed by a layer with 64 neurons, a linear activation and a softmax output. The network is trained with SGD (with learning rate set to 1) through 200 epochs with mini-batch size 64. We choose cross entropy as loss function.

For every classification algorithm, training is done on 80 percent of the PenDigits dataset (5995 observations): the algorithms learn to classify signatures of true pendigits. We evaluate the algorithm on the 20 remaining percent (1499 observations) as a reference point. Following this training, we compute the algorithm's accuracy on 1500 samples of the WGAN trained on one single digit: the algorithms are asked to classify the signatures we generate with the generator, which are all signature of one digit. Thus the algorithms should classify all generated signatures in one single class (if trained perfectly).

High accuracy of a well trained model on generated data suggests that the WGAN has learned to generate features that resemble those of true data. Using different classification techniques allows for a more precise picture of the WGAN's perfomance.

A caveat we do not adress is sample redundance: the WGAN might generate similar samples that artificially increase accuracy. For instance, if the WGAN was to generate only one identical copy of an observation of the true data, its accuracy would be near 100 percents on most algorithms. However, visual inspection of the generated signatures shows that the generated samples are diverse. This is due to the use of a WGAN instead of a GAN, which is known to cause less mode collapse, as explained for instance in Weng 2019.

4.4.3 Experimental Setup and Evaluation

The goal of this numerical experiment is to generate fake Pendigit signatures with a WGAN-GP. Our pipeline works as following:

1. Pendigits are normalized and signatures of a given order calculated using the Signatory library.

2. The WGAN-GP is trained on true Pendigit signatures of one given integer. At each iteration, the WGAN is trained on 100 true signatures.

3. After training, the generator should be able to create signatures of Pendigits. We invert the created signature and inspect the results.

Different settings are tried to investigate the effects of hyperparameters, and to sort out which parameters play an important role and which others are secondary. The settings are evaluated through different metrics and classification algorithms.

4.4.4 Results

Figure 16 shows some Pendigits obtained by our generation process. Our results are numerically documented in the Appendix C.

1. The first parameter of the network we examine is the number of training iterations. The result of our experiments is globally consistent with what we found in earlier experiments: the WGAN's performance increases with training. Notice however that there might be some moments in training where generation performance deteriorates significantly for a significant amount of time: this might be due to the networks escaping a local extrema and transitioning towards another one.

2. The second parameter we take a look upon is the gradient penalty parameter. The results suggest that a too strong gradient penalty parameter penalises the ability of the WGAN to generate samples that resemble the true signatures. Gulrajani et al. 2017, who initially introduced WGANs with gradient penalty, do not give guidelines for tuning this hyperparameter other that checking the quality of generated samples. In our case, classifier performance drops as we increase the gradient penalty parameter, while setting it to 10 (which is in fact the same value as in Gulrajani et al. 2017) seems to yield good results.

3. The last parameter whose effects were investigated is the size of the latent space. Interestingly, this parameter seems to have a non linear effect: over a certain latent space size, the classifiers performance decreases. The optimal noise size seems to be around 10.

These experiments and many others we do not report lead us to formulate following suggestions when training WGANs:

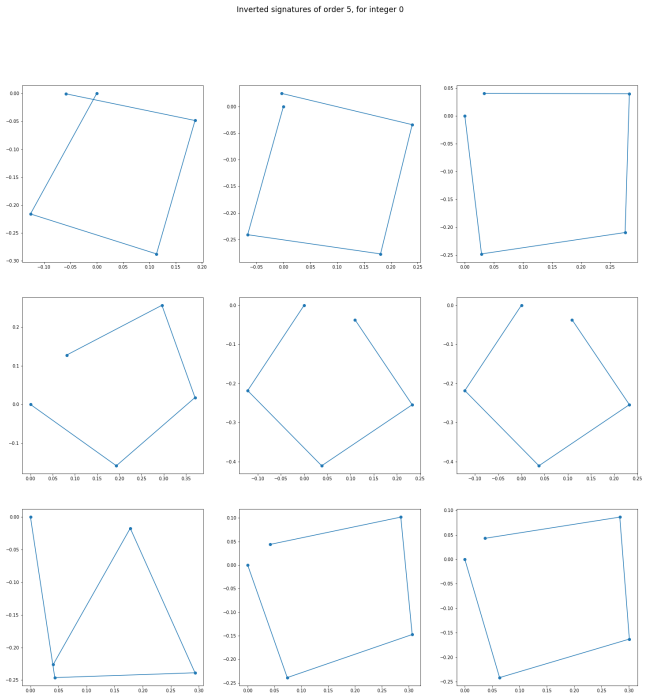
1. Use networks with a sufficient amount of layers. If generation is unsuccessful, try first to increase the number and height of layers of the neural networks. Networks can remain simple when generating signatures, but more sophisticated architectures, including convolutions, should be explored. Activation functions should be kept to be ReLu.

2. Generator and discriminator should be kept symmetric. Alternated training (training the discriminator k times for every generator step) is generally not a solution and should be used only if one notices oscillations of the network's performance metrics.

3. Try to train the network as much as possible. This might require great computational power but significantly improves generation performance.

4. The Adam optimizer works well; one should keep the hyperparameters suggested by Gulrajani et al. 2017.

5. Gradient Penalty greatly improves the performance of a network over clipping and other brutal methods, but the tuning of the hyperparameter λ (that adjusts the importance of the relaxed constraint) seems to little importance to us. One can check if the gradient's norm does indeed converge towards 1 as training goes on to ensure that the gradient penalty is efficient.



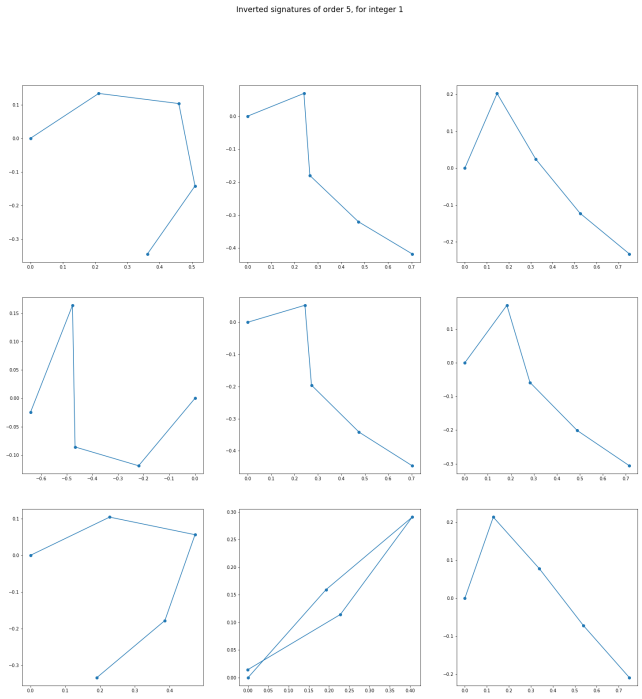


Figure 16: Zeros and ones obtained by generating signatures truncated at order 4 resp. 5 with a WGAN with Gradient Penalty after 20000 training iterations. The network has 4 layers that are two times the size of the signature vector high. Activation functions are ReLu, and optimization is done with Adam.

5 Conclusion

Starting from the observation that data generation is a key process of modern statistical models, we built a pipeline combining a particular object from rough path theory, namely signatures, and a distance coming from optimal transport theory, the Wasserstein distance, to build a signature generating WGAN that tries to create fake time series. Results are modest, but we provide below some ideas to explore.

Further research and developments. The short duration of this internship (3 months) and my relative ignorance of deep learning made it impossible to explore the following ideas, which could become subjects for further research:

1. **Improving performance and computational efficiency of the insertion algorithm**. As mentioned earlier, the insertion algorithm mainly suffers from the direct link between the order of the signature and the number of intervals on which the path's derivative is approximated. A great improvement of our method would be to break this link, making it possible to approximate the path on many intervals while using low order signatures. Another possible next step would simply be to optimize the insertion algorithm, for instance through finding a more efficient way to solve the optimization problem.

2. **Assessing WGAN performance through Wasserstein distance approximations.** Recent research has proposed some methods to approximate Wasserstein distances between two distributions, most famously through Sinkorn's Algorithm (see Cuturi 2013 and Chizat et al. 2020 for instance). Assessing WGAN perfomance is difficult, since the network only approximates the Wasserstein distance, but we cannot know how good this approximation is. A possible improvement on our methods would be to use Sinkhorn's algorithm to estimate the true Wasserstein distance between the true and the generated distribution.

3. **Partial signature generation.** One may notice that the insertion algorithm only uses the last terms of the signature to invert the path. Thus, if one wants to focus only on generating fake time series through insertion, one could train a WGAN to generate only the last terms of the signature.

4. **Assessing WGAN performance through adapting existing metrics.** As explained earlier, different techniques have been designed to evaluate the performances of WGANs, but are however designed for very specific dataypes. Adapting these measures to time series and signatures would be a key step.

5. **Using other Wasserstein distances**. Recent research (for instance H. Liu, Gu, and Samaras 2019) suggests using alternative Wasserstein distances, such as W_2 or W_p, for $p \geq 1$ in WGANs. Deshpande, Z. Zhang, and Schwing 2018 make use of the so-called sliced Wasserstein distance within a generative model. These are settings that one could possibly explore.

6. **More complex network structures.** One of the advantages of generating fake signatures instead of fake paths is the simplicity of the neural network one can use (a FF-NN is well suited for this task). However, using more complex networks such as LSTM networks or convolutional networks might lead to interesting results. Injecting noise at intermediate layers, as suggested by Goodfellow et al. 2014, could also increase generation perfomance.

7. **Theoretical properties of Signature-WGANs**. Theoretical properties of WGANs are an active domain of research and very little is know on the matter, as noted by Gérard Biau, Sangnier, and Tanielian 2020. The theoretical properties of a WGAN using signatures at an input stage or latter on in the network are also to be investigated.

8. **Signatures and time series**. Numerous results, methods and measures for the analysis of time series have been developed in the recent years (see the synthesis by Lütkepohl 2005). A theoretical and empirical exploration of these tools - as, for instance, auto-correlation measures, specific statistical tests and forecasting methods -, combined with signatures could lead to interesting and stimulating results.

What I learned from this research internship. Having a background in pure mathematics and social sciences, this internship was a great occasion to gain a better knowledge of machine learning and related topics.

1. **Improvement of coding skills.** I tremendously improved my coding skills in Python, especially in PyTorch, SKLearn and genereal project construction. I also had the occasion to learn how to use the SSH-servers of the LPSM.

2. **Deep learning.** Before this internship, I had almost no knowledge of deep learning and neural networks. Working on WGANs and GANs was a great opportunity to gain knowledge of these topics.

3. **Signatures, Statistics, Optimal Transport, Rough Path Theory**. I gained a better knowledge of Optimal Transport and statistics, which I was familiar with on a graduate degree level, and discovered rough path theory and signatures, which I did not previously know.

4. **Research in general.** I learned numerous things about working as a researcher in Machine Learning, especially as far a result presentation, scientific writing, communication and work organization are concerned.

References

[1] Fevzi Alimoglu and Ethem Alpaydin. "Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwritten Digit Recognition". In: *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96*. 1996.

[2] Moustafa Alzantot, Supriyo Chakraborty, and Mani Srivastava. "Sensegen: A deep learning architecture for synthetic sensor data generation". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2017, pp. 188–193.

[3] Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks". In: *arXiv preprint arXiv:1701.04862* (2017).

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[5] Gérard Biau, Maxime Sangnier, and Ugo Tanielian. "Some Theoretical Insights into Wasserstein GANs". In: *arXiv preprint arXiv:2006.02682* (2020).

[6] Gérard Biau et al. "Some theoretical properties of GANS". In: *Ann. Statist.* 48.3 (June 2020), pp. 1539–1566. DOI: 10.1214/19-AOS1858. URL: https://doi.org/10.1214/19-AOS1858.

[7] Samuel R Bowman et al. "Generating sentences from a continuous space". In: *arXiv preprint arXiv:1511.06349* (2015).

[8] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[9] Jiawei Chang, Nick Duffield, et al. "Signature inversion for monotone paths". In: *Electronic Communications in Probability* 22 (2017).

[10] Jiawei Chang and Terry Lyons. "Insertion algorithm for inverting the signature of a path". In: *arXiv preprint arXiv:1907.08423* (2019).

[11] Jiawei Chang, Terry Lyons, and Hao Ni. "Super-multiplicativity and a lower bound for the decay of the signature of a path of finite length". In: *Comptes Rendus Mathematique* 356.7 (2018), pp. 720–724.

[12] Jingwen Chen et al. "Image blind denoising with generative adversarial network based noise modeling". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3155–3164.

[13] Kuo-Tsai Chen. "Integration of paths–A faithful representation of paths by noncommutative formal power series". In: *Transactions of the American Mathematical Society* 89.2 (1958), pp. 395–407.

[14] Ilya Chevyrev and Andrey Kormilitzin. "A Primer on the Signature Method in Machine Learning". In: (2016). arXiv: 1603.03788 [stat.ML].

[15] Lenaic Chizat et al. "Faster Wasserstein Distance Estimation with the Sinkhorn Divergence". working paper or preprint. June 2020. URL: https://hal.archives-ouvertes.fr/hal-02867271.

[16] Marco Cuturi. "Sinkhorn distances: Lightspeed computation of optimal transport". In: *Advances in neural information processing systems*. 2013, pp. 2292–2300.

[17] Ishan Deshpande, Ziyu Zhang, and Alexander G. Schwing. "Generative Modeling using the Sliced Wasserstein Distance". In: *CoRR* abs/1803.11188 (2018). arXiv: 1803.11188. URL: http://arxiv.org/abs/1803.11188.

[18] Adeline Fermanian. "Embedding and learning with signatures". In: *arXiv preprint arXiv:1911.13211* (2019).

[19] Adeline Fermanian. "Linear functional regression with truncated signatures". In: *arXiv preprint arXiv:2006.08442* (2020).

[20] Adeline Fermanian. "Signature and statistical learning". 2018.

[21] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[22] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *Advances in neural information processing systems*. 2017, pp. 5767–5777.

[23] Lajos Gergely Gyurkó et al. "Extracting information from the signature of a financial data stream". In: *arXiv preprint arXiv:1307.7244* (2013).

[24] Ben Hambly and Terry Lyons. "Uniqueness for the signature of a path of bounded variation and the reduced path group". In: *Annals of Mathematics* (2010), pp. 109–167.

[25] Shota Haradal, Hideaki Hayashi, and Seiichi Uchida. "Biosignal data augmentation based on generative adversarial networks". In: *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE. 2018, pp. 368–371.

[26] Yanghua Jin et al. "Towards the automatic anime characters creation with generative adversarial networks". In: *arXiv preprint arXiv:1708.05509* (2017).

[27] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. "Feature-based comparison and generation of time series". In: *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 2018, pp. 1–12.

[28] Patrick Kidger et al. "Deep signature transforms". In: *Advances in Neural Information Processing Systems*. 2019, pp. 3105–3115.

[29] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[30] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[31] Franz J Király and Harald Oberhauser. "Kernels for sequentially ordered data". In: *arXiv preprint arXiv:1601.08169* (2016).

[32] Allison Koenecke and Hal Varian. "Synthetic Data Generation for Economists". In: (2020).

[33] A.B. Kormilitzin et al. "Application of the signature method to pattern recognition in the cequel clinical trial". In: *arXiv preprint arXiv:1606.02074* (2016).

[34] Juntao Li et al. "Generating classical chinese poems via conditional variational autoencoder and adversarial training". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 3890–3900.

[35] Yijun Li et al. "Generative face completion". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3911–3919.

[36] Yujia Li, Kevin Swersky, and Rich Zemel. "Generative moment matching networks". In: *International Conference on Machine Learning*. 2015, pp. 1718–1727.

[37] Tengyuan Liang. "On how well generative adversarial networks learn densities: Nonparametric and parametric results". In: *arXiv preprint arXiv:1811.03179* (2018).

[38] Huidong Liu, Xianfeng Gu, and Dimitris Samaras. "Wasserstein GAN with quadratic transport cost". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4832–4841.

[39] Peng Liu et al. *Hidden markov model based handwriting/calligraphy generation*. US Patent 7,983,478. July 2011.

[40] Helmut Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[41] Terry J Lyons, Michael Caruana, and Thierry Lévy. "Differential Equations Driven by Moderately Irregular Signals". In: *Differential Equations Driven by Rough Paths: École d'Été de Probabilités de Saint-Flour XXXIV-2004* (2007), pp. 1–24.

[42] Terry J Lyons, Michael Caruana, and Thierry Lévy. *Differential equations driven by rough paths*. Springer, 2007.

[43] Mohammad Malekzadeh et al. "Protecting sensory data against sensitive inferences". In: *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*. 2018, pp. 1–6.

[44] Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.

[45] Olof Mogren. "C-RNN-GAN: Continuous recurrent neural networks with adversarial training". In: *arXiv preprint arXiv:1611.09904* (2016).

[46] Hao Ni et al. "Conditional Sig-Wasserstein GANs for Time Series Generation". In: *arXiv preprint arXiv:2006.05421* (2020).

[47] Thiemo Pesch et al. "A new Markov-chain-related statistical approach for modelling synthetic wind power time series". In: *New journal of physics* 17.5 (2015), p. 055001.

[48] Gabriel Peyré, Marco Cuturi, et al. "Computational Optimal Transport: With Applications to Data Science". In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607.

[49] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[50] Yong Ren et al. "Conditional generative moment-matching networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2928–2936.

[51] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[52] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A dataset and taxonomy for urban sound research". In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 1041–1044.

[53] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. "How good is my GAN?" In: *CoRR* abs/1807.09499 (2018). arXiv: 1807.09499. URL: http://arxiv.org/abs/1807.09499.

[54] Luca Simonetto. "Generating spiking time series with Generative Adversarial Networks: an application on banking transactions". In: (2018).

[55] Shinnosuke Takamichi, Tomoki Koriyama, and Hiroshi Saruwatari. "Sampling-based speech parameter generation using moment-matching networks". In: *arXiv preprint arXiv:1704.03626* (2017).

[56] Subeesh Vasu, Nimisha Thekke Madam, and AN Rajagopalan. "Analyzing perception-distortion tradeoff using enhanced perceptual super-resolution network". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 0–0.

[57] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media, 2008.

[58] Cédric Villani. *Topics in optimal transportation*. 58. American Mathematical Soc., 2003.

[59] Lilian Weng. "From GAN to WGAN". In: *ArXiv* abs/1904.08994 (2019).

[60] Liyang Xie et al. "Differentially private generative adversarial network". In: *arXiv preprint arXiv:1802.06739* (2018).

[61] Lei Xu and Kalyan Veeramachaneni. "Synthesizing tabular data using generative adversarial networks". In: *arXiv preprint arXiv:1811.11264* (2018).

[62] Xinchen Yan et al. "Attribute2image: Conditional image generation from visual attributes". In: *European Conference on Computer Vision*. Springer. 2016, pp. 776–791.

[63] Weixin Yang, Lianwen Jin, and Manfei Liu. "Deepwriterid: An end-to-end online text-independent writer identification system". In: *IEEE Intelligent Systems* 31.2 (2016), pp. 45–53.

[64] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series generative adversarial networks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 5508–5518.

[65] Lantao Yu et al. "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[66] Chi Zhang et al. "Generative adversarial network for synthetic time series data generation in smart grids". In: *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE. 2018, pp. 1–6.

[67] He Zhang, Vishwanath Sindagi, and Vishal M Patel. "Image de-raining using a conditional generative adversarial network". In: *IEEE transactions on circuits and systems for video technology* (2019).

[68] Pengchuan Zhang et al. "On the discrimination-generalization tradeoff in GANs". In: *arXiv preprint arXiv:1711.02771* (2017).

A The Adam Optimization Algorithm

The Adam[4] algorithm was first designed and described by Kingma and Ba 2014. It falls into the broad category of gradient descent based optimization algorithms, but improves upon existing techniques by combining two features of two subclasses of stochastic gradient descent algorithms. Adam combines the advantages from *adaptative gradient algorithms*, that allow for some learning rate variance between parameters (and is thus highly effective for sparse problems), but also draws from *root mean square propagation* that allows for time varying learning rates. Ruder 2016 provides an excellent overview of existing gradient descent algorithms, and how they relate one to another. Adam is used to solve the stochastic optimization problem

$$\min_{\theta} \mathbb{E}\left[f(\theta)\right], \tag{45}$$

where f is a scalar value function with parameters θ. The stochasticity of f comes from the fact that it might, for instance, be evaluated on mini-batches of data. Let $f_1, \ldots, f_T$ denote its realizations. Intuitively, Adam works by updating parameters through a momentum-based second order estimation, adjusted through exponentially decaying bias correction. It requieres 4 hyperparameters: β_1, β_2 control the decacy rate of the bias correction for first and second moment estimates respectively; α is the learning rate; ε is a small number that prevents division by 0 in the implementation.

Let $g_t := \nabla_\theta f_t(\theta)$. Adam works in three times. It first computes moving averages of the gradient's mean and variance through

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{46}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2. \tag{47}$$

However, these estimates are biased. Thus, in a second time, the algorithm computes unbiased estimates by computing

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{48}$$

$$\hat{v}_t = \frac{m_t}{1-\beta_2^t}. \tag{49}$$

[4] ...which is not an acronym but is "derived from Adaptative Moment Estimation" as the authors write.

It finally updates θ_t through

$$\theta_{t-1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t. \tag{50}$$

Kingma and Ba 2014 find that Adam outperforms numerous optimization algorithms on standart problems, as one can see in Figure 17. The authors also provide a theoretical upper bound on regret.

Proposition 10. Assume that f_t has gradients bounded, for L_2 and L_∞ norm, by G and G_∞ for all $\theta \in \mathbb{R}^d$. Assume furthermore that the distance for any two parameters generated by Adam is bounded in L_2 norm, i.e. $\|\theta_n - \theta_m\|_2 \leq D$ for all n, m. Finally, assume that the hyperparameters satisfy $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$ and let $\alpha_t = \alpha/\sqrt{t}$, $\beta_{1,t} = \beta_1 \lambda^{t-1}$. Adam achieves, for every $t \leq T$, the following bound on regret:

$$R(T) \leq \frac{D^2}{2\alpha\,(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha\,(1+\beta_1)\,G_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} \left\|g_{1:T,i}\right\|_2 + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha\,(1-\beta_1)\,(1-\lambda)^2}.$$

Proof can be found in the original paper.

B Tools

B.1 Lexicographical order

Proposition 11. The set of words of length m from an arbitrary alphabet $\{1, \ldots, d\}$ can be equipped with an order relation, called the lexicographical order.

Let $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_m)$ be two words from the aformentioned alphabet. The lexicographical order is established by comparing successively the components of the two words. To check if $a \prec b$, one thus first checks if $a_1 < b_1$. If these two letters are equal, one moves on to checking if $a_2 < b_2$, and so forth. Let us make two remarks.

1. All words can be compared since the alphabet $\{1, \ldots, d\}$ is equipped with a total order. The only case we may fail to establish if $a \prec b$ or $b \prec a$ is when all successive letter tests fail : but this means that the two words are indeed equal letter by letter.

2. In mathematical terms, $a \prec b$ means that there exists $1 \leq i \leq m$ such that for all $j < i$, $a_j = b_j$ and $a_i < b_i$.

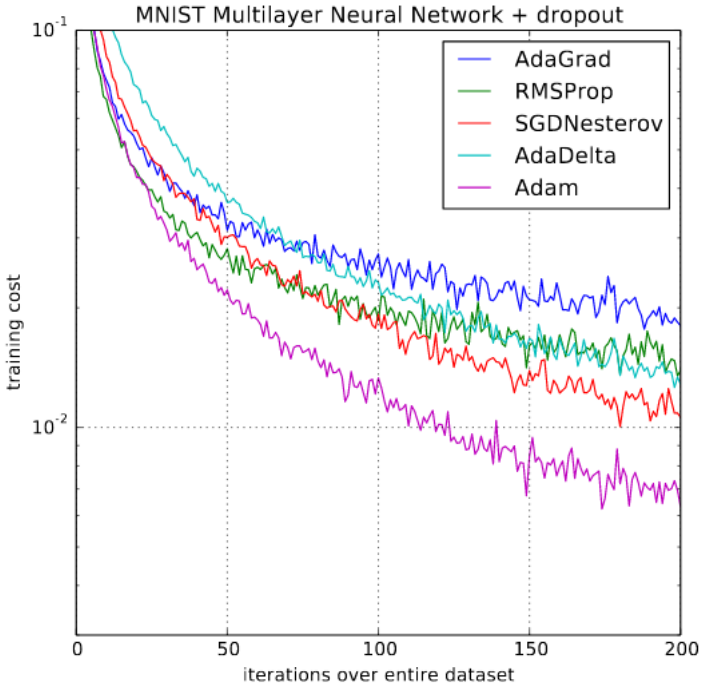


Figure 17: Efficiency of Adam algorithm for training an MNIST classification network, compared to other widely used algorithms. The trained network consists of two hidden layers with 1000 hidden units and ReLu activation functions. It is trained with mini-batch size 128. Source: Kingma and Ba 2014.

B.2 Tree-like paths

This section of the appendix states some mostly technical points that formalize the idea of tree-like paths. Broadly speaking, a path is tree-like if it is a constant path up to certain operations.

Definition 13. A path $X : [0,1] \to \mathbb{R}^d$ is said to be tree-like if there exists a continuous fonction $h : [0,1] \to \mathbb{R}_+$ such that $h(0) = h(1) = 0$ and such that for all $0 \leq s \leq t \leq 1$, one has

$$\|X_t - X_s\| \leq h(s) + h(t) - 2 \inf_{u \in [s,t]} h(u).$$

Definition 14. Two paths X, Y are said to be tree-like equivalent if $X * \overleftarrow{Y}$ is tree like, where $\overleftarrow{Y}_t := Y_{1-t}$ is the time reversed version of Y.

Remark. Some authors also call a path that is not tree-like a tree reduced path.

Theorem 7. Let X be a path of bounded variation. Then, the signature of X is unique up to tree-like equivalence, meaning that $S(X) = S(Y)$ if and only if $Y = X$ or Y and X are tree-like

equivalent[5].

One can also prove that a path that has at least one monotonic coordinate is not tree-like: thus, such paths are uniquely determined by their signature. For more details and an extensive discussion of this litterature, the reader is refered to Hambly and T. Lyons 2010.

B.3 Gluing Lemma

The gluing Lemma is an important tool to prove that the Wasserstein distance is, indeed, a distance.

Lemma. Let μ_1, μ_2, μ_3 be three probability measures on $\mathbb{R}^d$, and let (X_1, X_2) be an optimal coupling of μ_1 and μ_2 and (Z_2, Z_3) be an optimal coupling of μ_2 and μ_3. There exists a collection of random variables (Y_1, Y_2, Y_3) such that $\text{law}(Y_1, Y_2) = \text{law}(X_1, X_2)$ and $\text{law}(Y_2, Y_3) = \text{law}(Z_2, Z_3)$.

B.4 SVD Decomposition

Singular Value Decomposition (SVD) is a key process used in the insertion algorithm. It can be seen as a generalization of the spectral theorem (stating that all Hermitian matrices can be diagonalized in $\mathbb{R}$) to arbitrary matrices.

Theorem 8. Let M be a real matrix of size $n \times m$. M can be written as

$$M = U\Sigma V^*, \tag{51}$$

where U is a $m \times m$ real unitary matrix, Σ is $m \times n$ real diagonal matrice whose coefficients are all positive and V is a real unitary matrice of size $n \times n$.

[5] One can define an equivalence relation through the tree-like equivalence relation. See Fermanian 2018 for further explanations.

C Figure and Table Appendix

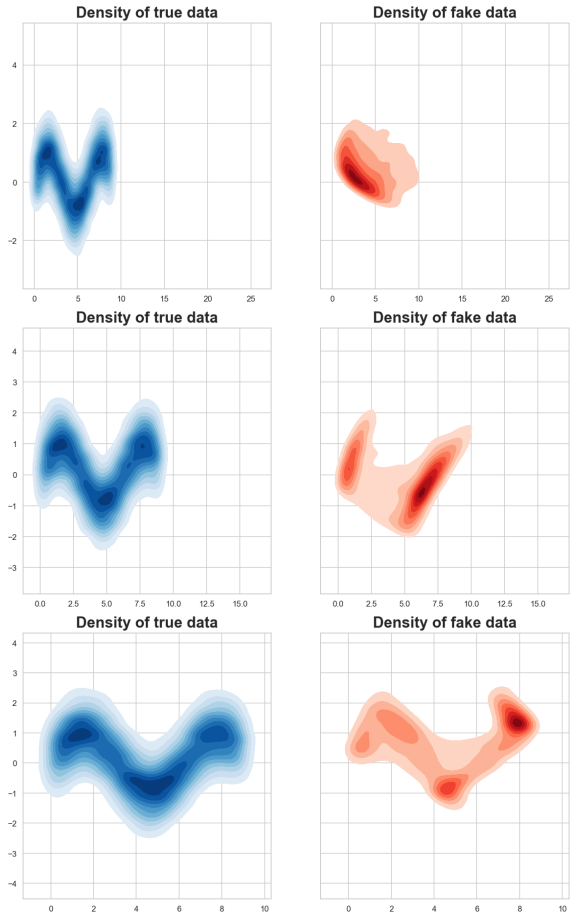


Figure 18: True (blue) and fake (red) empirical densities generated with a WGAN-GP trained for 5 000, 10 000 and 20 000 iterations (from top to bottom).

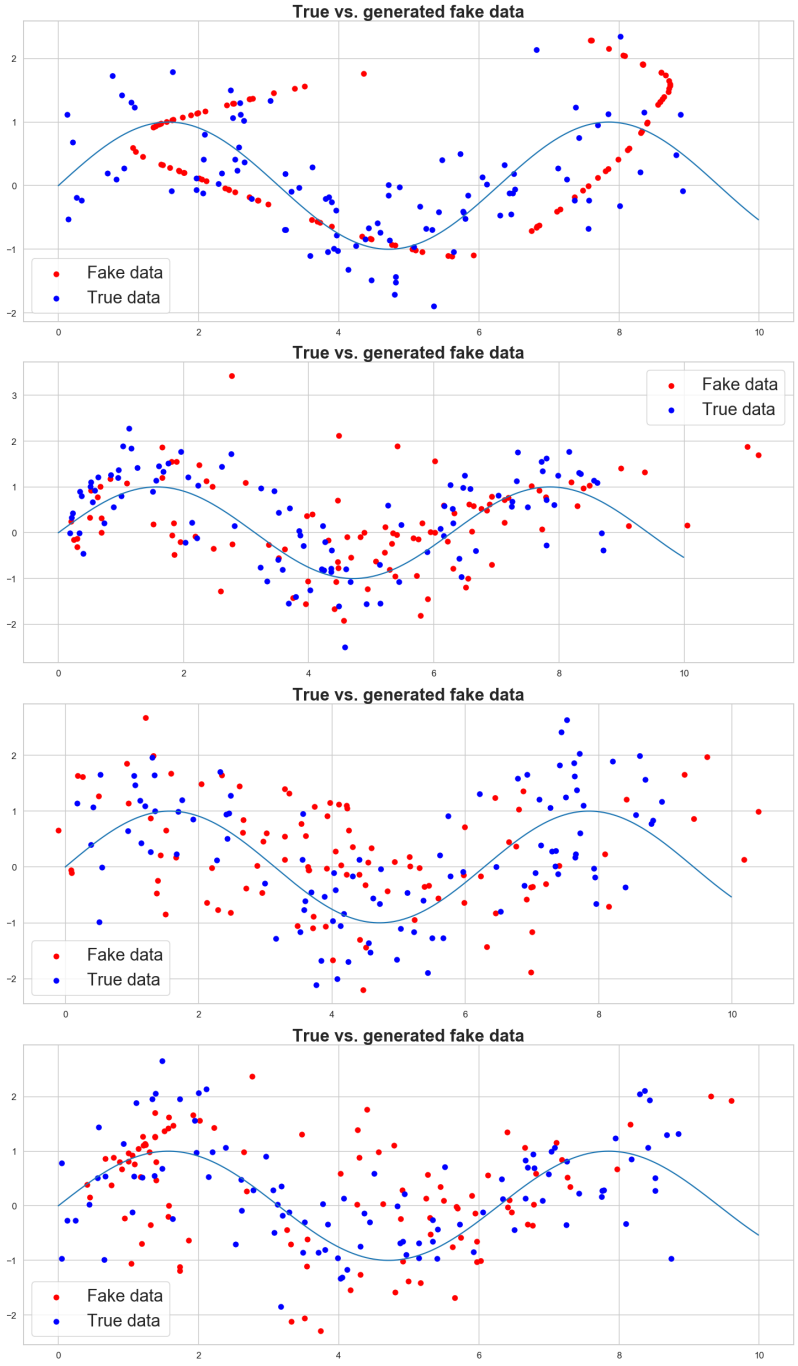


Figure 19: True (blue) and fake (red) data points generated by a WGAN-GP trained for 10 000 iterations with input noise size 1,5,10 and 50 (from top to bottom).

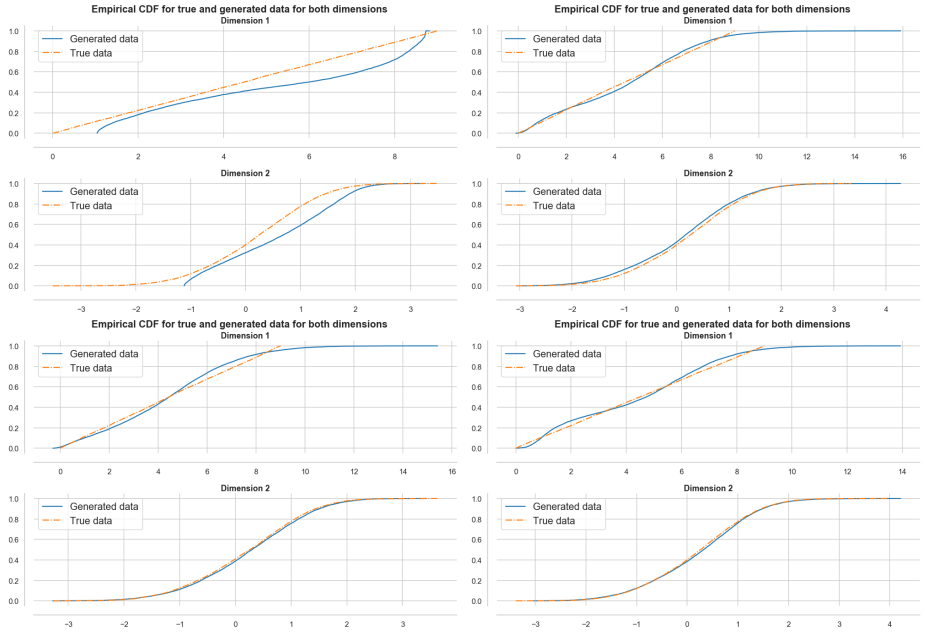


Figure 20: True (blue) and fake (red) CDFs generated by a WGAN-GP trained for 10 000 iterations with input noise size 1,5,10 and 50 (first from left to right, then top to bottom).

Noise size	**Mean metric**	**Shuffle metric**
1	0.1513	8.2566e-05
3	0.0728	1.2859e-05
5	0.0595	1.9140e-05
10	0.1547	2.4517e-05
20	0.1436	1.5056e-05
50	0.0866	1.0134e-05

Table 1: Pendigits signature generating WGAN performance for different input noise sizes. The shuffle metric is averaged over 100 signatures and 10 shuffle samples for every signature. Average value of the shuffle metric for true signatures is 1.5e-05.

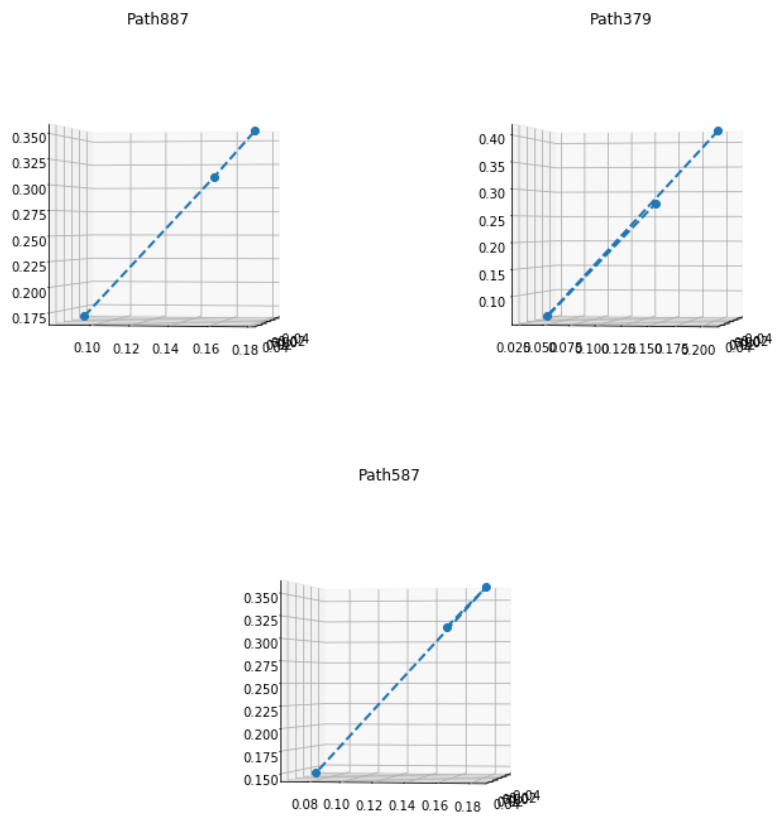


Figure 21: Some 3D path generated by inverting generated signatures of straight lines.

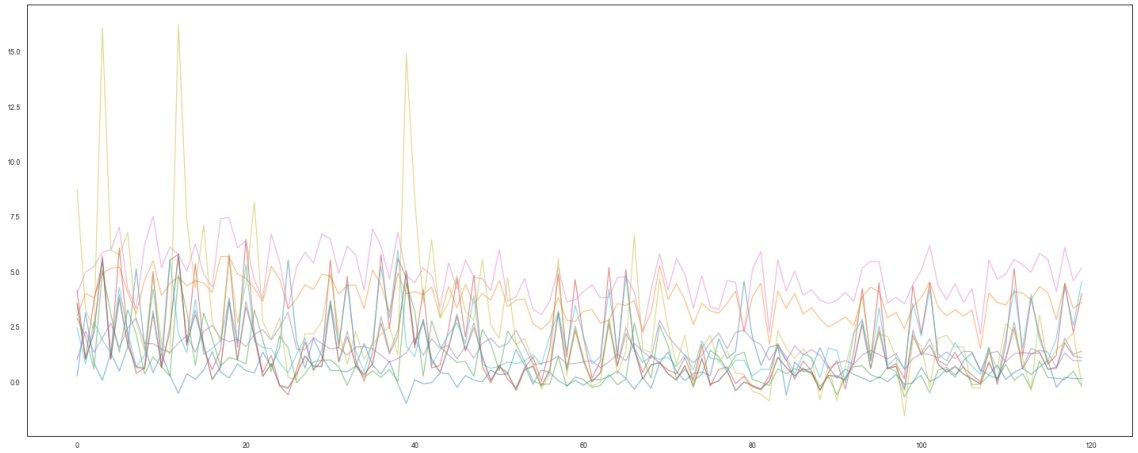


Figure 22: Fake signatures of straight lines generated at order 4 by the WGAN.

Noise size	Logistic regression	Random Forest	Nearest neighbors	Neural network
Reference	0.8939	0.9606	0.9352	0.8532
1	1	0.9033	1	0.9653
3	0.906	0.9373	0.9586	0.9393
5	0.9046	0.9066	0.9726	0.9026
10	0.9653	0.9153	0.9873	0.9793
20	0.9146	0.85	0.9546	0.9186
50	0.888	0.914	0.9646	0.932

Table 2: Accuracy of different classification techniques on signatures generated by the WGAN with different input noise sizes.

Iterations	Mean metric	Shuffle metric
5000	0.051	1.6845e-05
10000	0.136	7.9328e-06
15000	0.114	1.0078e-05
20000	0.100	2.0538e-05

Table 3: Pendigits signature generating WGAN performance for different number of iterations. The shuffle metric is averaged over 100 signatures and 10 shuffle samples for every signature.

Iterations	Logistic regression	Random Forest	Nearest neighbors	Neural network
Reference	0.8939	0.9606	0.9352	0.8532
5000	0.904	0.8133	0.9626	0.962
10000	0.922	0.7933	0.9606	0.946
15000	0.9406	0.7226	0.9826	0.8873
20000	0.9606	0.954	0.9733	0.972

Table 4: Accuracy of different classification techniques on signatures generated by the WGAN with different number of iterations.

GP parameter	Mean metric	Shuffle metric
0.1	0.142	2.8094e-05
1	0.224	2.5568e-05
5	0.056	1.2615e-05
10	0.157	4.4115e-05
50	0.070	1.2768e-05
100	0.036	1.2830e-05

Table 5: Pendigits signature generating WGAN performance for different gradient penalty parameters. The shuffle metric is averaged over 100 signatures and 10 shuffle samples for every signature.

Gradient Penalty	Logistic regression	Random Forest	Nearest neighbors	Neural network
Reference	0.8939	0.9606	0.9352	0.8532
0.1	0.9206	0.8493	0.9666	0.972
1	0.9873	0.9706	0.9913	0.984
5	0.9293	0.922	0.9806	0.9453
10	0.9946	0.9486	0.9893	0.9726
50	0.9393	0.7646	0.974	0.8986
100	0.9146	0.8693	0.9573	0.87

Table 6: Accuracy of different classification techniques on signatures generated by the WGAN with different gradient penalty parameters.