Information Sciences 609 (2022) 1539–1566

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Improving the quality of generative models through Smirnov transformation

Ángel González-Prieto [b,c], Alberto Mozo [a,*], Sandra Gómez-Canaval [a], Edgar Talavera [a]

[a] Universidad Politécnica de Madrid, Spain
[b] Universidad Complutense de Madrid, Spain
[c] Instituto de Ciencias Matemáticas (CSIC-UAM-UCM-UC3M), Spain

ARTICLE INFO

Article history:
Received 17 November 2021
Received in revised form 21 April 2022
Accepted 13 July 2022
Available online 16 July 2022

Keywords:
Generative adversarial network
Smirnov transformation
Jaccard index
generative model

ABSTRACT

Solving the convergence issues of Generative Adversarial Networks (GANs) is one of the most outstanding problems in generative models. In this work, we propose a novel activation function to be used as output of the generator agent. This activation function is based on the Smirnov probabilistic transformation and it is specifically designed to improve the quality of the generated data. In sharp contrast to previous works, our activation function provides a more general approach that deals not only with the replication of categorical variables but with any type of data distribution (continuous or discrete). Moreover, our activation function is derivable and therefore, it can be seamlessly integrated in the back-propagation computations during the GAN training processes. To validate this approach, we firstly evaluate our proposal on two different data sets: a) an artificially rendered data set containing a mixture of discrete and continuous variables, and b) a real data set of flow-based network traffic data containing both normal connections and cryptomining attacks. In addition, three publicly available data sets were added to the evaluation to generalize the obtained results. To evaluate the fidelity of the generated data, we analyze their results both in terms of quality measures of statistical nature and regarding the use of these synthetic data to feed a nested machine learning-based classifier.

The experimental results evince a clear outperformance of a Wasserstein GAN network (WGAN) tuned with this new activation function with respect to both a naïve mean-based generator and a standard WGAN. The quality of the generated data allows to fully substitute real data with synthetic data for training the nested classifier without a significant fall in the obtained accuracy.

1. Introduction

In many different domains, the application of machine and deep learning (MDL) techniques requires the availability of considerable amounts of data to take advantage of their powerful learning processes [1]. In addition, the applicability of MDL algorithms requires taking into account the evolution of data patterns over time [2], which implies to produce periodically additional volumes of relevant data for training new MDL models. Furthermore, producing the required volumes of data in industrial scenarios usually represents a considerable drawback since data gathering and processing tasks tend to

* Corresponding author.
E-mail address: a.mozo@upm.es (A. Mozo).

https://doi.org/10.1016/j.ins.2022.07.066
0020-0255/

be optimized to guarantee services and billing [3]. Even if efficient mechanisms for generating and labelling data sets can be implemented, data are increasingly protected by the legal regulations that governments impose to guarantee the privacy of their contents (e.g., European General Data Protection Regulation (GDPR) [4]). These restrictions may discourage the use of real data sets for MDL training and validation purposes.

To address this problem, in the last decade, Generative Adversarial Networks (GANs) [5] have gained significant attention due to its ability to estimate the underlying statistical structure of high-dimensional data and generate synthetic data simulating realistic media such as images, text, audio and video [6–9]. Nowadays, GANs are broadly studied and applied through academic and industrial research in different domains beyond media (e.g., natural language processing, medicine, electronics, networking, and cybersecurity).

Roughly speaking, a GAN model is represented by two independent neural networks, the so-called generator and discriminator [5], that compete to learn and reproduce the distribution of a real data set. After a GAN has been trained, its generator can produce as many synthetic examples as necessary [10], providing an efficient mechanism for solving the lack of labelled data sets and potential privacy restrictions.

1.1. Problems in GAN generation procedure and related work

There has been remarkable progress in terms of study of the theoretical properties of GANs, and well as their applications to industrial scenarios. However, less effort has been spent in evaluating GANs quantitatively and qualitatively. Several measures have been introduced mainly for measuring the image quality, but there is no consensus yet on which measures best capture the strengths and limitations of GAN models and which of them should be used for fair model comparison. Only a discrete number of quantitative criteria have emerged recently ([11–14]), although nearly all of them were designed to measure the quality of synthetic images [15].

Furthermore, there is no consensus on the deterministic stopping criteria during GAN training that produce high-quality synthetic data. Several works have already started to benchmark GANs to measure their evolution with respect to the quality of the generated data (for instance, [16–18]) and only a few of them have proposed a detention criterium not based on the visual inspection of the synthetic data (e.g., [19,20]). In fact, the former of these two papers proposes a stopping criterion that is closely related to the signal they generate, so that its direct application in other domains outside signal processing seems not very feasible.

Moreover, in [20] we observed that it was not uncommon that GANs fail to discover and replicate with sufficient quality the underlying statistical distribution of real data, in particular, if some feature was a discrete variable (e.g. a categorical feature). However, this is a cornerstone problem in generative approaches, highly demanded by an increasing number of industrial applications that need a mechanism to generate high-quality synthetic data that can fully replace real data in machine learning tasks to avoid privacy violations that might appear when using real data for training and testing purposes [21]. Therefore, providing a mechanism to generate high-quality labelled data sets that do not incur in privacy breaches will foster cross-development of MDL components by third parties. For example, a telecom provider developing ML-based components to be part of an IDS can import high-quality synthetic data from a telecom operator to train and validate these ML-based components. In general, using these data may incur in a privacy breach since these are real data from real customers. However, if the data used to train the ML system is fully synthetic, let us say with a GANs, the ML component after training will reach the desired level of performance and no breach of data privacy will occur.

To this aim, the recent work [22] proposed a solution to the generation of categorical variables in the domain of variational autoencoders (VAE) when stochastic neural networks are used, as these variables invalidate them to backpropagate through samples. Authors propose a gradient estimator that replaces the non-differentiable sample from a categorical distribution with a differentiable sample from a Gumbel-Softmax distribution. However, the proposed solution is not general enough since it does not consider continuous variables and only deals with categorical. In addition, they need to hard-code the size of each categorical variable into an array of neurons of size the number of categories of the replicated variable. Some other works in this direction propose to avoid backpropagation through discrete examples [23] using and additional pre-trained autoencoder to pass the decoded latent examples to the discriminator.

An interesting work in this line is [24]. In this paper, the authors study the effect of post-composing the generator network with a smooth map $T : \mathbb{R}^n \to \mathbb{R}^n$, showing that, in that case, the GAN tend to generated samples of the original random variable bent by T^{-1}. In some sense, the present paper exploits the same idea of bending the output of the generator agent but two key differences arise. First, in our work, the smooth transformation is specifically designed (by means of probabilistic methods) to improve the quality of the generation skills of the GAN. In this way, we handcraft a particular transformation T in such a way that the resulting GAN does not to generate a different sample twisted by T^{-1}, but the same original sample but with a higher resolution. The second key difference is that in [24] the map T is used as an external add-on to the GAN, and plays no role in the training process. In contrast, in our work, the Smirnov transformation in added as activation function of the generator network and hence it plays an active role in the backpropagation algorithm, leading to a more stable and accurate training.

Additionally, it is worth mentioning that the vast majority of the proposed GAN solutions in industrial domains do not apply any systematic measurement procedure for evaluating the GAN evolution during training [15]. This prevents the original data to be replaceable by the generated synthetic data and forces them to mix real and generated data to achieve accept-

able thresholds of quality. Unfortunately, these data augmentation solutions are not applicable in scenarios in which data privacy must be guaranteed as they use a combination of real and synthetic data. For instance, in the works [25,26], a data augmentation method is proposed to generate adversarial attacks against network Intrusion Detection Systems (IDSs). Furthermore, this idea was explored in [27], where an augmentation method with GANs was proposed to improve network traffic data sets with highly imbalanced classes.

Finally, we would like to point out that the Smirnov transformation (also known as inverse transform sampling) used in this paper is a well-known technique in probability and statistics that has been applied to data generation and random number sampling, among others. However, to the best of our knowledge, this transformation has never been used as a final stage of a GAN to improve the quality of its samples generated, but only as a static resource leading to a naïve generation. In sharp contrast, our proposal is precisely to bring together these two approaches and to use the Smirnov transformation as an intrinsic part of the GAN procedure, joining the best of both worlds.

1.2. Contribution

The main contribution of this work is to introduce a novel application of custom activation functions to the last layer of GAN generators to obtain synthetic data that, in the experiments carried out in our study, replicates with high fidelity the statistical behaviour of real data. We provide evidence of the robustness of this solution in several scenarios, including when dealing with data that contain discrete features (i.e., variables that follow a discrete distribution). The proposed custom activation function is based on the Smirnov transform (ST) and allows the GAN generator to mimic real data following even complex statistical distributions.

Roughly speaking, given an initial random variable Y and a target random variable X, the Smirnov transformation is a real-valued function $\mathcal{S}_{Y\to X}:\mathbb{R}\to\mathbb{R}$ such that the transformed random variable $\mathcal{S}_{Y\to X}(Y)$ distributes as X. In other words, $\mathcal{S}_{Y\to X}$ is a function that 'bends' the shape of the distribution of the input random variable Y and turns it into the distribution of the output X. Notice that the function $\mathcal{S}_{Y\to X}$ is completely deterministic with no stochastic behaviour, and moreover, derivable, which allows it to be seamlessly integrated in the backpropagation computations during the GAN training processes.

We apply this transformation to the case of the output of a GAN. Empirically, it can be observed that the outputs of a generative network that uses linear activation in the units of the last layer tend to follow a normal-like distribution (e.g. unimodal, with non-compact support, approximately symmetric, etc.). This behaviour discourages its use when the variables to be replicated are discrete or follow complex continuous distributions. To bend this output function to fit the actual distribution to be generated, we propose to use the Smirnov transform $\mathcal{S}_{\mathcal{N}\to X}$ as the activation function in the units of the last layer of the GAN to convert each normal distribution into the objective distribution X. This function $\mathcal{S}_{\mathcal{N}\to X}$ can be computed per output unit beforehand just by analyzing the training data set so that it is fixed before starting the training process of the GAN.

To show the effectiveness and utility of this proposal, we have conducted the following analysis and obtained the following conclusions:

1. We demonstrate in our experiments that the data generated with our solution presents a high quality comparable with the original data. This generative power is tested through different data sets: a) A rendered data set containing a mixture of discrete and continuous variables, b) a real data set of flow-based network traffic data containing benign user connections and cryptomining attacks, c) the "UCI Adult (Census Income)" data set, a publicly available data set from the renowned UCI machine learning repository, d) the "UCI E-Shop (Clickstream data for online shopping)" data set, a publicly available data set containing information on clickstream from online store offering clothing for pregnant women and e) the CTU-13 data set, a publicly available labeled data set with botnet, normal and background network traffic.
2. We provide empirical evidence of the quality improvement of our ST-based GANs with respect to state-of-the-art Wasserstein GANs (WGANs) and simple mean-based generators. Using WGANs we propose an ablation experiment that substitutes the linear activation functions of the last layer of the generator network (Vanilla WGAN) by Smirnov transform functions (ST-based WGAN). The data fidelity produced by vanilla and ST-based WGANs is assessed in two different ways: a) From a statistical perspective, we use two distance functions (based on the L^1 distance and Jaccard index) that allow us to quantitatively and graphically compare the evolution of the quality of the data generated by ST and vanilla WGANs during the GAN training; and b) from a practical viewpoint we test the performance of a nested machine learning (ML) classifier when the synthetic data generated by ST-based and vanilla WGANs completely substitute real data in the training of the classifier, comparing the performance of both classifiers when they are tested with real data.
3. The results obtained in these tests clearly outperform the existing solutions in the literature. In our experiments, the ST-based WGANs generate synthetic samples that can replace real data without leading to a significant fall in the performance of the nested model. To the best of our knowledge, this is the first work in which this goal is achieved outside the image generation domain.
4. Due to the ill-convergence of the GAN training, we do not use as stopping criterion the usual procedure based on stopping the training after a fixed number of epochs. In substitution, we introduce a novel approach based on the observed performance of a nested ML task that uses the synthetic data produced by the generator at each epoch.

Beyond the high quality of the obtained data itself, this approach has several collateral relevant consequences in GAN generation procedure, among which we can highlight the following:

1. Our solution provides a more general approach that deals not only with the replication of categorical variables but with any type of data distribution (continuous or discrete). To convert a standard GAN into a ST-based GAN we only need to change the activation functions of the last layer units of the generator by the corresponding ST functions. The ST activation functions are pre-computed before starting the GAN training and therefore, the additional computational effort can be considered negligible in the context of a full GAN training. In addition, when categorical or discrete variables are considered, it is not necessary to adapt the architecture of the generative network to the structure of the data, as it happens in the existing solutions. On the contrary, we only need to change the activation function of each output neuron in the generator with the corresponding ST activation, independently of the number of categories of the replicated variable.
2. The proposed ST-based activation function, and in particular, the spline-based implementation we used in our experiments, are derivable. Therefore, the Smirnov transformation can be integrated as the activation function of a neuron without creating any problem to the backpropagation algorithm that is used during the GAN training process. Recall that the backpropagation algorithm is based on computing the derivatives of the cost function with respect to the neural network parameters. For that reason, it is required that all the functions that form part of the neural network, and in particular the activation functions of the neurons, are derivable.
3. The generation procedure avoids the privacy violations that could appear when real data is used for different tasks (e.g., in MDL training and testing processes, and when sharing data in cross-developments or federated environments).
4. The generative power of our solution can be used to alleviate the existing shortage problem of available (labelled) data sets in many domains.

1.3. Paper structure

The rest of the paper is structured as follows: Section 2 presents a brief review of Generative Adversarial Networks. A detailed description of the Smirnov Transform we propose as the activation function in the generator is shown in Section 3. The performance metrics we use to compare the quality and similarity of the generated data are depicted in Section 4. The empirical evaluation of the ST-based GAN and the obtained results are presented in Section 5. We summarize the innovative aspects of our work and conclude with some interesting open research questions in Section 6.

2. Preliminaries

In this section we provide a brief review of GANs. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, where Ω is a set (the sample space), $\mathcal{F}$ is a σ-algebra on Ω and $\mathbb{P}$ is a probability measure. Let us also consider a random vector $X : \Omega \to \mathbb{R}^n$, i.e. a measurable function from Ω to $\mathbb{R}^n$ endowed with the Borel σ-algebra. Typically, neither the space Ω nor the variable X are fully known, and only very limited access to them is provided, for instance, through a collection of samples. The aim of a generative model is to 'replicate' X as reliably as possible, but admitting certain variability. In other words, the goal is the following: given a fixed probability space $(\Lambda, \mathcal{G}, \lambda)$, called the latent space, create a random vector $G : \Lambda \to \mathbb{R}^n$ such that G is 'near' to X in some sense (typically, in distribution).

The proposal of a Generative Adversarial Network (GAN) is to tune two different functions: a generator $G : \Lambda \to \mathbb{R}^n$ and a discriminator $D : \mathbb{R}^n \to \mathbb{R}$. The generator G will be trained to generate as faithful samples to X as possible, while the discriminator D is a binary classifier optimized for distinguishing between 'real' samples of X and 'fake' samples generated by G. Throughout this paper, we shall adopt the convention that the output of D is the likelihood of a sample x to be real, measured in logits. In other words, if $\sigma(t) = (1 + e^{-t})^{-1}$ is the logistic function, then $\sigma(D(x)) = 1$ means that D believes that x is a real sample whereas $\log(D(x)) = 0$ stands for D believing that x is a fake sample.

The classification error suffered by the classifier D is thus

$$\mathbb{E}_\Omega[-D(X)] + \mathbb{E}_\Lambda[D(G)],$$

where $\mathbb{E}_\Omega$ and $\mathbb{E}_\Lambda$ denote the mathematical expectation on Ω and Λ respectively. It is customary to weight this error with an increasing concave function $f : \mathbb{R} \to \mathbb{R}$ so that we instead consider the error function of D

$$\mathcal{E}(G, D) = \mathbb{E}_\Omega[f(-D(X))] + \mathbb{E}_\Lambda[f(D(G))]. \tag{1}$$

Typical choices for f are $f(s) = \log(\sigma(t))$ as in [5], or $f(s) = s$ as in the Wasserstein GAN (WGAN) [28].

To improve this error, we will suppose that both G and D depend on some parameters (usually, they are implemented as neural networks) and we shall adjust these parameters to optimize the error $\mathcal{E}$. However, notice that the parameters of the discriminator D are trained to minimize $\mathcal{E}$, whereas the generator G aims to cheat D, so it seeks to maximize $\mathcal{E}$. This gives rise to the competitive game

$$\max_G \min_D \mathcal{E}(G, D) = \max_G \min_D \mathbb{E}_\Omega f[-D(X)] + \mathbb{E}_\Lambda f[D(G)]. \tag{2}$$

Beware of the sign conventions. Sometimes in the literature, the error function (1) is weighted with the decreasing function $f(-s)$, so the resulting cost function is equivalent to the one presented here but the objectives of the functions G are exchanged: G aims to minimize it while D tries to maximize it.

In this way, the goal of the training process of a GAN system is to look for Nash equilibria of (2). These are pairs (G_0, D_0) of a generator and a discriminator such that the function $G \mapsto \mathcal{E}(G, D_0)$ has a local maximum at $G = G_0$ and the function $D \mapsto \mathcal{E}(G_0, D)$ has a local minimum at $D = D_0$. Nash equilibria exhibit very good theoretical properties of probabilistic nature. For instance, in the assumption that a perfect discriminator is reachable at a Nash equilibrium, the Jensen-Shannon divergence between the synthetic distribution G and the original one X is minimized. Similarly, Wasserstein's Earth-moving distance is minimized when a WGAN is applied [28].

Despite of that, the problem of finding Nash equilibria in GANs is still essentially open. In the seminar paper [5], it is proposed a simple optimization procedure by alternating gradient descend optimization. However, as pointed out in [29], the game (2) to be optimized is not a convex-concave problem, so in general the convergence of these simple training methods is not guaranteed. To stabilize this training process, several heuristic methods have been proposed, such as feature matching, minibatch discrimination, or semi-supervised training [30]; the introduction of spurious noise [31,32] or the application of regularization methods based on gradient penalty [33].

Finally, it is worth mentioning that most of these training methods have been designed to be applied to the case in which the input data X are graphical images. In this scenario, the statistical properties of the color distribution among the pixels foster the convergence of the GAN, which may achieve high quality results. Nevertheless, when the data to be replicated exhibit different statistical properties (e.g., categorical features, heavy-tailed non-normal distribution, and strong domain restrictions), to the best of our knowledge, no general purpose training method is known and the results are typically very poor [20].

3. The Smirnov transform

Borrowed from the field of theoretical probability, there is a mathematical transformation that will be very useful for our purposes. In this section, we shall outline the main concepts and properties of this map. For further information, please refer to [34] or [35].

Suppose that $X : \Omega \to \mathbb{R}$ is a random variable and let $F_X(x) = \mathbb{P}(X \leqslant x)$ its cumulative distribution function. In the case that $F_X : \mathbb{R} \to [0, 1]$ is a continuous increasing function, then its inverse $F_X^{-1} : [0, 1] \to \mathbb{R}$ is a well-defined function called the quantile function. Otherwise, we can still define an analogue of the quantile function by setting

$$F_X^{-1}(p) = \min_z \{F_X(z) \geqslant p\}. \tag{3}$$

Recall that F_X is non-decreasing and right continuous, so the set $F_X^{-1}([p, \infty))$ is actually a final segment including the leftmost endpoint. The value $F_X^{-1}(p)$ is thus nothing but the infimum of this set, which is actually a minimum by right continuity. A key feature of the quantile function is the following result.

Proposition 3.1. Let $U \sim U[0, 1]$ be the uniform continuous random variable with support $[0, 1]$. Then $F_X^{-1}(U) \sim X$.

Proof.

This is a very well-known result whose proof can be found, for instance, in [35]. We include here a brief proof for the convenience of the reader. Notice that, for any $x \in \mathbb{R}$ and $0 \leqslant p \leqslant 1$, we have that $\min_z\{z | F_X(z) \geqslant p\} \leqslant x$ if and only if $x \in \{z | F_X(z) \geqslant p\}$, which means that $p \leqslant F_X(x)$. Therefore, we get that, for all $x \in \mathbb{R}$

$$\mathbb{P}\left(F_X^{-1}(U) \leqslant x\right) = \mathbb{P}\left(\min_z\{F_X(z) \geqslant U\} \leqslant x\right) = \mathbb{P}(U \leqslant F_X(x)) = F_X(x).$$

Thus, the cumulative distribution function of the random variable $F_X^{-1}(U)$ coincides with F_X, as we wanted to show.

There is also a partial reciprocal result, but it requires to impose extra assumptions on F_X.

Proposition 3.2. If F_X is an increasing continuous function, then $F_X(X) \sim U[0, 1]$.

Proof.

Again, the proof can be found in [35]. Since F_X is increasing and continuous, its punctual inverse is well defined. Hence, for all $x \in [0, 1]$, we have

$$\mathbb{P}(F_X(X) \leqslant x) = \mathbb{P}\left(X \leqslant F_X^{-1}(x)\right) = F_X\left(F_X^{-1}(x)\right) = x.$$

Thus, $\mathbb{P}(F_X(X) \leqslant x) = x$ for all $x \in [0, 1]$, while $\mathbb{P}(F_X(X) \leqslant x) = 0$ for $x < 0$ and $\mathbb{P}(F_X(X) \leqslant x) = 1$ for $x > 1$, which shows that X distributes as a uniform random variable.

Some remarks are in order. In the previous proposition, the hypothesis that F_X is invertible is actually too strong. Repeating the type of arguments of Proposition 3.1, it can be shown that $F_X(X) \sim U[0,1]$ provided that $P(X = x) = 0$ for any x in the support of X. This holds, for instance, for all continuous random variables. In the case that F_X is discontinuous, a closed formula for the cumulative distribution function of $F_X(X)$ can be still obtained. Indeed, if $x \in \mathbb{R}$ is a continuity point, then $\mathbb{P}(F_X(X) \leqslant x) = x$ as usual; but if $x \in \mathbb{R}$ lies in the middle of a jump singularity in which F_X jumps to a value $x^+ > x$ (in other words, $x^+ = F_X^{-1}(F_X(x))$), then we have that $\mathbb{P}(F_X(X) \leqslant x) = x^+$.

The aforementioned results allow us to transform any random variable into another distribution desired. We state this as a theorem since it is crucial for our later developments. The proof is just a straightforward combination of Propositions 3.1 and 3.2.

Theorem 3.3. Let X be an arbitrary random variable and let Y be a continuous random variable. Set F_X and F_Y for the cumulative distribution functions of X and Y, respectively. Then, we have that

$$\mathcal{S}_{Y\to X}(Y) := F_X^{-1}(F_Y(Y))$$

is a random variable that distributes as X. The new random variable $\mathcal{S}_{Y\to X}(Y)$ is called the *Smirnov transform of Y into X*.

Notice that Proposition 3.1 does not require any assumption on the distribution of X, so the target distribution may be anything. However, to apply Proposition 3.2, we need that the original variable Y is continuous. Recall that, in the case that F_X is not continuous or increasing, the quantile function F_Y^{-1} is defined as in (3).

3.1. Empirical estimation of the Smirnov transform

Let us consider the scenario of Theorem 3.3, in which we want to transform a random variable Y into another random variable X. Typically, the distribution of Y will be known, but the actual distribution of X might be unclear (for instance, because it is a very involved phenomenon).

To address this issue, we propose to estimate it through the so-called empirical cumulative distribution function, denoted by $\widehat{F}_X$. To this purpose, we shall have access to a collection of samples $x_1, \dots, x_m$ of X, and we define $\widehat{F}_X$ by

$$\widehat{F}_X(x) = \frac{1}{m}\sum_{i=1}^{m} \chi_{(-\infty, x_i]}(x),$$

where χ_A is the characteristic function of the set A, that is, $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ otherwise.

By the Glivenko-Cantelli theorem, when $n \to \infty$, the empirical cumulative distribution function $\widehat{F}_X$ converges to the real distribution function F_X in the L^∞ distance almost surely. This means that, for large n, $\widehat{F}_X$ is a very good estimator of F_X. In particular, we can approximate the Smirnov transform $\mathcal{S}_{Y\to X}(Y)$ by the *empirical Smirnov transform*

$$\hat{\mathcal{S}}_{Y\to X}(Y) = \widehat{F}_X^{-1}(F_Y(Y)).$$

Here, $\widehat{F}_X^{-1}$ is defined as in (3) by $\widehat{F}_X^{-1}(p) = \min_z\left\{\widehat{F}_X(z) \geqslant p\right\}$.

3.2. Smirnov transform as GAN activation

After this probabilistic digression, let us come back to the problem of training GANs. Suppose that we want to generate a random vector $X = \left(X^1, \dots, X^n\right)$ of which only some samples $x_1 = (x_1^1, \dots, x_1^n), \dots, x_m = (x_m^1, \dots, x_m^n) \in \mathbb{R}^n$ are known. For this purpose, we want to train a GAN made of a generator $G = (G_1, \dots, G_n) : \Lambda \to \mathbb{R}^n$ and a discriminator $D : \mathbb{R}^n \to \mathbb{R}$.

The key problem is that, typically, without further tuning, the output distribution of each of the random variables $G_i : \Lambda \to \mathbb{R}$ is approximately normal. This is related with the mode-collapse problem [36], a well-reported behaviour of the GANs in which, when they try to generate non-normal variables, the output data tend to not be variate, and the generator degenerates to synthesize only small variations of a prototypical example of X. In some sense, the network G stucks in a particular example of X that cheats D very efficiently and ignores any other type of data to generate. This is optimal from the point of view of the GAN game (2), but leads to a degenerate behaviour in which the real distribution of X is not recovered.

To address this problem, in this work we propose to facilitate the job of the generator G by using as activation function a customized function able to capture the statistical subtleties of X. To be precise, let us denote by $F_{\mathcal{N}}$ the distribution function of the standard normal distribution, that is

$$F_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-s^2/2}\, ds.$$

Additionally, using the sample $x_1 = (x_1^1, \ldots, x_1^n), \ldots, x_m = (x_m^1, \ldots, x_m^n) \in \mathbb{R}^n$ of the n-dimensional random vector $X = \left(X^1, \ldots, X^n\right)$ we generate the n marginal empirical cumulative distribution functions $\widehat{F}_{X^1}, \ldots, \widehat{F}_{X^n}$. With this information, we create a new activation function as the juxtaposition of the Smirnov transformations from the standard normal distribution to X_i

$$\hat{\mathcal{S}}_X = \left(\hat{\mathcal{S}}_{\mathcal{N}\to X^1}, \ldots, \hat{\mathcal{S}}_{\mathcal{N}\to X^n}\right) = \left(\widehat{F}_{X^1}^{-1} \circ F_{\mathcal{N}}, \ldots, \widehat{F}_{X^n}^{-1} \circ F_{\mathcal{N}}\right) : \mathbb{R}^n \to \mathbb{R}^n.$$

To speed up the evaluation of the function $\hat{\mathcal{S}}_X$, as well as to avoid the vanishing gradient problems derivated from the piecewise constant nature of the functions $\widehat{F}_{X^1}^{-1}$, instead of using $\mathcal{S}$ we shall interpolate each of the component functions $\hat{\mathcal{S}}_{\mathcal{N}\to X^i}$ through a numerical interpolation method (typically, spline interpolation [37], but also Yoshida smoothing can be applied) to get approximate functions $\tilde{\mathcal{S}}_{\mathcal{N}\to X^i}$. Now, the interpolated global function is given by

$$\tilde{\mathcal{S}}_X = \left(\tilde{\mathcal{S}}_{\mathcal{N}\to X^1}, \ldots, \tilde{\mathcal{S}}_{\mathcal{N}\to X^n}\right) : \mathbb{R}^n \to \mathbb{R}^n.$$

In this way, as the activation function of the output layer of the generator network G we shall use the function $\tilde{\mathcal{S}}_X$. The gradients of the error suffered by this function will be propagated towards the initial layers of the network as usual in the usual backpropagation algorithm. Notice that, provided that the interpolation method returns C^1-functions $\tilde{\mathcal{S}}_{\mathcal{N}\to X^i}$ (i.e. derivable with continuous derivatives), then the activation function $\tilde{\mathcal{S}}_X$ is a differentiable map. This is the case, for instance, of spline interpolation, allowing us to deal with discrete distributions even though their underlying quantile function is not smooth.

The rationale behind this choice of activation function is the following. Suppose that $G : \Lambda \to \mathbb{R}^n$ is a generator for the random vector X implemented by a standard GAN. In this setting, G will tend to present normal marginal distributions according to the mode-collapse problem [36]. However, if instead of G we consider the new generator $\widetilde{G} = \tilde{\mathcal{S}}_X \circ G$, then Theorem 3.3 assures that $\widetilde{G}$ does (approximately) fulfills the marginal distributions of X, even though they may be discrete or with compact support. Furthermore, Glivenko-Cantelli theorem [38] assesses that this approximation is asymptotically consistent, in the sense that when the number of sample points increases, the marginal distributions of $\widetilde{G}$ converge (in distribution) to the marginal distributions of X.

As a final comment, notice that this Smirnov transformation converts random vectors with normal marginal distributions into random vectors with approximately marginal distribution X^i. However, the global dependence between the different output variables is not captured by $\tilde{\mathcal{S}}_X$. Far from being a problem (which may be addressed for example with copulas [39]), this is an advantage of this approach: with the use of $\tilde{\mathcal{S}}_X$ as activation function, we ease the job of the generator network of generating the marginal distribution, so that it can focus on the capture of the non-linear interrelations among the different components, which is typically the hardest part to generate.

4. Performance metrics

We propose to evaluate GANs performance using two different types of metrics. The first set of metrics is inspired by the L^1 functional distance and the Jaccard coefficient [40] and aims to quantify the similarity of the synthetic data with respect to the real data from a statistical perspective, considering the joint distribution of data features. On the other hand, the second set of metrics attempts to quantify the performance of synthetic data when it is used as a substitute for real data in the training of a ML classifier that aims to distinguish among the different types of data contained in the data set. To apply this metric, it is obvious to assume that the real data are labelled and that the data set contains more than one type of data.

These two types of metrics will be used to compare the similarity between real and synthetic distributions, and the later set will also be applied to implement a stopping criterion for GAN training that will allow us to select the best generators producing high-quality synthetic data. It is worth noting that in preliminary experiments we compared the aforementioned metrics with standard Jensen-Shannon and Wasserstein distributional distances. We finally decided to include only the former in our experiments, since the latter sometimes exhibited oscillatory behaviours that were not present in the former.

4.1. L^1 distance and Jaccard index

These two metrics try to measure the difference between the probabilistic distributions of real and synthetic data. They are based on two well-known statistical coefficients applied for hypothesis testing and probabilistic distances: the L^1-metric and the standard Jaccard coefficient.

Although both metrics use the probability density function of the two data distributions to compute the distance, they can be straightforwardly extended to a more practical scenario where the density functions of the data distributions are not known. Instead, we shall compute an empirical estimator through the histogram to replace the probability density function.

In the following, we shall sketch briefly the main ideas involved in the construction and estimation of these quality metrics. For further details, please refer to [20].

Empirical probability density function. Let us suppose that we have samples $x_1, \ldots, x_n \in \mathbb{R}^d$ of a d-dimensional random vector X. Let us choose a partition of the support of X into disjoint cubes $C_1, \ldots, C_s$. For simplicity, we shall take all cubes C_i of the same volume. The empirical probability density function $h_X : \mathbb{R}^d \to \mathbb{R}$ is the function

$$h_X(x) = \frac{1}{n}\sum_{j=1}^{s} \left|\{x_i \in C_j\}\right| \chi_{C_j}(x),$$

where χ_{C_j} is the characteristic function of the cube C_j (i.e. $\chi_{C_j}(x) = 1$ if $x \in C_j$ and is 0 otherwise) and $\left|\{x_i \in C_j\}\right|$ stands for the number of samples that belong to the cube C_j. By the Glivenko-Cantelli theorem [38], the empirical probability density function h_X is a faithful estimator of the actual probability density function of X.

L^1 Distance. Given two continuous d-dimensional random vectors X and Y with probability density functions f_X and f_Y, we can consider the L^1 distance between their probability density functions, that is

$$d_{L^1}(X, Y) = \int_{\mathbb{R}^d} |f_X(s) - f_Y(s)|\, ds.$$

Notice that $d_{L^1}(X, Y) = 0$ if and only if $X = Y$ almost sure.

However, in applications, it is not common to explicitly know the probability density functions of X and Y. Instead, from a collection of samples $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ of X and Y, respectively, we can compute their empirical probability density functions h_X and h_Y. In this way, the empirical L^1 distance can be taken as

$$d^{\text{emp}}_{L^1}(X, Y) = \int_{\mathbb{R}^d} |h_X(s) - h_Y(s)|\, ds = L\sum_{j=1}^{s} |h_X(C_j) - h_Y(C_j)|,$$

where L is a constant depending only on the volume and number of elements of the partitions taken and $h_X(C_j)$ (resp. $h_Y(C_j)$) is the value of the function h_X (resp. h_Y) on C_j. In this empirical setting, we have that $d^{emp}_{L^1}(X, Y) = 0$ if and only if $h_X(C_j) = h_Y(C_j)$, i.e. if and only if the number of samples of X on each cube C_j equals the number of samples of Y.

Jaccard index. This metric is designed to compare the supports of two distributions. In this way, instead of looking at the particular distribution function, the aim of this metric is to determine whether the two random variables satisfy the same value constraints.

Suppose that we have two random variables X and Y with supports $\text{supp}(X)$ and $\text{supp}(Y)$, respectively. The Jaccard index of X and Y is

$$J(X, Y) = \frac{|\text{supp}(X) \cap \text{supp}(Y)|}{|\text{supp}(X) \cup \text{supp}(Y)|},$$

where $|A|$ stands for the Lebesgue measure (i.e. the volume) of the measurable set A. This coefficient takes values in the interval $[0, 1]$ and the larger the value of $J(X, Y)$ the more similar the empirical supports.

Again, if the real support is not known, we can still estimate it through the empirical probability density functions as

$$J^{smp}(X, Y) = \frac{|\text{supp}(h_X) \cap \text{supp}(h_Y)|}{|\text{supp}(h_X) \cup \text{supp}(h_Y)|}.$$

4.2. Nested ML performance

The second set of metrics attempts to quantify the performance of synthetic data when it is used as a substitute for real data for training a ML classifier.

To be precise, suppose that our data set of real data, let us call it DS, is labelled for a supervised classification ML task. In other words, DS contains instances of $s \geqslant 2$ different classes which are appropriately labelled. For the sake of notational simplicity, we shall consider the case $s = 2$ of binary classification (as appears in the experiments of this work), but the approach can be straightforwardly generalized.

In order to train a ML model, as usual, we can split DS into two data sets, DS1 and DS1, with similar statistical properties. In this way, DS1 can be used for training a ML classifier, whereas DS2 is reserved for testing its accuracy through the standard classification quality measures: F_1-score, precision and recall.

Nevertheless, in addition to being used for the training of the ML classifier, DS1 can also be used to train GANs aiming to replicate its data. Hence, using DS1 we train two GANs (Λ_0, G_0, D_0) and (Λ_1, G_1, D_1) to synthesize data with label 0 and 1 respectively. Choose $N, M > 0$ and draw samples $x^0_1, \ldots, x^0_N$ and $x^1_1, \ldots, x^1_M$ of the latent spaces Λ_0 and Λ_1 respectively. Then, using the generators G_0 and G_1, we create a new fully synthetic training data set DS1′ with $N + M$ instances joining the synthetic data generated by both GANs.

With this new synthetic dataset DS1′, we train a standard ML classifier (say, a random forest classifier [41]). Then, screening the precision, recall, and F_1-score of the classifier against DS2, we are able to measure the quality of the generated data: the higher these measures, the better the synthetic data that was generated by (Λ_0, G_0, D_0) and (Λ_1, G_1, D_1). Hence, large values of these coefficients point out that the synthetic data generated by G_0 and G_1 can be used to faithfully substitute the real instances. Observe that no real data is used for such training purposes, although real data is always used for testing.

Additionally, as a baseline comparison for the metrics obtained with GAN synthetic data, we can also consider the ML classifier trained with DS1 and compute its performance metrics with DS2 as the testing data set. In this way, we can compare the performance of the ML classifier trained with GAN synthetic data against the benchmark-level performance obtained using real data during the training of the ML classifier. Notice that our approach highly differs from many existing works that only mix real with synthetic data (e.g., data augmentation solutions), which can generate data privacy breaches as real data is present in the resultant data set.

It is worth mentioning that the particular choice of the ML method to test the nested accuracy is not really important. This procedure seeks to discriminate between generated data of high and poor quality according to whether the classifier can learn from the data. In this way, it is not a matter of getting the highest possible performance, which may require exhaustive testing and hyperparameter tuning, but of analyzing the ability of the model to extract information from the generated data. In this idea, a sufficiently flexible ML algorithm, like Random Forest, with a rudimentary hyperparameter tuning is enough to obtain a compelling metric of the quality of the data.

Marginal quality evaluation. From a practical perspective, beyond the aforementioned process, we can also evaluate the marginal quality of each of the generators before jointly evaluating the quality of G_0 and G_1. We generate only one of the types of data, say label 0, and we mix the synthetic samples of label 0 with real samples of label 1 obtained from DS1. This data set is used to train a ML classifier and then the classifier is tested on DS2 to get the performance measures. This process is repeated for each type of data. In this way, the corresponding ML accuracy coefficients will only measure the ability of G_0 to generate label 0, regardless of the fitness of G_1 and vice versa.

In our experiments, we apply a variant of this approach that computes the performance of the ML classifier at each of the training epochs of the GAN. In this way, we are able to screen the evolution of the training and to relate it to the quality of the generated data. In particular, this idea enables a novel stopping criterion: when the GAN training epochs do not produce any significant enhancement in the performance of the ML classifier, the training process of the GAN is stopped. It is worth noting that this approach allows to train each type of GAN in parallel and therefore, each training can be stopped at different epochs when no significant enhancement is observed in a particular GAN.

Choice of the best GAN model. After each GAN is trained, the joint performance of both types of synthetic data is computed. Using the whole set of GANs obtained during the marginal quality evaluation would imply to compute the ML performance for each pair (G_0, G_1) of generators G_0 and G_1 at each of their training epochs. This leads to a quadratic number of generators to be tested in the ML task, both for training and testing, to obtain the full set of measurements.

However, we observed experimentally that drawing roughly a dozen samples by choosing uniformly at random one generator of each type of data tends to produce results equivalent to the brute force approach of trying all possible combinations. In addition, we applied more elaborated strategies based on performance elitism, ordering the generators of each type of traffic by F_1-score and choosing generators at random only from the subset containing the best generators.

Finally, we would like to remark that, since we have generative models able to create as many samples as needed, we can choose the number of generated samples N and M as large as desired. If we choose N and M in the same range as the number of instances in the original dataset DS1, we obtain a synthetic data set with very similar characteristics to the original one. In particular, any unbalancing between classes will remain. However, other choices can be made. For instance, we can decide to take $N = M$, so that the obtained data set corrects the unbalanced situation, or to take N and M much larger than the size of the original dataset, so that we increment the amount of data available for the ML classifier. It is worthy to mention that, even though this solution gives rise to a balanced dataset as with data augmentation procedures, the proposed solution is substantially stronger than simple data augmentation: the synthetic data is not a simple enrichment of the original data set but a completely new data set.

5. Experiments

With the aim of demonstrating the effectiveness of the ST-based solution, we trained state-of-the-art Wasserstein GANs (WGAN), using linear activation and Smirnof transform functions respectively in the output of the generator network. Initially, we selected two labelled data sets with interesting properties to investigate whether the ST-based WGAN can replicate the complexity of these data sets with higher fidelity. The performances of the two WGAN configurations were compared by running a set of experiments for each data set and label. These two data sets have been made publicly available to facilitate further experimentation and reproducibility of our experiments and can be found in [42].

In addition, we extended our experiments to three publicly available data sets to confirm the previously observed properties of the ST-based solution. In the rest of the section, we describe the data sets used for the experiments, the details of the two WGAN architectures, and the obtained performance of the two solutions.

5.1. Dataset description

To demonstrate the applicability of the proposed solution, we have defined two different use cases to experimentally validate the versatility of ST-based GANs. The general objective of the two use cases is to test whether GAN-generated synthetic traffic can fully replace real traffic in ML problems where the use of real data could generate privacy breaches. The two datasets used in this first round of experiments are publicly available in [42] to foster further experimentation and to allow other researchers to use them as benchmark datasets.

First use case: Rendered data set. For the first use case considered in this work, we have designed two fictitious data sets (denoted by DS1-r and DS2-r). As previously commented in subSection 4.2, we train GANs with DS1-r, while DS2-r is used for training the ML-classifier that will assess the quality of the synthetic data generated by the GANs. In addition, DS1-r is used for training the benchmark ML classifier. Both data sets contain entries of two different data types (i.e., 2 labels) and are perfectly balanced, containing $400,000$ entries for each label. The data sets are composed of four variables, two of them contain continuous values and the other two contain discrete values. Each of the entries of the data set was generated by sampling from two random vectors (one per label class) made of independent random variables with the distributions shown in Table 1 (see also their histogram in Figs. 7 and 8). Notice that, to hinder the generation of the data, fixed the output class the features are independent random variables and thus they are completely uncorrelated. In this way, a generative model cannot rely on a simple feature to generate a complex one, but it has to be able to replicate them simultaneously.

This use case aims to demonstrate that the use of linear activation functions at the last layer of the generator fails to replicate complex data distributions such as the ones we have rendered and in particular, the variables representing discrete data distributions. On the contrary, we show that, using ST-based activation functions, we are able to perfectly replicate from a statistical point of view, both continuous and discrete variables, even if the data variables follow complex statistical distributions.

It is worth noting that the statistical distributions of the two types of data have been generated in such a way that they are similar on average, which makes the task of an ML classifier more complicated when we want to train it to correctly identify the two types of data. Indeed, if the synthetic data have not been generated with sufficient fidelity to the two real data distributions, because the means of their 4 variables are so close, the ML classifier trained with synthetic data will obtain a significantly worse performance in terms of accuracy, precision and recall than a benchmark classifier trained with real data.

Finally, observe that some of the distributions of the 8 variables (4 per data type) have been generated with statistical patterns different from the Gaussian distribution (Table 1) to demonstrate that the generators with linear activation do not replicate with precision the real distributions when they are not Gaussian or discrete, and that on the contrary, when the generators have activation functions based on ST, the distributions of the synthetic data exactly replicate the real variables even if their distributions follow statistical patterns very different from Gaussian distributions (e.g. discrete distribution).

Second use case: Network traffic. The second use case aims to evaluate the replication by GANs of data coming from a real scenario in the cybersecurity domain. The real data used in this experiment were previously generated in a realistic network laboratory called the Mouseworld lab [43]. The Mouseworld is a network digital twin created at Telefónica R + D facilities that allows deploying complex network scenarios in a controlled way. In this lab, a set of virtual machines were deployed for the generation of regular network traffic (e.g., web and video flows) jointly with cryptomining clients connected to public mining pools in the Internet [44].

We ran the experiment twice for one hour, collected the transmitted packets, and generated two data sets (denoted by DS1-c and DS2-c) each with 4 millions of flow-based entries containing statistics of the connections. Normal traffic connections were labelled with 0 and cryptomining ones with 1. It is worth noting that both data sets are totally unbalanced, containing only $4,000$ entries of criptomining connections.

A set of 59 statistical features were extracted from each TCP connection, although we selected a reduced set of 4 for our experiments: (a) number of bytes sent from the client, (b) average round-trip time observed from the server, (c) outbound bytes per packet, and (d) ratio of packets inbound/ packets outbound. These four features were selected as they exhibit two interesting properties for our generative experiments that were previously commented in the first use case: (i) each feature presents a different statistical behaviour far from a Gaussian distribution and (ii) the mean of each feature in the two types of traffic (normal and cryptomining) were close, which makes the task of an ML classifier more complicated when we want to train it to correctly identify the two types of data.

The nature of both types of traffic is very different, a fact that will be reflected in the quality of the GAN-generated data. The normal traffic has a great variety since it is composed of many types of connections (e.g., video, audio, web elements, and

Table 1
Distributions used to render data sets DS1-r and DS2-r. Each of the features was drawn independently.

Label	Feature 0	Feature 1	Feature 2	Feature 3
Label 0	Normal ($\mu = 0, \sigma = 1$)	Binomial ($n = 15, p = 0.3$)	Exponential ($\sigma = 3$)	Poisson ($\lambda = 1.0$)
Label 1	Normal ($\mu = 0, \sigma = 1$)	Discrete uniform (supp.$[0, 15]$)	Snedecor F ($v_1 = v_2 = 3$)	Poisson ($\lambda = 2.0$)

multimedia elements). On the contrary, the cryptomining connections are handled by a reduced set of protocols and therefore, their statistical patterns are not expected to differ substantially. Due to the greater diversity that normal traffic connections exhibit when compared to cryptomining connections, GANs that try to replicate label-0 data will perform slightly worse than their counterparts that replicate label-1 data.

5.2. Additional data sets

In addition to the former data sets, we selected three publicly available data sets to demonstrate in different scenarios that the ST activation function used in a Wasserstein GAN generates synthetic data of greater quality and performance than when the normal linear activation function is applied to the same WGAN. We choose two data sets from the renowned UCI repository and one containing a set of network captures including normal traffic and several botnet attacks:

1. UCI Adult data set also known as *Census Income data set*(https://archive.ics.uci.edu/ml/datasets/adult). This data set is used to predict whether income exceeds $50 K/yr based on census data [45]. The data set contains 14 features, one of them (Income) was used as the label for the nested classification task and the rest were used as the objective data to be replicated by the GANs.
2. UCI E-shop data set also known as Clickstream data for online shopping data set (https://archive.ics.uci.edu/ml/datasets/clickstream+data+for+online+shopping). The dataset contains information on clickstream from online store offering clothing for pregnant women [46]. The data set contains 14 features, one of them (Price) was used as the binary label for the nested classification task and the rest were used as the objective data to be replicated by the GANs.
3. CTU-13 data set (https://www.stratosphereips.org/datasets-ctu13). The CTU-13 is a dataset of botnet traffic that was captured in the CTU University, Czech Republic, in 2011 [47]. The dataset contains a large capture of real botnet traffic mixed with normal traffic and background traffic. The CTU-13 dataset consists of 13 captures (called scenarios) of different botnet samples that were processed to obtain a single data set with 11 features plus a label (1: botnet attack, 0: normal traffic).

5.3. Proposed architecture

Aiming to mimic synthetic data with several types of behaviour, we adopted in preliminary experiments a well-known conditional GAN model, the so-called Auxiliary Classifier GAN (AC-GAN) [48], as the architecture to generate at the same time all types of variables. In both use cases, the ACGAN did not produce an adequate performance when replicating the two types of data and moreover, it generated significant oscillations in the convergence process. For that reason, we opted to use a different GAN for each type of data to be generated.

To get rid of the mode collapse problems that frequently appear during GAN training, we adopted as a reference model the WGAN architecture [28], in which a Wasserstein loss function is used as the loss function instead of a standard cross-entropy function. We tested two different strategies to enforce the required Lipschitz constraint in the cost function, weight clipping ([28]) and gradient penalty ([49]), not observing any significant enhancement in the convergence of the GAN training. Therefore, we chose a WGAN architecture with no additional strategy to enforce the Lipschitz constraint and with a discriminator with small learning rates as a heuristic to avoid reaching mode collapse situations.

For the shake of brevity we will use the terms *standard GAN* or *vanilla GAN* in the rest of the paper to refer to a state-of-the-art WGAN equipped with linear activation functions in the last layer of the generator network. Additionally, we will use *ST-based GAN* to refer to a WGAN in which the linear activation functions of the last layer of the generator network have been substituted by the corresponding Smirnov transform functions.

Regarding that the statistical nature of the 4 features to be synthetically replicated in both use cases did not exhibit any topological structure or time relationship among them, convolutional or recurrent networks would not take any advantage of it. Therefore, we selected fully connected neural networks (FCNNs) as the architectural model for both the discriminant and generator networks. LeakyRelu functions were used as the activation function in all layers, except for the last layer of the generator. Based on previous experiments [20], no filtering based on the output of the discriminator was applied to discard synthetic data, nor was noise added to synthetic or real data during the training of the discriminator to help GAN convergence.

5.4. Experimental results

In this section, we analyze the performance on each of the use cases of Section 5.1 of four different generative approaches: (i) when real data is used and no generation occurs, (ii) with a simple mean-based generator, (iii) with a standard WGAN, and (iv) with a WGAN with ST-based activation function. As we will show, the ST-based solution outperforms both the standard GAN and the simple mean-based generator, reaching an accuracy in a nested ML classifier similar to the one obtained with real data.

As ML classifier, we selected a Random Forest model with 300 estimators. In the first use case, we limited each tree depth to 20 to avoid some overfitting effects that appeared in preliminary experiments. No depth limit was applied to trees in the second use case. In the first use case, a balanced set of samples were obtained for each label (200,000 samples per label

totalling 400,000 samples) for training and testing. In the second use case, we kept the original ratio of the two labels (many more normal traffic connections than cryptomining ones) and we got 400,000 samples of label 0 (normal traffic) and only 4,000 of label 1 (cryptomining connections) for training and testing. In the testing process, we establish additional decision thresholds of 0.2, 0.4, 0.6 and 0.8 to the default 0.5 in order to analyse the results of the default and the best performing threshold. Finally, the training and testing of ML classifiers were run 100 times in all experiments to minimise biased behaviours during sampling and training.

For training WGANs, we used previous knowledge from past experiments [20], and applied the set of hyperparameters detailed in Table 2 performing a blind random search in the hyperparameter space guided by the F_1-score obtained in a nested ML-model that was executed evaluating the marginal quality of the generator after 10 mini-batch trains (see Section 4.2). For each type of data, the WGAN selected was the one that obtained the best F_1-score for the nested classifier in any of its mini-batches. As optimization algorithms, we used Adam for generators and RMSProp for discriminators and the binary cross-entropy loss function was substituted by the Wasserstein loss. The hyper-parameters chosen for the generator and discriminator in each use case are detailed in the last two columns in Table 2.

5.4.1. Real data

To establish an upper bound on the expected performance of the nested ML classifier, a benchmark classifier was trained 100 times for each of the two use cases with real samples from the first data set DS1 and tested using samples from the second data set DS2. The first row in Table 3 summarises the obtained F_1-score values and confusion matrices in testing for the best decision threshold and the default threshold (0.5) in the first use case (rendered data). The first row of Table 4 summarises the same information for the second use case (cryptomining attack).

Alongside, Fig. 13a plots the histogram of the statistical distribution of F_1-score results obtained in the first use case when a ML classifier was trained with DS1-r and tested with DS2-r. Similarly, Fig. 15a shows the histogram of the statistical distribution of F_1-score results obtained in the second use case when a ML classifier was trained with DS1-c and tested with DS2-c.

5.4.2. Mean generator

Opposed to the previous results, we established a baseline in our experiments through a naïve generator designed to generate new data by adding Gaussian noise to the means of the data features. The standard deviation of the noise was manually adjusted to produce the best results and hence, a more challenging baseline. Each nïaive model was tested with the corresponding DS2 data set of each use case.

A summary of the baseline results obtained with this naïve model can be found in the second rows of Table 3 and Table 4, for the first and second use cases respectively. Histograms showing the F_1-score values obtained after running the mean generators 100 times are shown for each use case in Fig. 13b and Fig. 15b.

Table 2
WGAN hyperparameters.

		Range of values	Rendered data	Network data
Generator	*# layers*	[2..6]	$G_0, G_1 \in$	$G_0 \in [200, 500, 3000, 500, 4]$
	# units per layer	[100..10000]	$[500, 3000, 5000, 400, 4]$	$G_1 \in [600, 3000, 1000, 4]$
	latent vector	Fixed value (100, 123)	100	123
	noise for latent vector (distribution, std)	distr = [norm, uniform] std = [0.1..100]	uniform, std = 1.5	normal, std = 0.5
	batch normalization	[True..False]	True	True
	regularization:	L2 = $[1e-5..10]$	L2: $G_0, G_1 = 0.1$	L2: $G_0, G_1 = 0$
	L2, dropout	Dropout = [0..1]	Dropout: $G_0, G_1 = 0$	Dropout: $G_0, G_1 = 0$
	LeakyRelu alpha	Fixed value (0.15)	$G_0 = G_1 = 0.2$ $D_0 = D_1 = 0.15$	$G_0 = G_1 = 0.2$ $D_0 = D_1 = 0.15$
Discriminator	learning rate	Default value (0.001)	$G_0, G_1 = 0.001$	$G_0, G_1 = 0.001$
	# layers	[2..6]	$D_0, D_1 \in$	$D_0 \in [380, 800, 600, 177, 23]$
	# units per layer	[100..10000]	$[280, 503, 177, 23]$	$D_1 \in [280, 903, 500, 23]$
	batch normalization	[True..False]	True	True
	regularization:	L2 = $[1e-5..10]$	L2: $D_0, D_1 = 0.001$	L2: $D_0 = 0.02$, $D_1 = 0.05$
	L2, dropout	Dropout = [0..1]	Dropout: $D_0, D_1 = 0$	Dropout: $D_0 = 0.1$, $D_1 = 0.15$
	LeakyRelu alpha	Fixed value (0.15)	$D_0 = D_1 = 0.15$	$D_0 = D_1 = 0.15$
	learning rate	Default value(0.001)	$D_0, D_1 = 0.0001$	$D_0, D_1 = 0.001$

Table 3
Rendered data set (first use case). Performance of synthetic traffic combining labels 0 and 1. Training with (i) a real data set (R-DS1), (ii) a mean-based noise generator and (iii and iv) GAN synthetic data (with linear and ST activation). Results on testing with real data (R-DS2). Experiment is drawn 100 times uniformly at random (Fig. 13 and Fig. 14). For the GANs, in each sample we choose one generator from among all label 0 generators and one generator from among all label 1 generators.

Dataset	Quality Measure	Best		Default	
Training 200K/200K	*Threshold*	0.5		0.5	
Real data (R-DS1)	F_1*-score*	0.812		0.812	
	Confusion	349534	50466	349534	50466
	matrix	99271	300729	99271	300729
Training 200K/200K	*Threshold*	0.2		0.5	
Noise generator with means	F_1*-score*	0.764		0.731	
	Confusion	349292	50708	379238	20762
	matrix	135673	264327	185058	214942
Training 200K/200K	*Threshold*	0.5		0.5	
WGAN with linear activation	F_1*-score*	0.789		0.789	
	Confusion	357013	42987	357013	42987
	matrix	123978	276022	123978	276022
Training 200K/200K	*Threshold*	0.4		0.5	
WGAN with ST activation	F_1*-score*	0.795		0.793	
	Confusion	338960	61040	333288	66712
	matrix	102240	297760	98781	301219
Training 200K/200K	*Threshold*	0.4		0.5	
WGAN with linear activation	F_1*-score*	0.779		0.775	
F_1-score elitism (top 10)	*Confusion*	346911	53089	339251	60749
	matrix	122176	277824	118428	281572
Training 200K/200K	*Threshold*	0.5		0.5	
WGAN with ST activation	F_1*-score*	0.783		0.783	
F_1-score elitism (top 10)	*Confusion*	339440	60560	339440	60560
	matrix	111704	288296	111704	288296

Table 4
Performance of synthetic traffic combining labels 0 and 1. Training with (i) a real data set (DS1), (ii) a mean-based noise generator and (iii and iv) GAN synthetic data (with linear and ST activation). Results on testing with real data (DS2). Experiment is drawn 100 times uniformly at random (Fig. 15 and Fig. 16). For the GANs, in each sample we choose one generator from among all label 0 generators and one generator from among all label 1 generators.

Dataset	Quality Measure	Best		Default	
Training 400K/4K	*Threshold*	0.8		0.5	
Real data (DS1)	F_1*-score*	0.957		0.898	
	Confusion	399426	574	399873	127
	matrix	205	4183	1384	3004
Training 400K/4K	*Threshold*	0.8		0.5	
Noise generator with means	F_1*-score*	0.710		0.624	
	Confusion	393386	6614	381851	18149
	matrix	1364	3024	816	3572
Training 400K/4K	*Threshold*	0.8		0.5	
WGAN with linear activation	F_1*-score*	0.920		0.820	
	Confusion	399692	308	398651	1349
	matrix	969	3419	1657	2731
Training 400K/4K	*Threshold*	0.8		0.5	
WGAN with ST activation	F_1*-score*	0.897		0.869	
	Confusion	399780	220	399123	877
	matrix	1334	3054	1288	3100
Training 400K/4K	*Threshold*	0.8		0.5	
WGAN with linear activation	F_1*-score*	0.941		0.933	
F_1-score elitism (top 10)	*Confusion*	399646	354	399696	304
	matrix	626	3762	782	3606
Training 400K/4K	*Threshold*	0.6		0.5	
WGAN with ST activation	F_1*-score*	0.879		0.873	
F_1-score elitism (top 10)	*Confusion*	399572	428	399261	739
	matrix	1425	2963	1316	3072

5.4.3. GANs: Linear and custom activation

For the two use cases of Section 5.1, we ran a set of experiments to obtain the performance of a ML classifier trained with GAN synthetic data and tested with DS2 data sets (DS2-r for the first use case and DS2-c for the second). GANs were always trained using DS1 data sets (DS1-r for the first use case and DS1-c for the second). The GANs were trained during a fixed set of $25,000$ mini-batches (50 epochs). Every 10 mini-batches, the GAN generator model was saved for posterior use, and the obtained L^1 and Jaccard distances of synthetic and real data were computed. In addition, as mentioned in Section 4.2, a training data set was generated mixing GAN synthetic data of the current label with real data sampled from the other label of DS1. Using this hybrid data set, we trained a ML classifier, and then the ML model was tested with DS2. The obtained F_1-score represents the marginal performance of the GAN synthetic data for the current label and, in addition, provides a potential early stopping criterion for GAN training. Note that although both DS1 and DS2 contain real data, the performance against DS2 provides a more reliable measure, since DS1 was used for training the GAN and therefore, the GAN generator could have learnt specific information only contained in DS1.

After training standard and ST-based WGANs for the two labels, we run the following experiment for each type of GAN (standard and ST-based) to highlight the advantage of the proposed ST solution: For each label, a WGAN generator is selected uniformly at random among all models stored previously every 10 mini-batches. Then, a completely synthetic data set is produced using the generator of label 0 and the generator of label 1. Using this synthetic data set we train the ML classifier and test its performance with DS2 obtaining the F_1-score value and the confusion matrix. This process was run 100 times to compare the statistical distribution of the obtained F_1-score values for the standard and ST-based WGANs. In addition, we repeated the experiment not selecting uniformly at random each label generator among the whole set of stored generators for a label but among the top 10 sorted by the marginal F_1-score for this label (i.e., using F_1-score elitism). In this way, we explored whether it is more efficient to search the best performing label 0 and 1 generators among all stored generators or using the F_1-score elitism. In addition, we analyse whether the ST-based solution performs better than the standard WGAN using this elitism.

Distances from synthetic to real data. Fig. 1 shows the evolution of L_1 distance and Jaccard index during the GAN training for label 0 and label 1 in the first use case. It can be observed from both labels that the L_1 distance curve for the ST-based WGAN stabilises faster, shows less oscillations and achieves smaller values (around 0.3 for both labels when GAN training is stabilised) than in the standard WGAN with linear activation (around 0.6 for both labels in the minimum points of the curves). With respect to the Jaccard index, the results of the ST-based WGAN conclude in a similar way: The curves for both labels achieve high values (around 0.7 for label 0 and 0.9 for label 1), stabilise faster (in 4 epochs for label 0 and 6 epochs for label 1) and do not exhibit significant oscillations. On the contrary, the Jaccard curves of the standard WGAN for the two labels show a bad performance with values not greater than 0.4. These results highlight that in the first use case the similarity of the synthetic data generated by the ST-based WGAN generator and the real data is much higher than when the synthetic data is generated by the standard WGAN generator.

The L^1 distance and Jaccard index results for the second use case are shown in Fig. 2. Similarly to the first use case, the distance in ST-based WGAN stabilises quickly (10 epochs for label 0 and 20 for label 1), with a very low value (around 0.25) and without significant oscillations indicating that the quality of the generated synthetic data is high. On the contrary, stan-

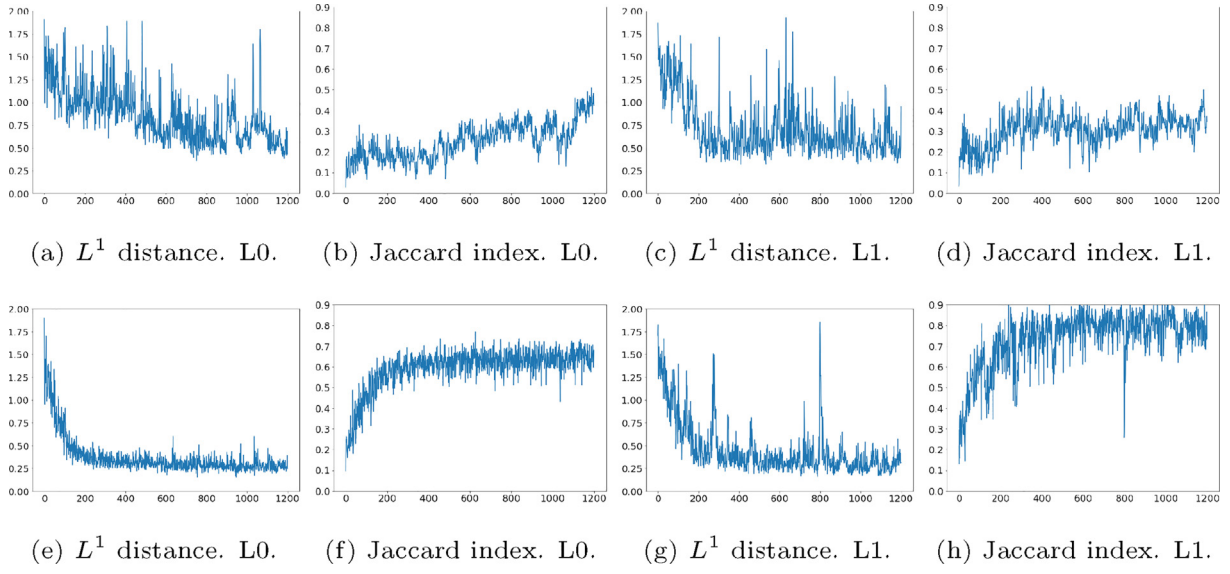


Fig. 1. Rendered data (use case #1). Evolution of L^1 distance and Jaccard index, using GAN generators with linear activation (top row) and ST-based activation (bottom row) for labels 0 and 1. The x-axis represents the GAN training epochs (1 epoch = 50 ticks).

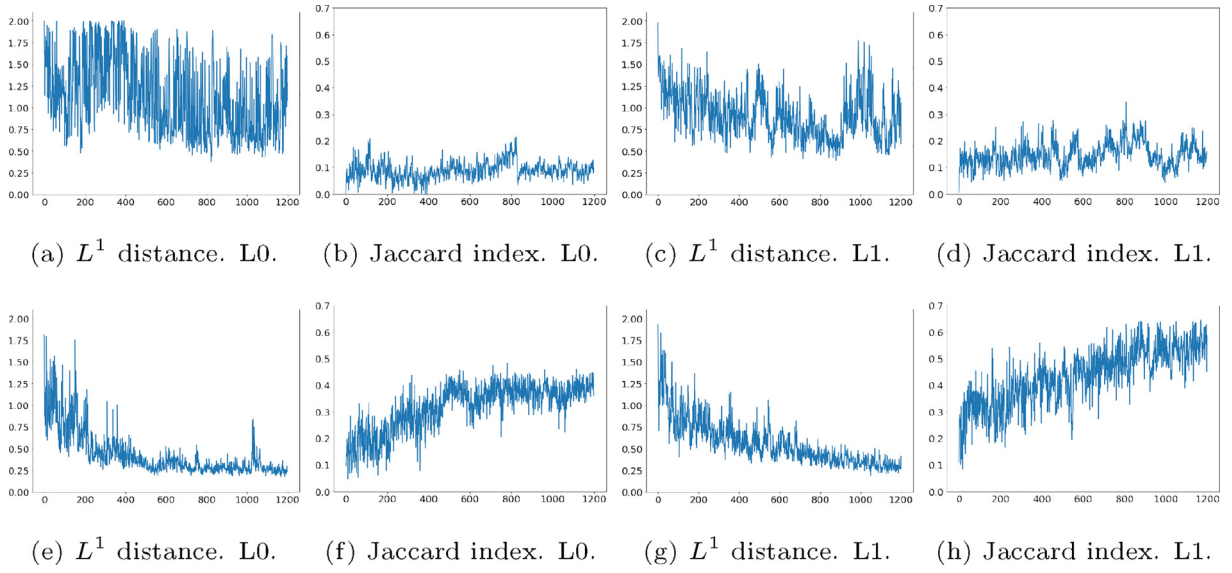


Fig. 2. Cryptomining attack scenario (use case #2). Evolution of L^1 distance and Jaccard index, using GAN generators with linear activation (top row) and ST-based activation (bottom row) for labels 0 (normal traffic) and 1 (cryptomining connections). The *x*-axis represents the GAN training epochs (1 epoch = 50 ticks).

dard WGAN suffers from remarkable oscillations and the distance value is not small (from 0.65 to 1.6 for label 0 and from 0.6 to 1.25 for label 1), which highlights that the similarity of the synthetic data generated by the GAN and the real data is not very high. Analyzing the Jaccard coefficient in the figure for the ST-based WGAN curve, it can be seen that values around 0.4 and 0.6 are obtained for labels 0 and 1. It is intuited in the figure that with more training epochs the former values would continue to grow. In contrast, the standard WGAN curve quickly stabilises around a very small value of 0.1 for both labels, which shows that in this case, the statistical distributions of real and synthetic data are quite different.

We can conclude that in both use cases, the ST-based WGAN replicates for the two labels the statistical behaviour of the real data (DS1 data set) with high precision and requiring only a few training epochs. In addition, the quality of the synthetic data produced throughout the training process does not suffer from significant oscillations. On the contrary, the standard WGAN replicates with worse quality the statistical distribution of the real data, needs more training epochs, and the quality of the generated data suffers from high oscillations during the training process, which prevents its use in real applications.

Evolution of synthetic data quality. In this section, we analyze the evolution of the quality of the synthetic data generated by the standard and ST-based WGANs with respect to the real data. Figs. 3 and 4 for the first use case, and Figs. 5 and 6 for the second, show graphically the obtained distributions. To this end, we compare and plot samples of real and synthetic data distributions at different mini-batches (1, 100, and 1000) corresponding to the epochs 0, 2 and 20 respectively.

To ease the visualization of the above-mentioned figures, we have flattened the four histograms corresponding to the four features of the data set into a 1-dimensional plot. To be precise, each of the points in the *x*-axis of the plot represents a cube (a bin) in which the empirical probability density function was computed (see Section 4.1). In this way, comparing the cubes of two samples, we can infer whether the two data distributions are similar or not. For example, samples of two data distributions with elements placed in different cubes would point to data distributions with significant differences. On the contrary, if the elements of both distributions are mapped to the same cubes and the number of elements in each cube is similar for both samples, we could infer that the two data distributions are similar.

To plot and compare the synthetic and real data distributions, the cubes of the real data sample are sorted by the number of elements in each cube in ascending order. In this way, the marks on the *x*-axis represent the cubes as ordered for the real data sample. Therefore, the real data curve always exhibits an ascending shape. The *y*-axis represents the number of sample elements that are placed in each cube. It is worth noting that higher numbers on the *x*-axis indicate that the WGAN has created many nonexistent elements (in the real data set) that are assigned to new bins. These bins containing elements outside the real data domain are placed on the left side of the WGAN curve since the real data curve has no elements in such bins.

In the first use case, it can be observed in Fig. 3 that the standard WGAN for label 0 creates many more nonexistent data elements than the ST-based WGAN. Considering that the real data samples generate around 80 cubes, at epoch 2, the standard WGAN created 350 cubes and the ST-based WGAN only 120, and when they reach epoch 20, the standard WGAN generated 200 cubes and the ST-based WGAN remained at 120. A similar behaviour is observed in Fig. 4 for label 1, where the standard WGAN, in the three epochs discussed above, doubles the number of cubes generated by the ST-WGAN. The situation in the second use case is similar to the first. The ST-based WGAN generates significantly fewer cubes than the standard WGAN for both labels (Fig. 5 and Fig. 6).

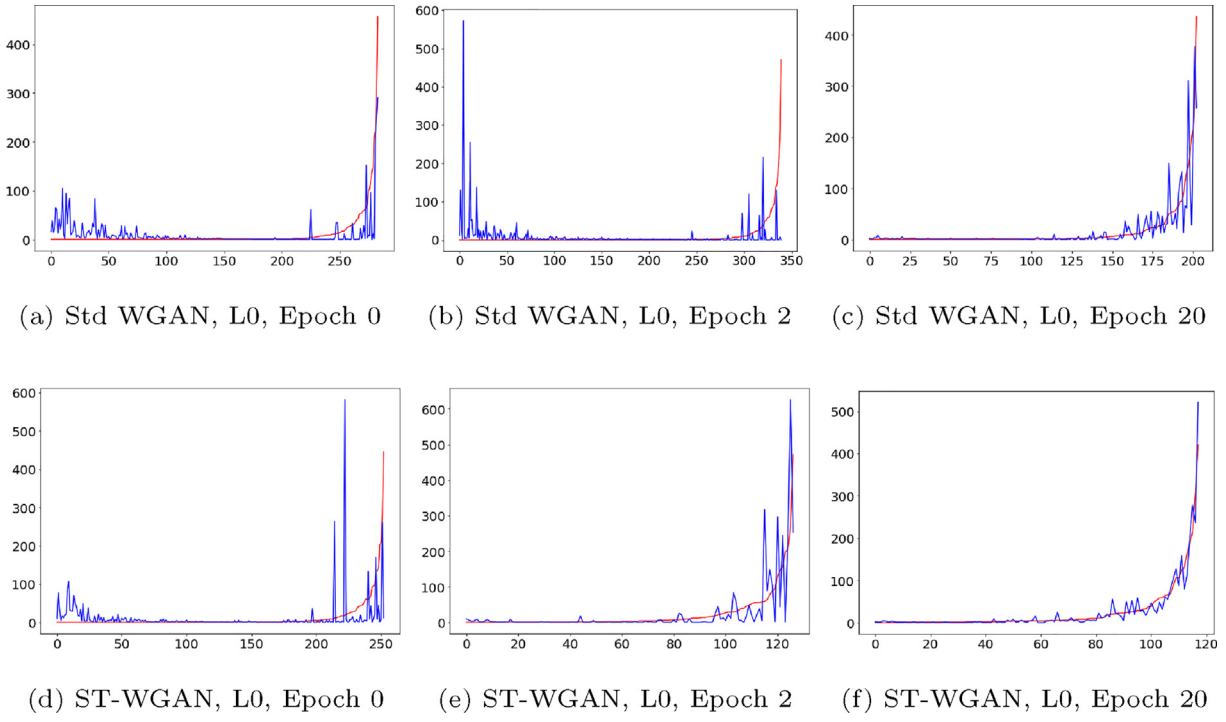


Fig. 3. Rendered data set (first use case). Comparison of synthetic (blue) and real (red) data distributions using WGAN generators for label 0 with linear activation (top row) (3a, 3b and 3c) and with ST-based activation (bottom row) (3d, 3e, and 3f) in different epoch.s (1, 2 and 20).

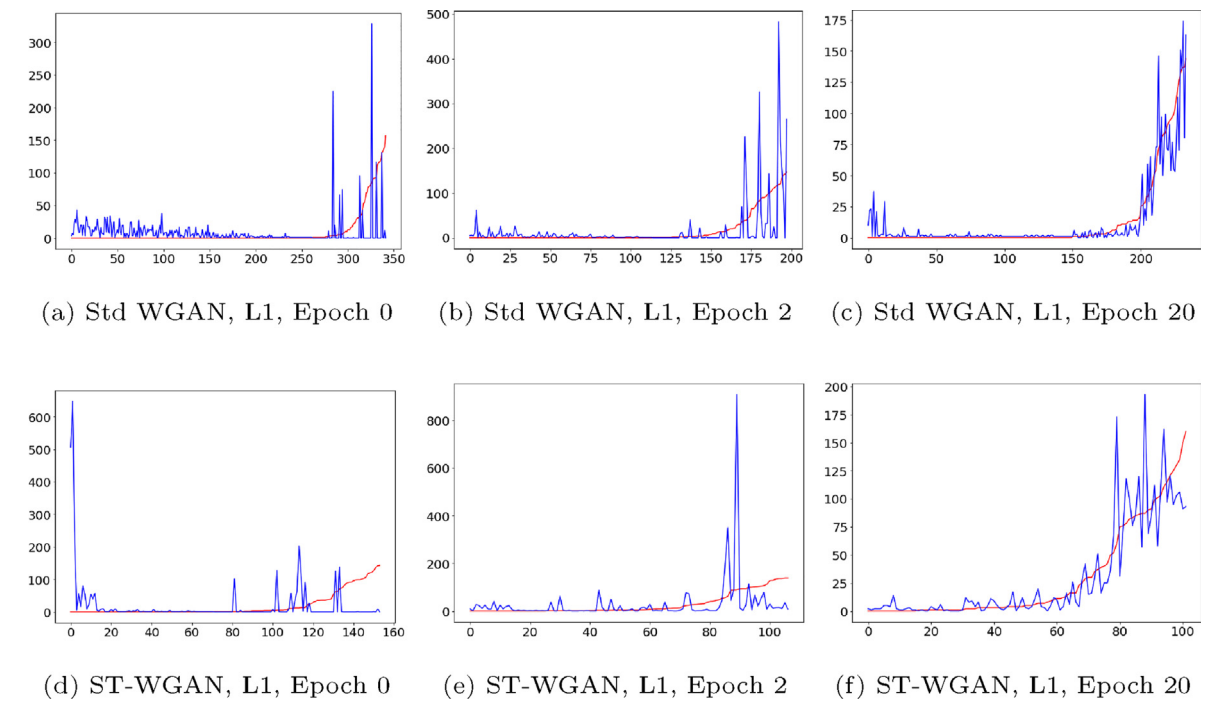
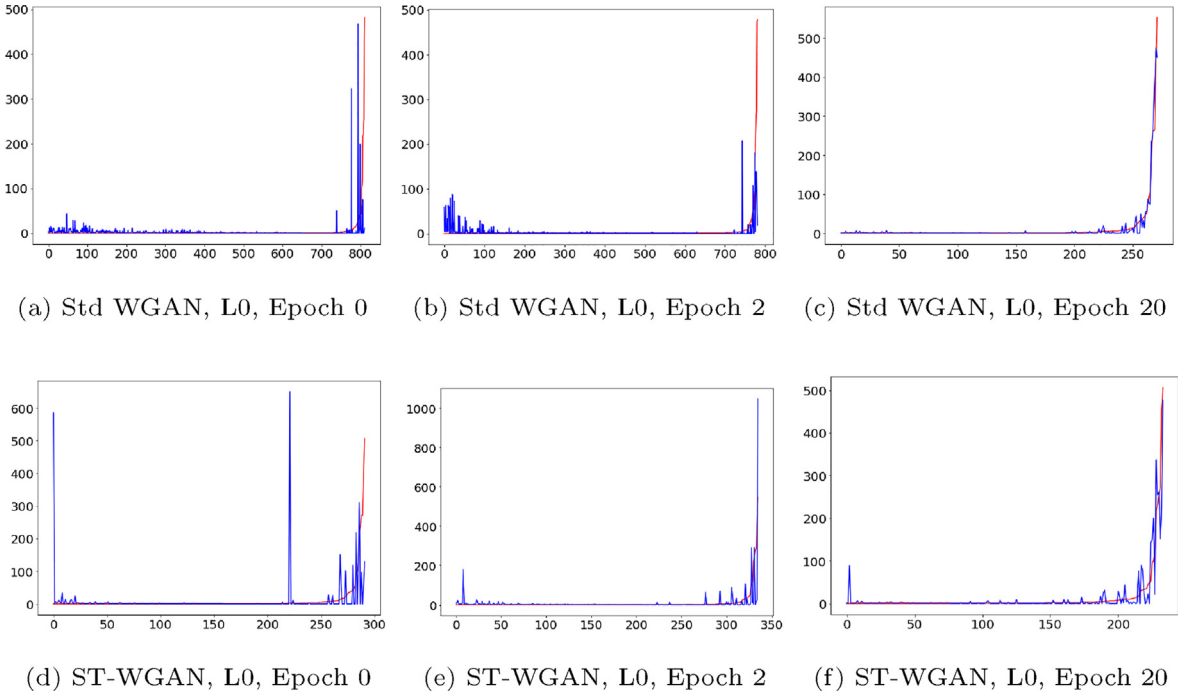


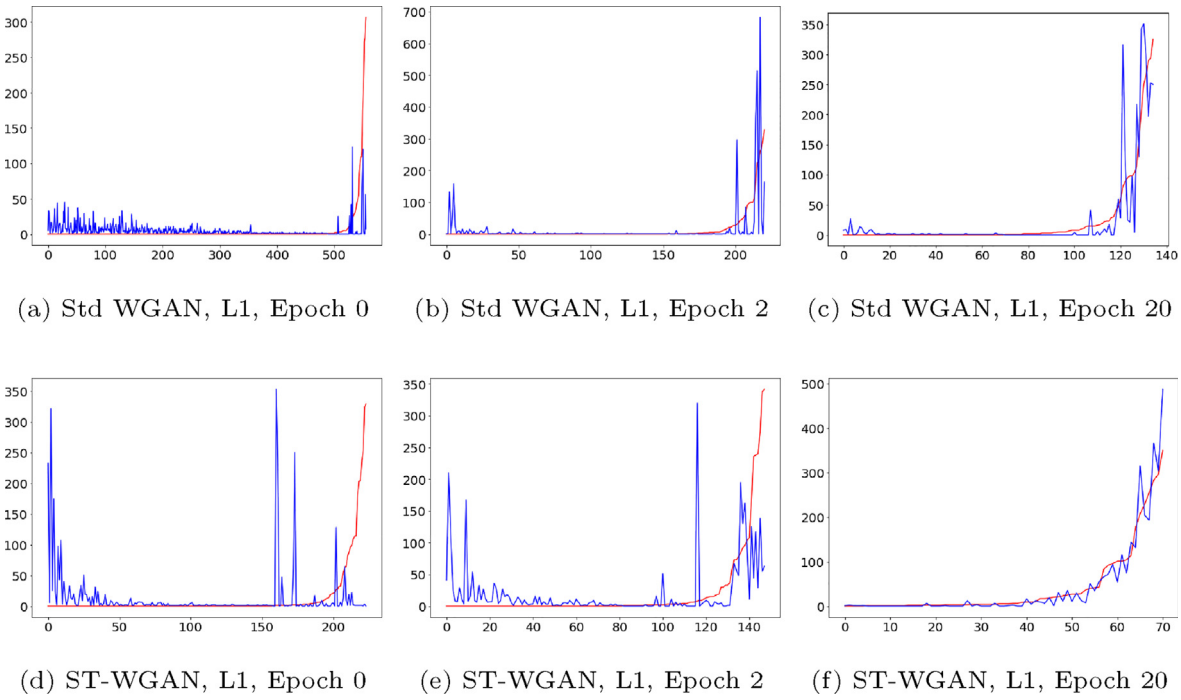
Fig. 4. Rendered data set (first use case). Comparison of synthetic (blue) and real (red) data distributions using WGAN generators for label 1 with linear activation (top row) (4a, 4b, and 4c) and with ST-based activation (bottom row) (4d, 4e, and 4f) in different epochs (0, 2 and 20). The 4-dimensional vector has been flattened into by sorting by frequency in ascending order on t.he *x*-axis.



(a) Std WGAN, L0, Epoch 0 (b) Std WGAN, L0, Epoch 2 (c) Std WGAN, L0, Epoch 20

(d) ST-WGAN, L0, Epoch 0 (e) ST-WGAN, L0, Epoch 2 (f) ST-WGAN, L0, Epoch 20

Fig. 5. Cryptomining attack (second use case). Comparison of synthetic (blue) and real (red) data distributions using GAN generators for label 0 with linear activation (top row) (5a, 5b and 5c) and with ST-based activation (bottom row) (5d, 5e and 5f) in different epoch.s (0, 2 and 20).



(a) Std WGAN, L1, Epoch 0 (b) Std WGAN, L1, Epoch 2 (c) Std WGAN, L1, Epoch 20

(d) ST-WGAN, L1, Epoch 0 (e) ST-WGAN, L1, Epoch 2 (f) ST-WGAN, L1, Epoch 20

Fig. 6. Cryptomining attack (second use case). Comparison of synthetic (blue) and real (red) data distributions using GAN generators for label 1 with linear activation (top row) (6a, 6b, and 6c) and with ST-based activation (bottom row) (6d, 6e, and 6f) in different epochs (0, 2 and 20).

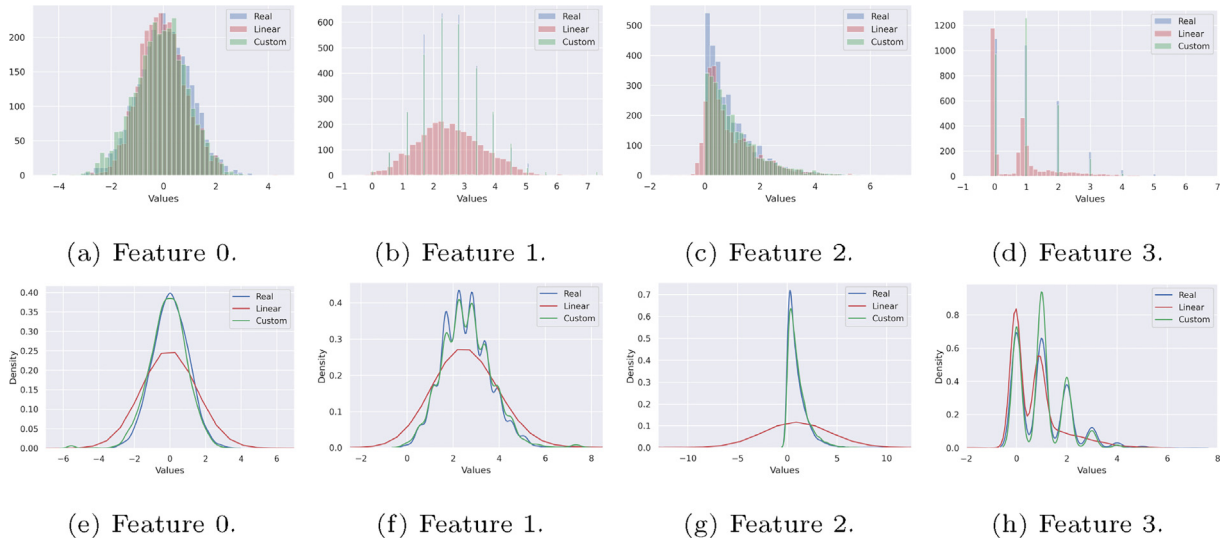


Fig. 7. Rendered data set (first use case). Frequency distribution from Label 0 of real data and standard and ST-based WGAN's synthetic data. Histogram (top row) and Kernel Density Estimator (KDE) function (bottom row) are shown for the four variables.

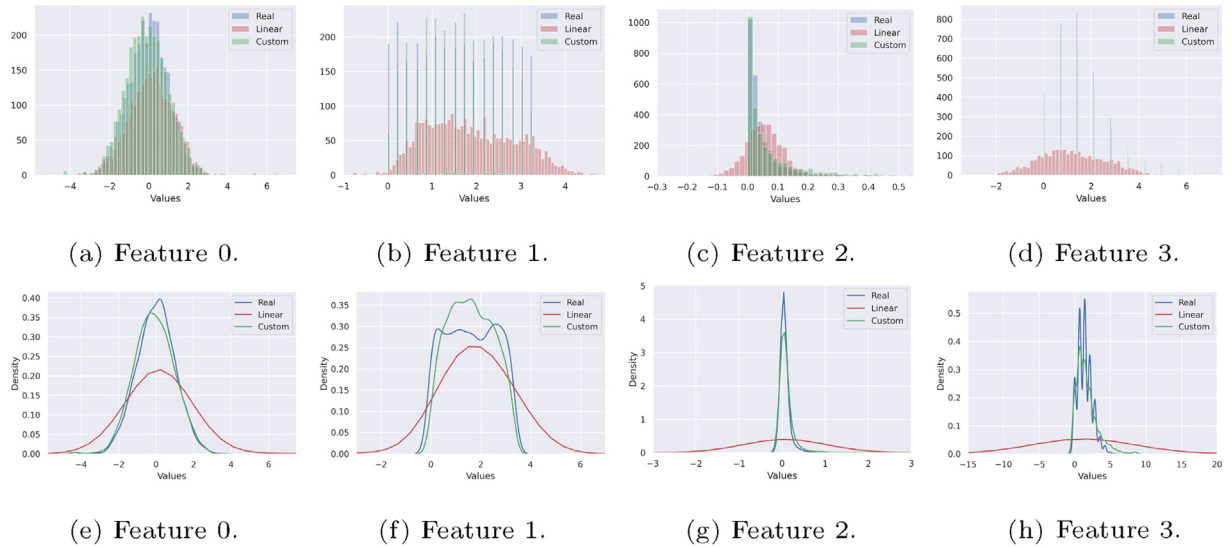


Fig. 8. Rendered data set (first use case). Frequency distribution from Label 1 of real data and standard and ST-based WGAN's synthetic data. Histogram (top row) and Kernel Density Estimator (KDE) function (bottom row) are shown for the four variables.

Feature replication. After having analysed in the previous subsections the quality of the synthetic data generated by ST-based WGANs, we study the fidelity with which WGAN generators replicate the statistical distribution of each variable. We compare the statistical distributions of each of the 4 variables that compose the data elements in the first use case in Fig. 7 (label 0) and Fig. 8 (label 1), and for the second use case in Fig. 9 (label 0) and Fig. 10 (label 1). At the top row of each figure, we plot the histogram of the real variables and the synthetic variables generated by the standard and ST-based WGANs. The bottom row shows the Kernel Density Estimator (KDE [50]) function of all of them. Synthetic data samples were obtained at epoch 25.

In general, it can be observed in all figures that the ST-based WGAN replicates the histogram and the KDE function of each variable with high accuracy. On the contrary, for the two use cases and the two labels, the standard WGAN tends not to mimic the statistical distribution of the real data.

It is worth noting that one of the key innovations of our proposal is the ability of ST-based WGANs to replicate discrete data variables. To the best of our knowledge, none of the existing solutions provides a clean approach to solve this crucial

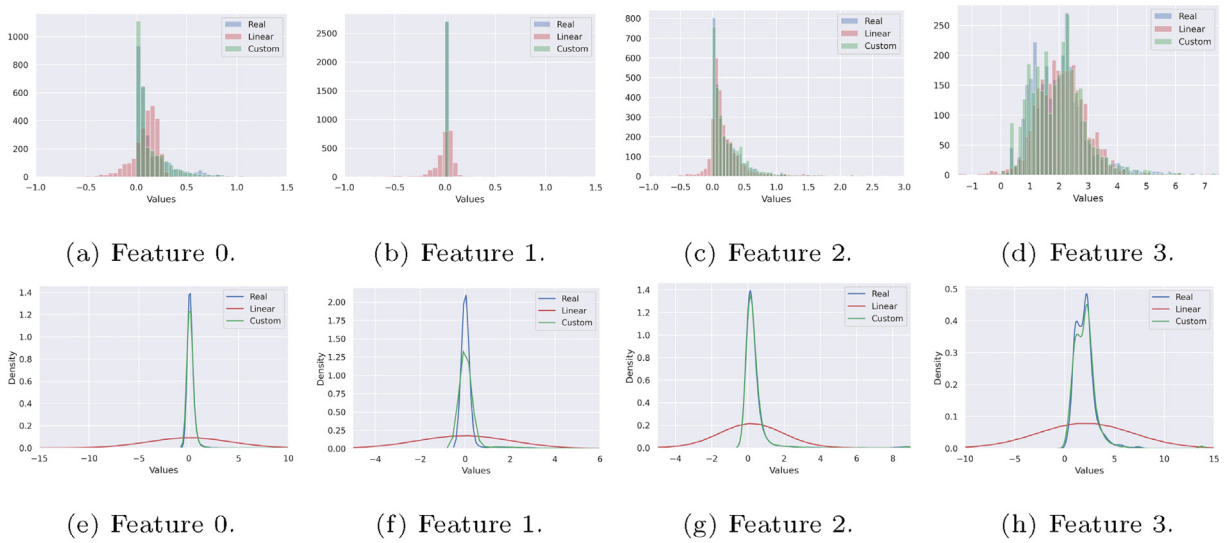


Fig. 9. Cryptomining data set (second use case). Frequency distribution from Label 0 of real data and standard and ST-based WGAN's synthetic data. Histogram (top row) and Kernel Density Estimator (KDE) function (bottom row) are shown for the four variables.

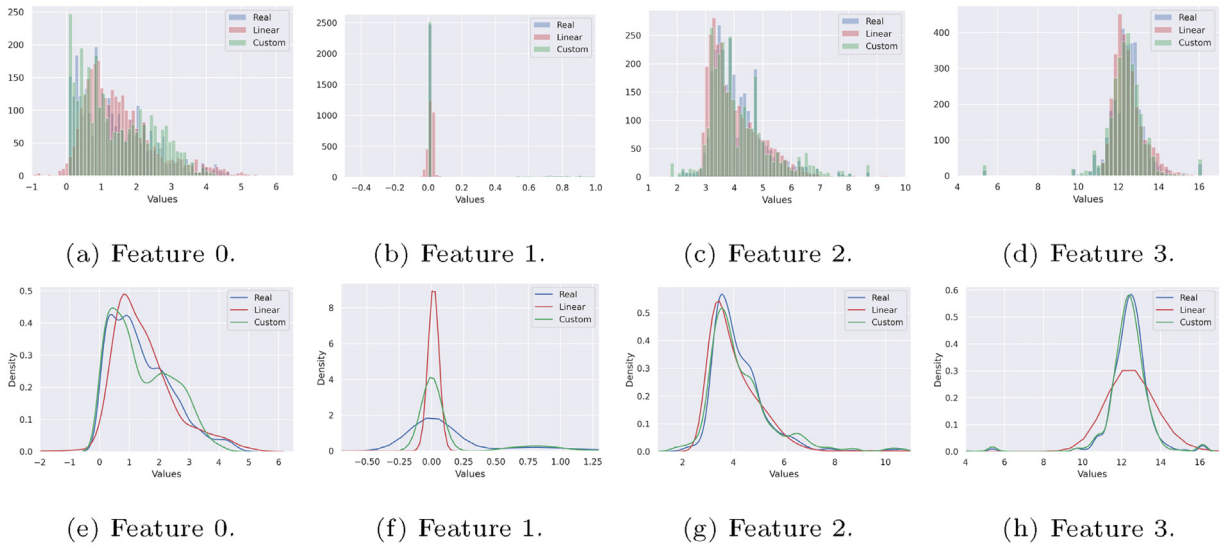


Fig. 10. Cryptomining data set (second use case). Frequency distribution from Label 1 of real data and standard and ST-based WGAN's synthetic data. Histogram (top row) and Kernel Density Estimator (KDE) function (bottom row) are shown for the four variables.

issue as the ST-based approach does. In the first use case, an ad hoc data distribution was designed to contain two discrete variables (features 1 and 3) in each label to highlight the advantage of our solution in comparison with current solutions. It can be observed in the Label 0 histograms for features 1 and 3, (Fig. 7b and Fig. 7d) that the ST-based WGAN perfectly replicates the discrete nature of these features. In sharp contrast, the standard WGAN fails on this task and generates two synthetic variables that follow a continuous distribution. The same situation appears for label 1, as variables 1 and 3 (Fig. 8b and Fig. 8d) are perfectly replicated by the ST-based WGAN and on the contrary, the standard WGAN generates synthetic variables following a continuous distribution.

As a final remark, note that the discrete variables can be categorical or not (e.g., an ordered sequence of integer numbers) as the ST solution does not impose any assumption on this and therefore, provides a general solution for any discrete variable.

Marginal F_1-score. Looking at the marginal F_1-score values obtained for each label in the first use case (Fig. 11), we observe that the standard WGAN generators for both labels fail completely when they are used to marginally replace real data in

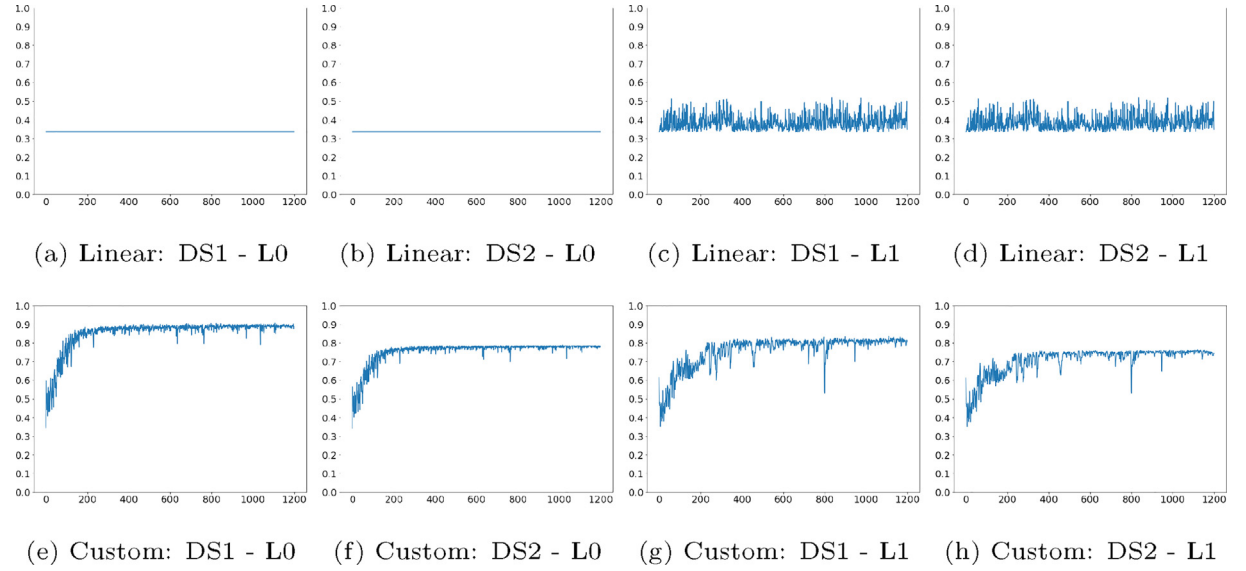


Fig. 11. Rendered data (first use case). Evolution of the marginal F_1-score on training and testing, using GAN generators with linear activation (top row) and ST-based activation (bottom row) for labels 0 and 1. The first and second columns correspond to the evolution for the GAN trained to generate label 0, whereas the third and forth columns correspond to the generation of label 1. The *x*-axis represents the GAN training epochs.

training an ML classifier, as they obtain very bad F_1-score metrics (around 0.35 for label 0 and 0.4 for label 1) in both testing (DS2-r) and training (DS1-r) data sets. On the contrary, the ST-based WGAN obtains F_1-score values close to the ones obtained using real data in the training of the ML classifier. The benchmark classifier trained with real data obtained an F_1-score of 0.812 with DS2-r and using the synthetic data generated by the ST-based WGAN we obtained around 0.75 for label 0 and 0.7 for label 1 with DS2-r as testing data set. Furthermore, the ST-based WGAN did not generate any significant oscillations in the F_1-score curve after the maximum F_1-score values were reached at 4 and 8 epochs respectively for labels 0 and 1.

In general, the results of the marginal F_1-score for the second use case are aligned with those obtained in the first use case. When the ST-based WGAN was used with DS2-c as the testing data set, higher F_1-score values were achieved (around 0.6 for label 0 and 0.85 for label 1) without incurring significant oscillations when GAN training was estabilised (around 8 epochs for both labels). In contrast, the standard WGAN performance was not good, with small F_1-score values for label 0 (from 0.1 to 0.5) and noticeable oscillations throughout the GAN training for both labels. Fig. 12.

We can conclude that in both use cases, the ST-based WGAN obtained better marginal F_1-score values, stabilised to the maximum values faster and without producing significant oscillations once stabilised.

Nested ML evaluation. In this section, we analyze the performance of the nested ML classifier when a fully synthetic data set is generated using GANs. Recall that this data set is created by mixing samples created by the GAN generators of labels 0 and 1, and it is used to feed a ML classifier. As criteria for picking the generators to be used for each label, we applied two different strategies: (i) drawing a random sample among the whole set of stored generators, and (ii) drawing a random sample from the top 10 models sorted by the marginal F1-score of the label (i.e., using F_1-score elitism).

Table 3 summarises the results obtained for the first use case and Fig. 14 shows detailed histograms of the F_1-score results obtained after running 100 times each experiment. In the light of the results presented in these plots, the main conclusions are:

(i) Both the ST-based and standard WGANs can be used for training a ML classifier as their performance is better than that of the noise generator and only slightly worse than that of the real data.
(ii) The synthetic data generated by the ST-based WGAN performs slightly better than that of the standard WGAN, both when random selection is done on the whole set of models (0.793 against 0.789 for the best F_1-score obtained) and when the F_1-score elitism is used (0.783 against 0.775 for the best F_1-score obtained).
(iii) The interval of the F_1-score values obtained when using the ST-based WGAN (from 0.75 to 079) was concentrated near to the maximum value and was significantly smaller than that obtained with the standard WGAN that are noticeably dispersed in a larger interval from 0.5 to 0.79. Hence, selecting at random ST-based generator models is highly likely to obtain a synthetic data set that performs close to the best model combination and the real data. In contrast, if we select generators at random from the standard WGAN, it is less likely to obtain a synthetic data set that comes close to the performance of the real data.

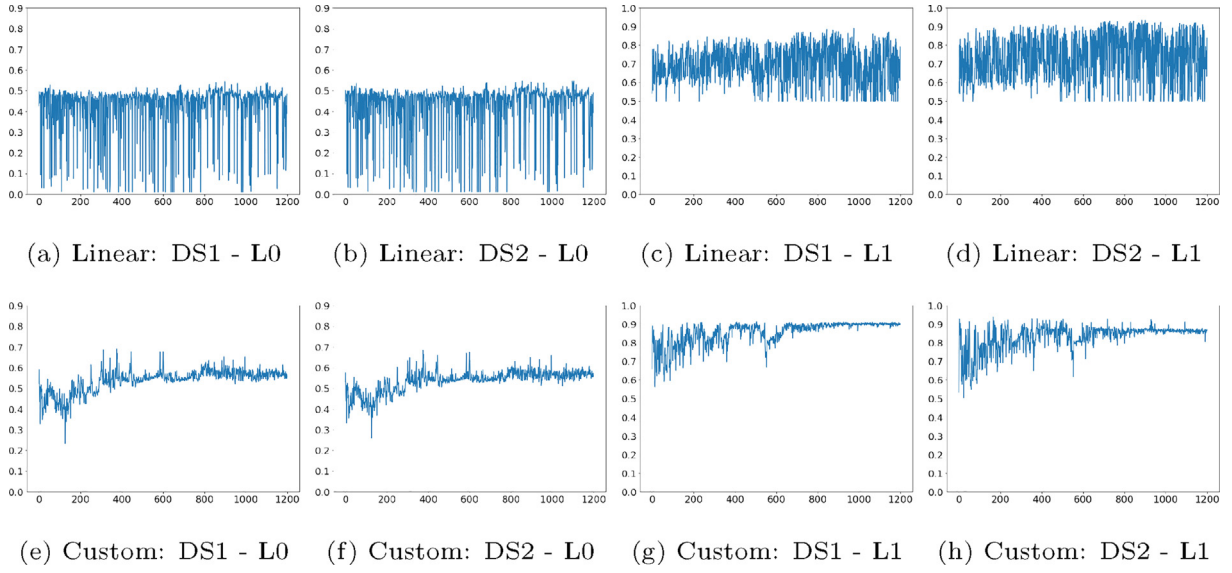


Fig. 12. Cryptomining attack scenario (second use case). Evolution of the marginal F_1-score on training and testing, using GAN generators with linear activation (top row) and ST-based activation (bottom row) for labels 0 and 1. The first and second columns correspond to the evolution for the GAN trained to generate label 0, whereas the third and forth columns correspond to the generation of label 1. The x-axis represents the GAN training epochs.

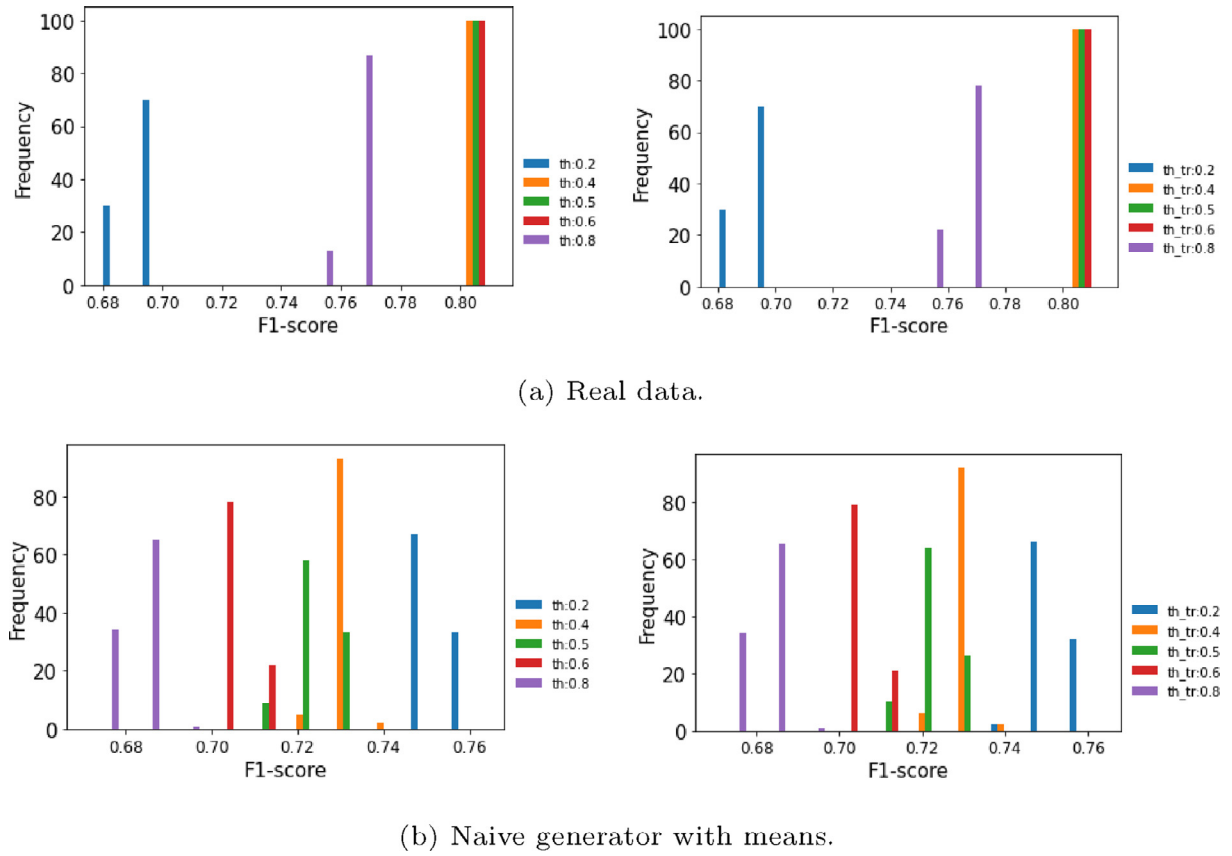
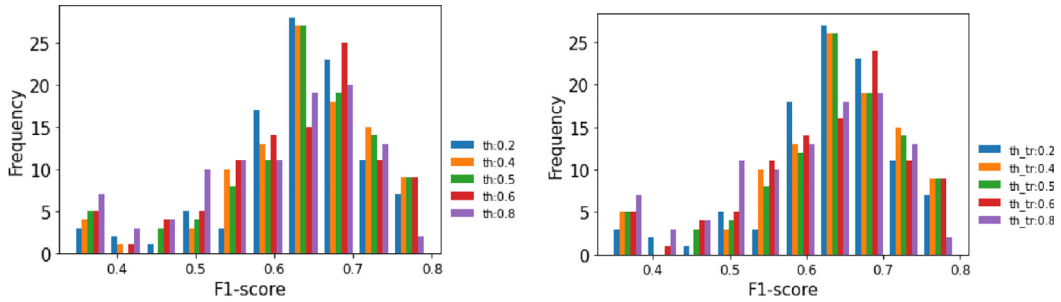
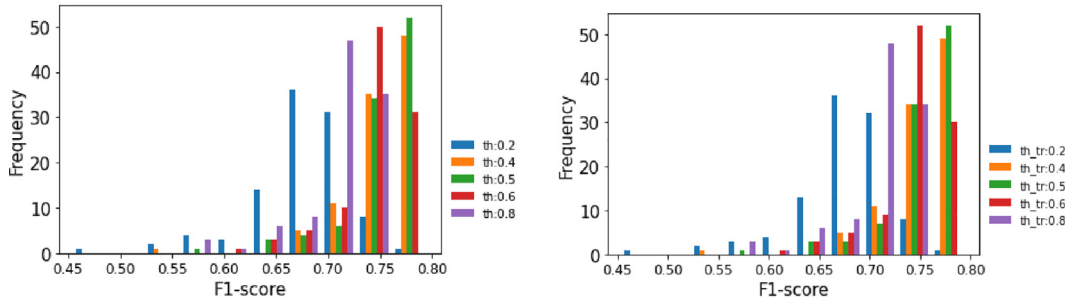


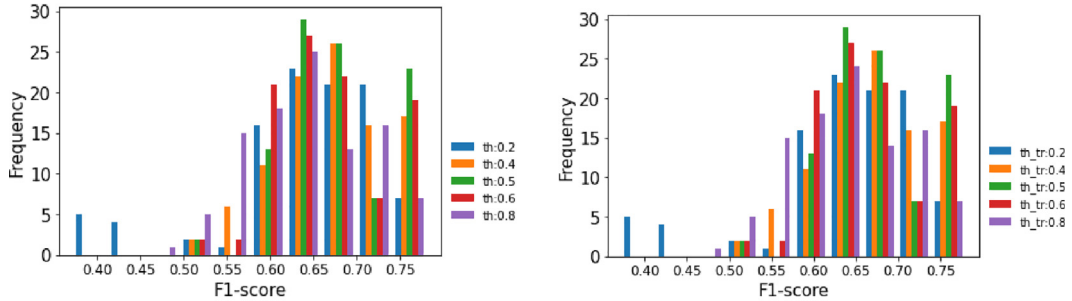
Fig. 13. Rendered data set (first use case). Baseline. F_1-score on DS2-r (left) and DS1-r (right) using as training data set: (13a) a real data set (DS1-r) and (13b) a naive noise generator with means. Results for decision thresholds of 0.2, 0.4, 0.5, 0.6 and 0.8 are represented. Each experiment wa.s run 100 times.



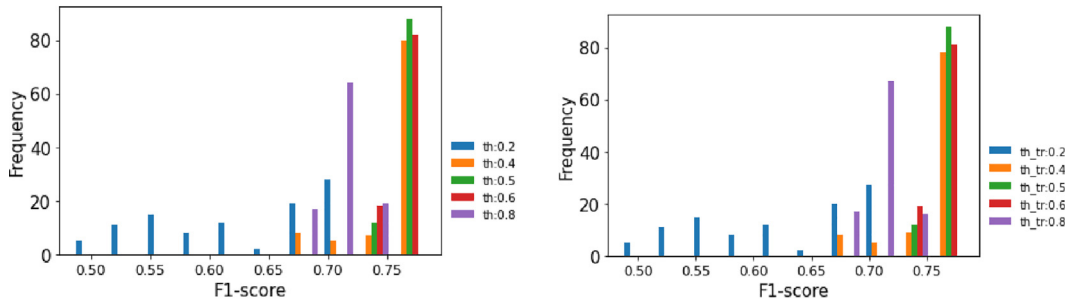
(a) Generator with linear activation.



(b) Generator with custom activation.

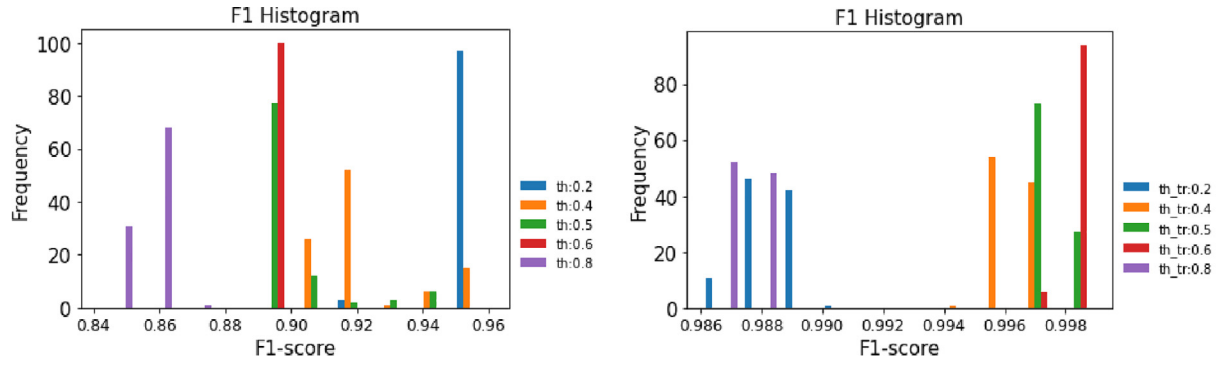


(c) Generator with linear activation. Elitism criterion: Top 10 on F1-score.

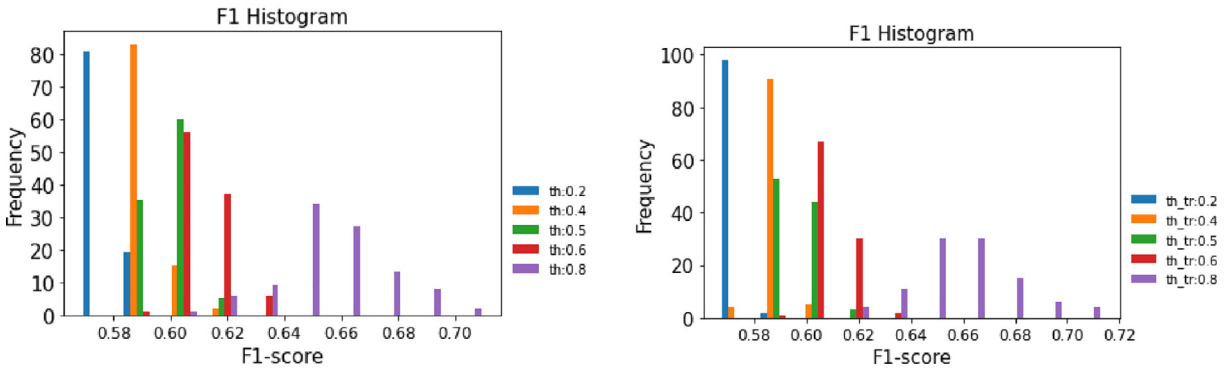


(d) Generator with custom activation. Elitism criterion: Top 10 on F1-score.

Fig. 14. Rendered data set (first use case). F_1-score on DS2-r (left) and DS1-r (right) using as training dataset GAN generators with linear and IST-based activation. Random selection of generators was done from among all generators (14a and 14b) and using an elitism criterion (top 10 of F1-scores) (14c and 14d). Results for decision thresholds of 0.2, 0.4, 0.5, 0.6 and 0.8 are represented. Each experiment wa.s run 100 times.



(a) Real data.



(b) Naive generator with means.

Fig. 15. Cryptomining attack data set (second use case). Baseline. F_1-score on DS2-c (left) and DS1-c (right) using as training data set: (15a) a real data set (DS1-r) and (15b) a naive noise generator with means. Results for decision thresholds of 0.2, 0.4, 0.5, 0.6 and 0.8 are represented. Each experiment wa.s run 100 times.

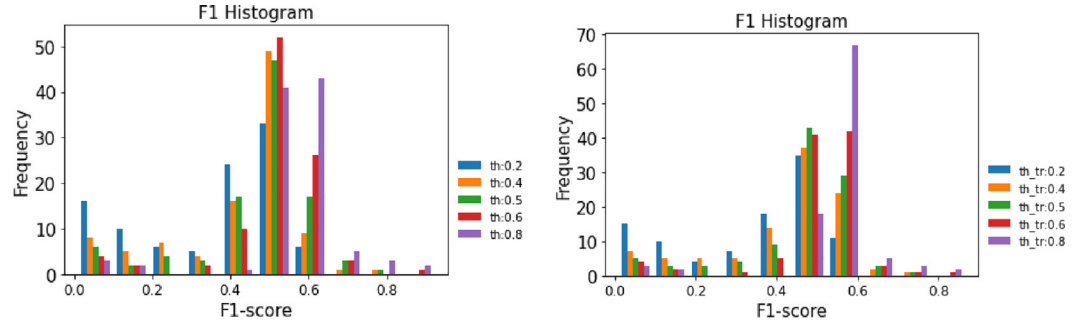
(iv) When the ST-based activation was used, although the maximum value obtained with F_1-score elitism was slightly lower than that obtained with the uniform drawing method, the distribution of results was slightly more concentrated near the maximum value.

Table 4 summarizes the results obtained for the second use case and Fig. 16 shows detailed histograms of the F_1-score results obtained after running each experiment 100 times. In a general sense, the results are aligned with the ones of the first use case:

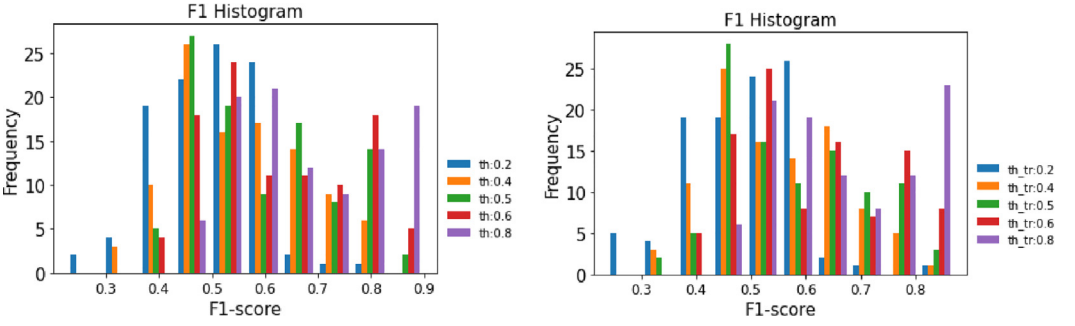
(i) ML classifiers trained with synthetic data generated by WGANs obtain a similar performance than when trained with real data. In fact, the best value of standard WGAN with F_1-elitism was slightly greater than the best value obtained with real data.
(ii) The interval of the F_1-score values obtained when using the ST-based WGAN was concentrated near to the maximum value and this effect was accentuated when F_1-elitism was applied. Hence, selecting at random ST-based generator models is highly likely to obtain a synthetic data set that performs close to the best model combination and the real data.
(iii) In contrast, when using the standard WGAN, the interval of F_1-score values is significantly wider and only a very small percentage of them are close to the best value obtained with real data, which precludes to use this method as it is not likely to produce a realistic synthetic data set when generators are selected at random. However, this effect slightly decreases when F_1-score elitism is used for selecting the generators.

5.5. Further experiments

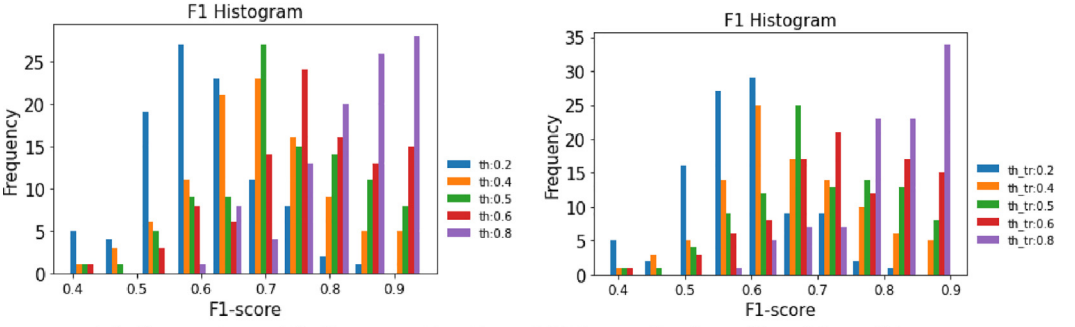
To analyse whether the obtained results can be generalised to other data sets, three publicly available data sets (UCI Census Income, UCI E-shop and CTU-13 botnet attacks) were used to run a new round of experiments as in subSection 5.4.3.



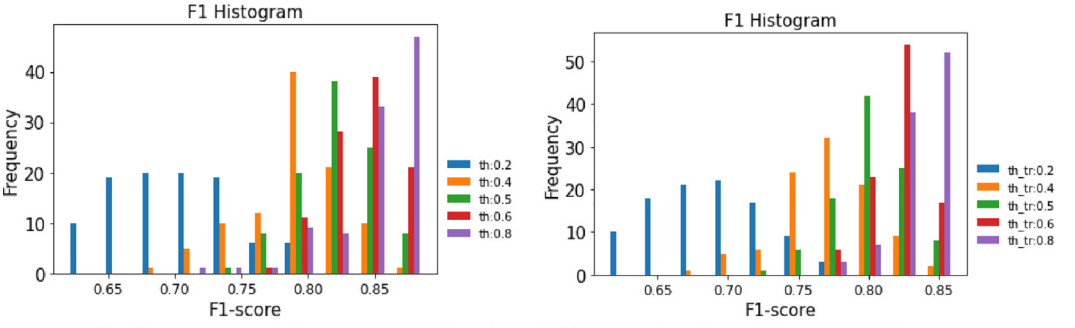
(a) Generator with linear activation.



(b) Generator with custom activation.



(c) Generator with linear activation. Elitism criterion: Top 10 on F1-score.



(d) Generator with custom activation. Elitism criterion: Top 10 on F1-score.

Fig. 16. Cryptomining attack data set (second use case). F_1-score on DS2-c (left) and DS1-c (right) using as training dataset GAN generators with linear and custom activation. Random selection of generators was done from among all generators (16a and 16b) and using an elitism criterion (top 10 of F_1-scores) (16c and 16d). Results for decision thresholds of 0.2, 0.4, 0.5, 0.6 and 0.8 are represented. Each experiment was run 100 times.

Table 5
Additional experiments with publicly available datasets.

Dataset	Quality Measure	Real data	Hybrid dataset : Label 0 (WGAN) Label 1(realdata)	Hybrid dataset : Label 0(realdata) Label 1(WGAN)	Fully synthetic dataset
UCI Adult dataset *Vainilla WGAN* (*linear activation*)	F_1-*score*	0.7874	0.1965	0.4334	0.5358
	L^1*distance to real data*	$L0$: 1.5348 $L1$: 1.4133	L0: 3.4925	L1: 2.5711	–
UCI Adult dataset *Smirnov WGAN* (*ST activation*)	F_1-*score*	0.7874	0.7745	0.7654	0.7858
	L^1*distance to real data*	$L0$: 1.5348 $L1$: 1.4133	L0: 1.9878	L1: 1.8455	–
UCI E – shop dataset *Vainilla WGAN* (*linear activation*)	F_1-*score*	0.9445	0.3779	0.3501	0.3585
	L^1*distance to real data*	$L0$: 1.5331 $L1$: 1.5516	L0: 3.4925	L1: 2.5711	–
UCI E – shop dataset *Smirnov WGAN* (*ST activation*)	F_1-*score*	0.9445	0.6602	0.7384	0.6339
	L^1*distance to real data*	$L0$: 1.5331 $L1$: 1.5516	L0: 2.8880	L1: 2.5077	–
CTU – 13 botnet dataset *Vainilla WGAN* (*linear activation*)	F_1-*score*	0.9899	0.5522	0.3503	0.8076
	L^1*distance to real data*	$L0$: 0.5735 $L1$: 0.1891	L0: 3.1067	L1: 3.6628	–
CTU – 13 botnet dataset *Smirnov WGAN* (*ST activation*)	F_1-*score*	0.9899	0.9395	0.9051	0.8551
	L^1*distance to real data*	$L0$: 0.5735 $L1$: 0.1891	L0: 3.1283	L1: 1.8701	–

These datasets were previously described in subSection 5.2 It is worth noting that we configured the same type of WGAN architecture for these experiments to design an ablation experiment in which only the activation function at the last layer of the generator was changed from linear to Sminorv transform. We summarize in Table 5 the results obtained with such WGAN configuration but similar results were obtained when the WGAN configuration was changed (e.g., adding or removing layers and neural units to the generator and discriminator).

We sampled twice the real data set for each label and computed the L^1 distance of the two samples. Each sample got 500 elements from the real data uniformly at random and with replacement. The experiment was repeated 5 times and the result shown in Table 5 is the median of the distances obtained in the 5 experiments. The distance from synthetic data to real data for each label was computed using the generator that obtained the best score in the marginal $F1$-score test for each label. A sample of 500 elements obtained from the WGAN generator was used to compute the L^1 distance from the synthetic data to the real one. The experiment was repeated 5 times and the median of the obtained distances was shown in the table.

To obtain the $F1$-score when only synthetic data is used to train the nested ML classifier, we used the $F1$-elitism criteria and the mixing procedure described in [20]. For each label, the top ten generators in the marginal $F1$-score test were selected. Then, one generator of each label was selected uniformly at random to get a fully synthetic sample. The size of the synthetic sample and the proportion of each label were the same as in the real data set. Using the synthetic data, a nested ML classifier was trained and tested on a second real data set. The experiment was repeated 50 times and the best result was shown in the table.

Table 5 compiles the main results of these experiments: (i) L^1 distance from a real to a synthetic sample, (ii) marginal $F1$-score on testing of labels 0 and 1 when only one type (label) of synthetic data is used for training the nested Ml model and (iii) $F1$-score on testing when only synthetic data from the two labels is used for training the nested ML model. The third column of Table 5 shows the $F1$-score on testing when only real data is used to train the nested ML model and the distance of two samples of real data for each label. These figures establish the real data level performance, to which WGANs should try to be as close as possible. Fourth and fifth columns present the $F1$-score and L^1 distance for each label and the last column shows the $F1$-score when only synthetic data from both labels is used for training the nested ML model.

The results in Table 5 show that when using WGAN synthetic data to train a nested classifier, the $F1$-score is significantly higher if the Sminorv transform is used as the activation function in the last layer of the generator. This improvement appears both when we use single-label data (columns 4 and and 5) and when we blend data from the two labels to generate a fully synthetic dataset (column 6).

It is worth noting that in general, the quality of the synthetic data generated by WGANs with linear activation was very poor, as the marginal $F1$-scores (columns 4 and 5) show values less than 0.5. Analysing in detail the corresponding confusion matrices, we observed that the nested classifier did not achieve to distinguish label 0 from label 1 data and nearly all inputs were classified as the same type (i.e., the stopped clock case). Regarding the improvement achieved by linear activation WGANs when the nested classifier in the CTU-13 scenario is trained with synthetic data for both labels (column 6),we conjecture that it is mainly due to some statistical artifacts generated by combining two different data distributions for a a relatively simple to solve classification problem. Conversely, the synthetic data generated by ST-based WGANs achieved $F1$-score values closer to the ones obtained when the nested ML classifier was trained with real data and in particular, when the UCI-Adult data set was considered, the obtained performance was the same.

When L^1 distances are analysed, it can be observed that, in general, ST-based WGANs obtain synthetic data distributions (columns 4 and 5) that are closer to the real data than when linear activation was used. In some scenarios (e.g., UCI E-shop, label 1), even when the linear activation-based WGAN can produce synthetic data that is close in distance to the real data, this metric is not reflecting that some discrete features in the real data (e.g. categorical variables) are being generated as continuous variables by the vanilla WGAN. On the contrary, ST-based WGANs generate data that is equally distributed as the real data (i.e., small distance to real data) but in sharp contrast with vanilla WGANs, only the discrete values that are present in the real data are replicated in the synthetic data.

6. Conclusions

We propose a novel activation function to be used as output of the generator agent of a GAN. This activation function is based on the Smirnov probabilistic transformation (ST) and is specifically designed to improve the quality of the generated data. This transformation bends the shape of an input random variable and turns it into the distribution of a given output.

The proposed ST-based activation function provides a general approach that deals not only with the replication of categorical variables but with any type of data distribution (continuous or discrete). Moreover, this activation function is derivable and therefore, it can be integrated in a neural network model without affecting to the gradient calculation done by the backpropagation algorithm during the GAN training process. To convert a standard GAN into a ST-based GAN, we only need to change the activation functions of the last layer in the generator by the corresponding ST functions that are precomputed before starting the GAN training.

We empirically demonstrated that, in the experiments carried out, the synthetic data generated with our solution presents a high quality comparable to the original data. We used a rendered and a real data set to test the quality of the generated data by our ST-based WGANs with respect to vanilla WGANs and simple mean-based generators. In addition, we tested our solution on three publicly available data sets obtaining similar results with respect to the quality of the synthetic data obtained by IST-based WGAN when compared to the data obtained using vanilla WGANs.

We assessed data fidelity in two different ways: a) From a statistical perspective, using two distance functions (based on the L^1 distance and Jaccard index) that allow us to quantitatively and graphically compare the evolution of the quality of the synthetic data generated during the GAN training; and b) from a practical viewpoint, testing the performance of a nested machine learning classifier when the synthetic data completely substitute real data in the training of the classifier.

The experimental results evidence a clear outperformance of the GAN network tuned with ST-based activation function with respect to a standard GAN. The quality of the generated data is so high that it can fully substitute real data for training a nested classifier without a fall in the obtained performance. This result encourages the use of GANs to produce high-quality synthetic data that are applicable in scenarios in which data privacy must be guaranteed.

Due to the ill-convergence of the GAN training, we introduced a novel approach based on the observed performance of a nested ML task that uses the synthetic data produced by the generator at each epoch. Future work should investigate new distances that can (i) guide the convergence of the GAN during training toward high-fidelity data generation and (ii) measure data quality not only from a statistical perspective, but also considering to what extent synthetic data can completely replace real data in different tasks (e.g., to train ML models). Using these distances, efficient stopping criteria for GAN training should also be investigated.

It is worth mentioning that, even though in this paper we have applied a fully connected neural network architecture, the solution introduced in this paper can be straightforwardly generalized to work with more complex architectures. The reason is that the Smirnov transformation does not interact with the inner structure of the network, but only acts as a proxy that 'bends' the output to adapt it to a certain probability distribution. In particular, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) may benefit from using this activation function at the output stage.

Finally, and interesting prospective work would be to analyze the impact of this activation function to other types of neural networks, like classical neural classifiers of Long-Short Term Memory (LSTM) networks, to create more complex neural outputs, such as Poisson-like with compact support. Furthermore, besides the deep learning world, the proposed transformation can be used to transform data in other ML fields, like a new kernel method for Support Vector Machines (SVM) or a projection method for a dimensionality reduction algorithm, giving rise to new and exciting solutions.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partially supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 833685 (SPIDER) and Grant 101015857 (Teraflow). The first-named author wants to thank the hospitality of the Department of Mathematics at Universidad Autónoma de Madrid, where this work was partially completed.

References

[1] X.-W. Chen, X. Lin, Big data deep learning: Challenges and perspectives, IEEE Access 2 (2014) 514–525, https://doi.org/10.1109/ACCESS.2014.2325029.
[2] J. Ferlez, C. Faloutsos, J. Leskovec, D. Mladenic, M. Grobelnik, Monitoring network evolution using mdl, in: 2008 IEEE 24th International Conference on Data Engineering, IEEE, 2008, pp. 1328–1330.
[3] S. Vakaruk, J.E. Sierra-García, A. Mozo, A. Pastor, Forecasting automated guided vehicle malfunctioning with deep learning in a 5g-based industry 4.0 scenario, IEEE Commun. Mag. 59 (11) (2021) 102–108.
[4] Regulation (eu) 2016/679 (general data protection regulation) in the current version of the oj l 119, 04.05.2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679
[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates Inc, 2014, pp. 2672–2680.
[6] Z. Wang, Q. She, T.E. Ward, Generative Adversarial Networks in computer vision: A survey and taxonomy, arXiv preprint arXiv:1906.01529 (2019).
[7] N. Gao, H. Xue, W. Shao, S. Zhao, K.K. Qin, A. Prabowo, M.S. Rahaman, F.D. Salim, Generative Adversarial Networks for spatio-temporal data: A survey, arXiv preprint arXiv:2008.08903 (2020).
[8] A. Jabbar, X. Li, B. Omar, A survey on Generative Adversarial Networks: Variants, applications, and training, arXiv preprint arXiv:2006.05132 (2020).
[9] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, Y. Zheng, Recent progress on Generative Adversarial Networks (GANs): A survey, IEEE Access 7 (2019) 36322–36333.
[10] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, F.-Y. Wang, Generative adversarial networks: introduction and outlook, IEEE/CAA J. Automatica Sinica 4 (4) (2017) 588–598, https://doi.org/10.1109/JAS.2017.7510583.
[11] M.F. Naeem, S.J. Oh, Y. Uh, Y. Choi, J. Yoo, Reliable fidelity and diversity metrics for generative models, in: International Conference on Machine Learning, PMLR, 2020, pp. 7176–7185.
[12] P. Grnarova, K.Y. Levy, A. Lucchi, N. Perraudin, I. Goodfellow, T. Hofmann, A. Krause, A domain agnostic measure for monitoring and evaluating gans, Adv. Neural Inform. Process. Syst. 32 (2019) 12092–12102.
[13] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, K. Weinberger, An empirical study on evaluation metrics of generative adversarial networks, arXiv preprint arXiv:1806.07755 (2018).
[14] V. Khrulkov, I. Oseledets, Geometry score: A method for comparing generative adversarial networks, in: International Conference on Machine Learning, PMLR, 2018, pp. 2621–2629.
[15] A. Borji, Pros and cons of gan evaluation measures, Comput. Vis. Image Underst. 179 (2019) 41–65.
[16] M. Lucic, K. Kurach, M. Michalski, S. Gelly, O. Bousquet, Are gans created equal? a large-scale study, arXiv preprint arXiv:1711.10337 (2017).
[17] K. Kurach, M. Lučić, X. Zhai, M. Michalski, S. Gelly, A large-scale study on regularization and normalization in gans, in: International Conference on Machine Learning, PMLR, 2019, pp. 3581–3590.
[18] K. Shmelkov, C. Schmid, K. Alahari, How good is my GAN?, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp 213–229.
[19] L. Cui, P. Zhao, B. Li, X. Li, K. Wang, Y. Yang, X. Bu, S. Fei, A stopping criterion for the training process of the specific signal generator, Inform. Technol. Control 50 (1) (2021) 153–170.
[20] A. Mozo, Á. González-Prieto, A. Pastor, S. Gómez-Canaval, E. Talavera, Synthetic flow-based cryptomining attack generation through generative adversarial networks, Sci. Rep. 12 (1) (2022) 1–27.
[21] J. Jordon, J. Yoon, M. Van Der Schaar, Pate-gan: Generating synthetic data with differential privacy guarantees, in: International conference on learning representations, 2018.
[22] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, arXiv preprint arXiv:1611.01144 (2016).
[23] E. Choi, S. Biswal, B. Malin, J. Duke, W.F. Stewart, J. Sun, Generating multi-label discrete patient records using generative adversarial networks, in: Machine learning for healthcare conference, PMLR, 2017, pp. 286–305.
[24] Z. Kong, S. Ding, Generative adversarial networks with inverse transformation unit, arXiv preprint arXiv:1709.09354 (2017).
[25] Z. Lin, Y. Shi, Z. Xue, IDSGAN: Generative Adversarial Networks for attack generation against intrusion detection, CoRR abs/1809.02077 (2018). arXiv:1809.02077
[26] H. Navidan, P.F. Moshiri, M. Nabati, R. Shahbazian, S.A. Ghorashi, V. Shah-Mansouri, D. Windridge, Generative Adversarial Networks (GANs) in networking: A comprehensive survey & evaluation, Comput. Netw. 108149 (2021).
[27] L. Vu, C.T. Bui, Q.U. Nguyen, A deep learning based method for handling imbalanced problem in network traffic classification, in: Proceedings of the Eighth International Symposium on Information and Communication Technology, SoICT 2017, Association for Computing Machinery, New York, NY, USA, 2017, p. 333–339.
[28] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks, in: International conference on machine learning, PMLR, 2017, pp. 214–223.
[29] V. Nagarajan, J.Z. Kolter, Gradient descent GAN optimization is locally stable, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, 2017, pp. 5585–5595.
[30] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, arXiv preprint arXiv:1606.03498 (2016).
[31] C.K. Sønderby, J. Caballero, L. Theis, W. Shi, F. Huszár, Amortised MAP inference for image super-resolution, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings, 2017.
[32] M. Arjovsky, L. Bottou, Towards principled methods for training Generative Adversarial Networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings, 2017.
[33] K. Roth, A. Lucchi, S. Nowozin, T. Hofmann, Stabilizing training of Generative Adversarial Networks through regularization, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, 2017, pp. 2018–2028.
[34] S.M. Ross, A first course in probability, Tech. rep., New York, 1976.
[35] L. Devroye, Nonuniform random variate generation, Handbooks in operations research and management science 13 (2006) 83–121.
[36] H. Thanh-Tung, T. Tran, Catastrophic forgetting and mode collapse in gans, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–10.
[37] C. De Boor, C. De Boor, A practical guide to splines, vol. 27, Springer-verlag New York, 1978.
[38] A.W. van der Vaart, J.A. Wellner, Glivenko-Cantelli theorems, Weak Convergence and Empirical Processes, Springer (1996) 122–126.
[39] R.B. Nelsen, An introduction to copulas. new york, usa, Lecture Notes in Statistics, 1999.
[40] T.T. Tanimoto, Elementary mathematical theory of classification and prediction, Internal IBM Technical Report (1958).
[41] H.T. Kam, et al., Random decision forest, in: Proceedings of the 3rd international conference on document analysis and recognition, Vol. 1416, Montreal, Canada, August 1995, p. 278282.
[42] Á. González-Prieto, A. Mozo, S. Gómez-Canaval, E. Talavera, Github repository for "Improving the quality of generative models through Smirnov transformation", URL: https://github.com/AngelGonzalezPrieto/GANSmirnov (2022). doi:10.5281/zenodo.6050239.
[43] A. Pastor, A. Mozo, D.R. Lopez, J. Folgueira, A. Kapodistria, The Mouseworld, a security traffic analysis lab based on NFV/SDN, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, 2018, pp. 1–6.
[44] A. Pastor, A. Mozo, S. Vakaruk, D. Canavese, D.R. López, L. Regano, S. Gómez-Canaval, A. Lioy, Detection of encrypted cryptomining malware connections with machine and deep learning, IEEE Access 8 (2020) 158036–158055.

[45] R. Kohavi, et al., Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid., in: Kdd, Vol. 96, 1996, pp. 202–207.
[46] M. Łapczyński, S. Białows, Discovering patterns of users' behaviour in an e-shop-comparison of consumer buying behaviours in poland and other european countries, Studia Ekonomiczne 151 (2013) 144–153.
[47] S. Garcia, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, Comput. Secur. 45 (2014) 100–123.
[48] A. Odena, C. Olah, J. Shlens, Conditional image synthesis with Auxiliary Classifier GANs, in: International conference on machine learning, PMLR, 2017, pp. 2642–2651.
[49] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville, Improved training of Wasserstein GANs, arXiv preprint arXiv:1704.00028 (2017).
[50] E. Parzen, On estimation of a probability density function and mode, Ann. Math. Stat. 33 (3) (1962) 1065–1076.