Generative Adversarial Networks for Bitcoin Data Augmentation

Francesco Zola* , Jan Lukas Bruse*, Xabier Etxeberria Barrio*, Mikel Galar†, Raul Orduna Urrutia*
* Vicomtech Foundation, Basque Research and Technology Alliance (BRTA)
Paseo Mikeletegi 57, 20009 Donostia/San Sebastian, Spain
{fzola, jbruse, xetxeberria, rorduna}@vicomtech.org
†Institute of Smart Cities, Public University of Navarre, 31006 Pamplona, Spain
mikel.galar@unavarra.es

***Abstract*— In Bitcoin entity classification, results are strongly conditioned by the ground-truth dataset, especially when applying supervised machine learning approaches. However, these ground-truth datasets are frequently affected by significant class imbalance as generally they contain much more information regarding legal services (Exchange, Gambling), than regarding services that may be related to illicit activities (Mixer, Service). Class imbalance increases the complexity of applying machine learning techniques and reduces the quality of classification results, especially for underrepresented, but critical classes.**

In this paper, we propose to address this problem by using Generative Adversarial Networks (GANs) for Bitcoin data augmentation as GANs recently have shown promising results in the domain of image classification. However, there is no "one-fits-all" GAN solution that works for every scenario. In fact, setting GAN training parameters is non-trivial and heavily affects the quality of the generated synthetic data. We therefore evaluate how GAN parameters such as the optimization function, the size of the dataset and the chosen batch size affect GAN implementation for one underrepresented entity class (Mining Pool) and demonstrate how a "good" GAN configuration can be obtained that achieves high similarity between synthetically generated and real Bitcoin address data. To the best of our knowledge, this is the first study presenting GANs as a valid tool for generating synthetic address data for data augmentation in Bitcoin entity classification.

***Index Terms*— Generative Adversarial Network, class imbalance, data augmentation, Bitcoin classifier, address behaviour**

I. Introduction

Bitcoin (or BTC) is a cryptocurrency based on a publicly shared ledger called blockchain [23]. All transactions are stored in blocks of the blockchain that cannot be manipulated or changed [4]. The access to this information is free for each user belonging to the Bitcoin network, while Bitcoin user identity is protected by anonymity. Decreasing anonymity in the Bitcoin network has become a challenge when trying to discover new entities in the network [11], or when aiming to detect entities related to illicit or abnormal activities in order to improve the trustworthiness within the network.

For Bitcoin entity classification, results are strongly conditioned by the ground-truth dataset, especially when using supervised machine learning approaches. Usually, public datasets or data generated by scraping forums and web-sites are used. Nevertheless, due to the complexity of the Bitcoin network and its anonymity policy, these datasets are typically characterized by heavily imbalanced classes of Bitcoin entities - with some classes being highly underrepresented compared to others. Such class imbalance affects the quality of a learning system because, once trained using imbalanced classes, learning algorithms are conditioned to resolve complicated classification problems based on a skewed class distribution and thus fail to detect underrepresented classes well [6].

The class imbalance problem becomes even more relevant for restricted datasets for which it is difficult to detect and add new information, such as the Bitcoin blockchain. In fact, there is generally much more information available pertaining to legal services that do not need to obscure their information (such as Exchanges or legal Markets) than for entities that intend to mask the traces of their services (such as Mixers or Ransomware), which are often related to illicit activities. This calls for novel approaches that can augment the dataset of such critical services with synthetically created instances, ultimately aiming to improve classification results. The most common technique currently adopted to address the imbalance problem is to not consider certain classes [32], or to apply various types of sampling methods, like over-sampling for the least represented classes [11] or under-sampling for the most represented ones [18].

To the best of our knowledge, this is the first work that addresses Bitcoin class imbalance by introducing address data augmentation using generative adversarial networks (GANs). The key idea behind GANs is to create synthetic data that cannot be distinguished from real data. The "adversarial" aspect is introduced by using two algorithms/networks working against each other in order to improve their ability to learn and reproduce a real input dataset. The potential to learn and copy almost every dataset distribution has led researchers to apply GANs predominantly in the domain of image processing.

Here, these concepts are leveraged in order to take a step forward towards resolving the imbalance problem related to Bitcoin entity classification. Setting GAN training parameters is non-trivial and heavily affects the quality of the generated synthetic data [9]. Therefore, the aim of this paper is to study how GAN configuration parameters affect the produced synthetic data and which configuration should be used for achieving "best" data augmentation i.e. for generating synthetic data as similar as possible to real data. To investigate the impact of GAN configurations, several tests

changing three important GAN parameters were carried out: the optimizer function, the size of the real input dataset and the batch size. In this manner, we were able to evaluate how each variable affects the training and the generation phase. Moreover, we present how a "good" GAN configuration can be obtained that allows for efficient generation of synthetic address data.

The initial Bitcoin blockchain dataset used in this work was composed by entities from 6 distinct Bitcoin classes: Exchange, Gambling, Marketplace, Mining Pool, Mixer and Service. Here, our experiments are predominantly focused on one critical Bitcoin class - the one that presented with the smallest number of samples in the address behavioural dataset - the Mining Pool.

The rest of the paper is organized as follows. Section II describes related work. Section III introduces GAN concepts and how they were implemented in this paper. Section IV shows an overview of the used datasets and presents our experiments. Section V describes the obtained results and finally, in Section VI, we draw conclusions and provide guidelines for future work.

II. Related work

A. Bitcoin entity recognition and class imbalance problem

To date, analysis of the Bitcoin network is predominantly focused on entity classification with the aim of detecting illicit or abnormal activities and relating them with cyber-security threats. However, often there is only little information available regarding certain actors of the Bitcoin network and critical classes are underrepresented making robust classification of all known entities difficult. Generally, imbalanced class distributions are known to hinder classifier learning [6].

Harlev et al. [11] applies a supervised machine learning algorithm to predict the type of yet-unidentified entities, resolving the imbalance problem by using Synthetic Minority Over-Sampling Technique (SMOTE) to over-sample under-represented classes. However, results show that the model struggles with classes that have a low number of samples. In [24], a single entity (exchange) is detected by extracting features related to the transaction directed hypergraph. In this case, the authors randomly sample an equal number of labeled and unlabeled addresses for training and testing their model, repeating the process 10 times. Zola et al. [33] implements a cascading machine learning model to detect Bitcoin entities belonging to 6 classes, but does not consider the unbalance of the data. In [17], new features for Bitcoin address classification are introduced addressing the imbalance problem using stratified random sampling. The best results show a general increase in entity classification except for two classes (Faucet and Market). Liang et al. [16] present an algorithm based on network representation learning to train their address multi-classifiers with imbalanced data. Bartoletti et al. [1] experiment with several machine learning algorithms to detect the Bitcoin Ponzi scheme using two literature approaches to solve the class imbalance problem: a cost-sensitive [31] and a sampling-based approach [3].

Since GANs have recently shown very promising results in the image processing domain [29], we sought to investigate their potential for solving the class imbalance problem in Bitcoin entity classification by generating additional synthetic data.

B. Generative Adversarial Network (GAN) applications

Generative Adversarial Networks (GANs) are an approach of generative modelling using deep learning methods such as convolutional neural networks (CNN). In [10], the potential of a GAN estimation approach is presented through a qualitative and quantitative evaluation of the generated samples.

The majority of GAN studies use images as input data, for example, in [13] a new training methodology for creating a GAN that can generate realistic photographs of human faces is presented. Minaee et al. [21] presents a machine learning framework based on GANs, which is able to generate fingerprint images; while in [12], authors investigate adversarial networks as a solution for the image-to-image translation problem transferring style from one image to another.

Recently, GANs have been used to perform data augmentation as well. In [15] and in [5], two distinct GAN versions are designed to approximate the true data distribution and to generate data for the minority class of various imbalanced datasets. In [19], the presented Balancing GAN (BAGAN) methodology aims to generate realistic minority-class images. In [27], a method to generate synthetic abnormal magnetic resonance images (MRI) with brain tumors using a GAN approach is presented. Similarly, Frid et al. [8] apply GANs to show that a classifier trained with synthetic images of liver lesions achieves better values of sensitivity and specificity than a classifier implemented with an imbalanced dataset. Other examples include breast cancer classification as shown in [30], where a conditional GAN is implemented in order to synthesize lesions from real mammogram images.

In the field of cyber-security, Merino et al. [20] suggest that GANs are a viable approach for improving cyber-attack intrusion detection systems. They generate new cyber-attack data from existing data with the goal of balancing the datasets.

Based on the promising results in the image processing domain and - recently - in other domains, we propose here to investigate how GANs can be applied to generate synthetic behavioural data of a specific, typically underrepresented Bitcoin entity. In particular, several GAN configurations are tested in order to determine the parameter setting that should be used to generate "good" quality synthetic samples with high similarity between synthetic and real data.

To the best of our knowledge, this is the first work that explores the use of GANs for Bitcoin data augmentation.

III. Methodology

A. GAN overview

Generative modelling is an unsupervised learning task in the field of machine learning introduced by [10]. A GAN is composed of two networks that compete with each other, thereby increasing their ability to learn from each other. The

aim of GANs is to discover and learn regularities and patterns present in input data and generate new synthetic samples with high similarity to the original/real dataset. GANs usually follow the structure presented in Figure 1.

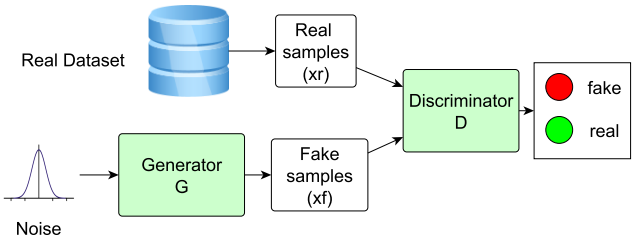


Fig. 1: Generative Adversarial Network (GAN) architecture

A GAN is composed by two concurrent neural networks: a Generator (G) and a Discriminator (D). The task of the first network G is to generate synthetic samples while the second one D evaluates their authenticity. In particular, a generative model G captures the input data distribution, and a discriminative model D estimates the probability that a sample came from the training data rather than from G. This procedure is repeated for a certain number of times, named epochs. During each epoch, cost functions are calculated, and according to these values the weights of the two networks are changed. The idea is to minimize discriminator mistakes increasing the similarity between the synthetic (fake) and real (original) samples.

As introduced by Goodfellow et al. [10], GAN training is characterized by a game between two loss functions: one that involves the discriminator D and the other that involves the generator G. This problem is typically represented as a min-max optimization problem (Equation 1).

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z))] \quad (1)$$

During the training phase, using gradient descent optimization, both G and D are updated simultaneously through stochastic gradient updates. Yet, this method could be affected by several issues that can cause non-convergence of the solution. Local minima and saddle points, for example, can stall the training and pathological curvature can slow down training without finding the solution. This problem has already been addressed by many authors [26], [22], [14], where each one proposes and analyzes several optimization functions. These studies conclude that each scenario has its own suitable optimization function, and that there is no "one-fits-all" solution that works in every situation.

For the implementation of GANs in this paper, we thus aimed to find a suitable GAN configuration and compare two of the most used optimization methods: Root Mean Square Propagation (RMSProp) and Adaptative Moment Optimization (ADAM). Root Mean Square Propagation tries to diminish oscillations that can slow down the optimization [28] and automatically adjusts the learning rate - it is able to choose a different learning rate for each parameter. In RMSProp, updates are done according to Equation 2 [25].

$$\begin{aligned} &\nu_t = \alpha\nu_{t-1} + (1-\alpha) * g_t^2 \\ &\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t \\ &\omega_{t+1} = \omega_t + \Delta\omega_t \\ &\eta : \text{Initial Learning rate} \\ &\nu_t : \text{Exp. Avg. of squares of gradients} \\ &g_t : \text{Gradient at time t along } \omega^j \\ &\alpha : \text{Hyperparameter} \end{aligned} \quad (2)$$

In Equation 2, ν_t represents the exponential average of the square of the gradient. The exponential average is useful as it helps weigh more recent gradient updates more than the less recent ones. Then, step size $\Delta\omega_t$ is calculated, moving in the direction of the gradient. This step size is affected by the exponential average and two parameters η (Initial Learning rate) and ϵ. Finally, the step (ω_{t+1}) is updated. The α hyperparameter is typically chosen to be 0.9 and ϵ is chosen to be $1e^{-10}$.

ADAM was introduced for the first time in [14] and, similar to RMSProp, has the aim to diminish oscillations during the gradient descent process. However, ADAM also accelerates the optimization in the direction of the minimum.

Equation 2 and Equation 3, show the similarity between RMSProp and ADAM. However, for ADAM (Equation 3), an additional equation is considered and the step size computation has a small variation. The additional equation is the exponential average of gradients. This equation tries to avoid zig-zag directions. ADAM computes step size by multiplying the exponential average of the gradient as well. Regarding the hyperparameters, β_1 is typically chosen to be 0.9, β_2 is kept around 0.999 and ϵ is chosen to be $1e^{-10}$.

For training the GAN, it is important fix the size of the initial dataset and choose the size of the batch. This batch size, in fact, represents the number of elements of the real dataset and the synthetic dataset used at once to update the weight of the generator and discriminator network [9].

B. GAN implementation

The aim of this study is to determine the adequate configuration of a GAN able to recreate synthetic Bitcoin address behaviour. The proposed approach may be used to resolve Bitcoin class imbalance problems for improved entity classification.

As described in Section III-A, the GAN is composed of two networks. For the purpose of this study, a neural network with three hidden layers was used as generator G. In particular, each layer was composed by respectively 512, 256 and 128 neurons, all using the Rectified Linear Unit (ReLu) activation function. For the discriminator D, a neural network with three hidden layers was implemented. Each layer was created by respectively 256, 512 and 256 neurons.

$$
\begin{aligned}
&\nu_t = \beta_1 * \nu_{t-1} + (1-\beta_1) * g_t \\
&s_t = \beta_2 * s_{t-1} + (1-\beta_2) * g_t^2 \\
&\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t} + \epsilon} * g_t \\
&\omega_{t+1} = \omega_t + \Delta\omega_t \\
&\eta : \text{Initial Learning rate} \\
&\nu_t : \text{Exp. Avg. of gradients along } \omega_j \\
&s_t : \text{Exp. Avg. of squares of gradients along} \omega_j \\
&g_t : \text{Gradient at time t along } \omega^j \\
&\beta_1, \beta_2 : \text{Hyperparameters}
\end{aligned} \tag{3}
$$

The objective of the two networks during the training phase is to optimize and minimize the generation loss and the discrimination loss as explained in Section III-A. The tested optimization functions were RMSProp and ADAM. Both of the functions were configured with equal learning rate (0.001).

In order to improve the GAN learning skills, each feature belonging to the original input dataset was normalized. This normalization allowed us to limit all feature distributions in the range of 0 to 1. For each feature, the maximum and minimum were computed and Equation 4 was applied.

$$X_{norm} = \frac{x - X_{min}}{X_{max} - X_{min}} \tag{4}$$

When the training phase starts, in each epoch, a set of N samples coming from a random normal distribution were used as input to G in order to create a first version of synthetic samples. In our implementation, we decided to keep the N input values fixed and equal to 100. Once the synthetic samples were generated, D was trained in order to distinguish the synthetic samples and the real samples. D was trained with a labelled dataset composed of 2xM elements: M synthetic data with the $0-label$ and M real data with the $1-label$. During training, each T epochs, the process was stopped in order to evaluate the respective solution. Only if the solution satisfied the conditions indicated in Algorithm 1, the training was stopped. For our purposes, T was chosen equal to $1,000$ epochs.

As indicated in Algorithm 1, during evaluation, G creates a set of M synthetic samples $G(N, M)$ that subsequently was used as input for the discriminator $D(Z)$. Once D finished the classification, the number of synthetic samples detected as real samples with an accuracy greater than a threshold value ($thr1$) was normalized. This operation is reflected in Algorithm 1 by count($C > thr1$) / len(C), where $thr1$ was fixed to 0.90. This test was repeated n (5) times and the training process was stopped if in each of the 5 tests the number of samples with accuracy higher than $thr1$ was more than 90% of the population ($thr2$ = 0.90). If this condition was not verified, the GAN resumed training until the next test (in $1,000$ epochs).

Algorithm 1: GAN evaluation phase

```
epoch = 0;
training GAN;
epoch = epoch + 1;
if epoch % T == 0 then
    test = 0 ;
    accuracy = 0 ;
    for i=0 to n do
        Z = G(N, M) ;
        C = D(Z) ;
        accuracy = count(C > thr1) / len(C) ;
        if accuracy > thr2 then
            test = test + 1;
        end
    end
    if test == n then
        exit training;
    else
        resume training;
    end
end
```

G = Generator network
D = Discriminator network
N = 100 (size of input Gaussian samples)
M = batch size
T = $1,000$ (epochs for test)
n = 5 (number of the test repetition)
$thr1$ = 0.90 (threshold for synth. classification)
$thr2$ = 0.90 (threshold for similarity real/synth.)

IV. Experimental Framework

A. Blockchain data

The first step was to extract a Bitcoin address dataset from the Bitcoin mainnet[1]. The whole blockchain was downloaded, from the beginning until block number 570,000, corresponding to blocks mined until April 3rd 2019, 09:20:08 AM. The Bitcoin blockchain data were downloaded using the Bitcoin Core[2].

The second dataset used in our analysis was obtained from WalletExplorer[3]. This platform represents a database where information about known entities and their related addresses are stored. This database is continuously updated and has been used as a "ground truth" for many Bitcoin-related studies, such as [2], [32], [24]. Here, WalletExplorer data were divided into six classes:

- *Exchange*: entities that allow their customers to trade among cryptocurrencies or to change cryptos for fiat currencies (or vice-versa);
- *Gambling*: entities that offer gambling services based on Bitcoin currency (casino, betting, roulette, etc.);

[1]https://bitcoin.org/en/glossary/mainnet
[2]https://bitcoin.org/en/download
[3]https://www.walletexplorer.com/

- *Mining Pool*: entities composed of a group of miners that work together sharing their resources in order to reduce the volatility of their returns;
- *Mixer*: entities that offer a service to obscure the traceability of their clients' transactions;
- *Marketplace*: entities allowing to buy any kind of goods or services using cryptocurrencies. Some of them potentially related to illicit activities [7];
- *Service*: entities that allow users to lend Bitcoins and passively earn interests, or allow them to request a loan.

Class	# Address WalletExplorer	# Entity WalletExplorer
Exchange	9,947,450	144
Gambling	3,050,899	76
Marketplace	2,349,111	20
Mining Pool	85,887	27
Mixer	475,781	37
Service	250,788	23
Total	**16,159,916**	**327**

TABLE I: Overview of used WalletExplorer data

As shown in Table I, 327 different entities and more than $16,000,000$ addresses were downloaded from WalletExplorer. The WalletExplorer data were combined with the whole blockchain in order to restrict the following analyses to properly labelled data only. This new, labelled "ground truth" dataset allowed us to create the Bitcoin address dataset and implement supervised machine learning based on known data.

Following indications provided in [33], features related to each distinct addresses were extracted from the Bitcoin address dataset. The created address dataframe was composed of 7 features: the number of transactions in which a certain address was detected as receiver/sender, the amount of BTC received/sent from/to this address, the balance, uniqueness (if this address was used in one transaction only) and the number of siblings.

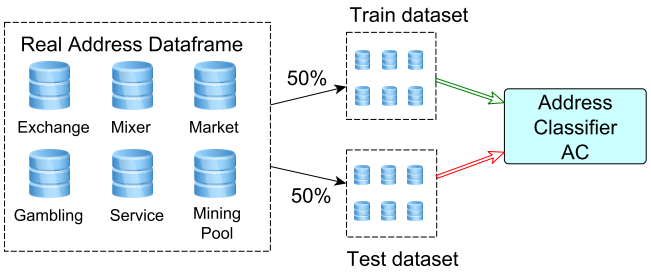


Fig. 2: Address Classifier (AC) schema

We then split the address dataframe into two parts, as indicated in Figure 2 - the train and test dataset with a proportion of 50/50 keeping class distributions unchanged (stratified). The training set was used to train a machine learning model based on a Random Forest classifier creating our baseline classifier, called Address Classifier (AC). Then, the testing set was used to compute a first evaluation of how

Class	Accuracy %	F1-score	Number of testing samples
Exchange	97.44	0.96	4,975,431
Gambling	87.61	0.90	1,523,652
Marketplace	96.46	0.98	1,174,643
Mining Pool	74.16	0.81	42,622
Mixer	79.32	0.82	238,056
Service	65.68	0.73	125,257
Total			**8,079,661**

TABLE II: Address dataframe: obtained testing accuracy with real data

the baseline classifier trained with real data predicts the entity classes related to a certain address, as shown in Figure 2.

Table II shows a summary of accuracy, f1-score and number of real samples used for testing the baseline model (AC). The created AC model yielded generally higher accuracies for detecting samples belonging to the most populated classes. In fact, the Exchange and Marketplace classes were detected with an accuracy over 96%, while the Service class (having overall the fewest number of samples) was detected with 65.68% accuracy only. Table II highlights how underrepresented classes resulted in overall weak classification results.

In this study, we conducted experiments focusing only on the most underrepresented class - the Mining Pool class - and tried to generate respective synthetic samples for data augmentation. We opted to consider only the Mining Pool class as it represented the "worst case" in terms of data due to its few number of samples (distinct addresses). The Mining Pool population was more than 100 times smaller than the Exchange population, and almost 3 times smaller than the Service population, and was detected with an accuracy of 74.16% by the baseline classifier AC. Note that in the following sections, each time we talk about a dataset (or training dataset) we refer to a dataset composed of Mining Pool samples only.

B. GAN experiments

The following experiment sought to investigate the effects of different GAN configurations. The configuration for each test was obtained by changing three relevant GAN parameters: the optimization function, the size of the input (ground-truth) dataset and the batch size used for training the network. The two optimization functions tested in our setting were the Root Mean Square Propagation (RMSProp) and the Adaptative Moment (ADAM) as explained in Section III-A.

The ground-truth dataset formed the actual input of the GAN and represented a part of the training dataset which was too big and variable to be used entirely ($> 42,000$). In the following experiment, this dataset was generated starting from the training dataset considering Mining Pool samples only, and was normalized as described in Section III-B. In particular, three ground-truth datasets were considered, each one starting from row 0 of the Mining Pool training dataset, respectively with $10,000$, $5,000$ and $1,000$ samples. For the batch size, values of 400, 200, 100 and 50 samples were chosen.

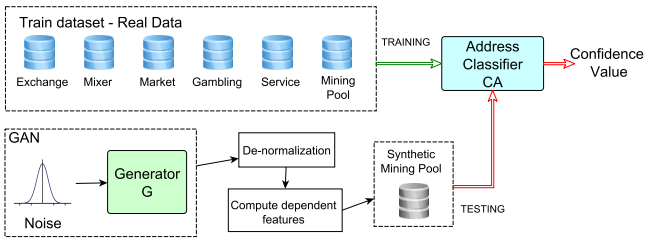


Fig. 3: Experimental schema for calculating confidence values, used as measure of similarity between synthetic and real data

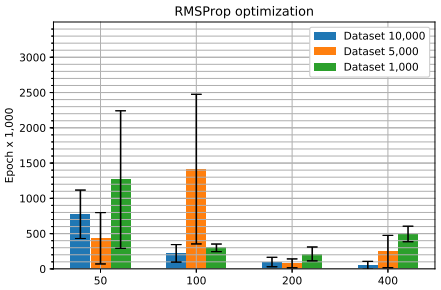
Each configuration was tested three times in order to check the repeatability of the respective configuration. According to these pre-requisites, we generated 72 different GANs, each one trained as explained in Section III-B.

As the real dataset was composed of some non-independent features, the 7 initial features were reduced to 5, as explained in Section IV-A. In fact, the total amount of BTC received, the total amount of BTC sent and the balance are non-independent variables, so we decided to train the GAN such that it only learns two of them, while the third one was calculated. Thereby, the GAN learned the distribution of the amount of BTC received and the balance, and the amount of BTC sent was computed a-posteriori. In the same way, the uniqueness and the total received transactions are related, since one address is unique (1) when it is used exactly once for receiving money, otherwise is not unique (0). Following this rule, the total received transaction was used to train the GAN, and the uniqueness values was computed a-posteriori.

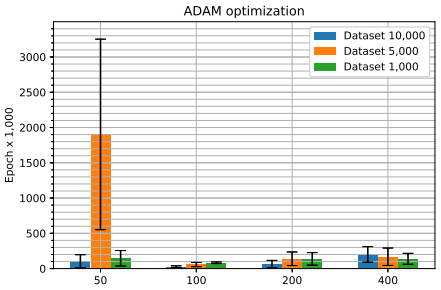
For each implementation, the number of epochs needed to train the GAN, the accuracy from D and the confidence value generated from the baseline classifier AC were calculated, as shown in Figure 3. The confidence value represents the accuracy calculated via the baseline classifier, and was used here as a metric for similarity between synthetic and real samples. The accuracy from D shows the GAN's ability to cheat the discriminator D, while the confidence value evaluates the quality of the generated samples during the classification. In order to compute the confidence value, $10,000$ synthetic samples were generated from each GAN after training. Then, these samples were de-normalized. In fact, G learned to create a distribution in the range of 0-1, as explained in Section III-B. This de-normalization was done by inverting Equation 4 for each feature value. In this manner, the 0-1 distribution was expanded to the real range of the single feature. Once the de-normalization was computed, the 2 dependent features were computed (total amount of BTC sent and uniqueness), the obtained dataset was used to feed the AC, and to ultimately compute the confidence value (Figure 3).

V. Results

Figure 4 shows that the number of epochs for training a GAN using the ADAM optimizer was generally less than $200,000$ epochs. More unstable results were generated for the RMSProp optimizer, where it was not possible to establish a fixed range. Moreover, in terms of training time, the RMSProp configuration showed low repeatibility, which was reflected by its high values of standard deviation (Figure 4a). Meanwhile, the ADAM optimizer presented high variability only for the combination of batch size = 50 and dataset size = $5,000$, which can be considered as an outlier (Figure 4b).



(a) Epoch average for training GANs with RMSProp optimization function



(b) Epoch average for training GANs with ADAM optimization function

Fig. 4: Epoch average and variance with respect to batch size in training GAN

Generally, the GANs implemented with the ADAM optimizer reached a solution faster than the ones using the RMSProp optimizer, no matter what values of batch size and dataset size were chosen, as shown in Figure 4.

# Test set	Batch size	Dataset 10,000 Acc. D	Dataset 10,000 Acc. AC	Dataset 5,000 Acc. D	Dataset 5,000 Acc. AC	Dataset 1,000 Acc. D	Dataset 1,000 Acc. AC
1	400	1.0	0.48	0.99	0.03	1.0	0.95
2	400	1.0	0.23	1.0	0.08	0.99	0.44
3	400	0.92	0.06	0.96	0.13	0.97	0.63
1	200	1.0	0.07	1.0	0.23	1.0	0.18
2	200	1.0	0.07	0.99	0.56	1.0	0.12
3	200	1.0	0.09	1.0	0.04	1.0	0.26
1	100	1.0	0.06	1.0	0.07	1.0	0.08
2	100	1.0	0.09	0.99	0.04	1.0	0.13
3	100	0.98	0.08	1.0	0.73	1.0	0.13
1	50	0.97	0.13	0.99	0.01	1.0	0.04
2	50	1.0	0.86	1.0	0.01	1.0	0.06
3	50	1.0	0.12	0.95	0.14	0.92	0.05

TABLE III: Comparison between the accuracy computed by the discriminator D and the respective accuracy from the AC classifier fed with 10,000 samples generated from GANs with RMSProp optimizer

# Test set	Batch size	Dataset 10,000 Acc. D	Dataset 10,000 Acc. AC	Dataset 5,000 Acc. D	Dataset 5,000 Acc. AC	Dataset 1,000 Acc. D	Dataset 1,000 Acc. AC
1	400	0.99	0.05	0.91	0.57	1.0	0.77
2	400	0.97	0.03	0.99	0.18	0.98	0.98
3	400	1.0	0.40	0.95	0.86	0.98	0.60
1	200	0.95	0.34	0.94	0.13	0.99	0.74
2	200	0.99	0.32	0.94	0.79	1.0	0.54
3	200	1.0	0.21	1.0	0.63	0.99	0.66
1	100	1.0	0.13	0.93	0.36	0.95	0.67
2	100	1.0	0.52	0.91	0.13	1.0	0.57
3	100	0.93	0.06	0.95	0.32	0.98	0.87
1	50	0.93	0.20	1.0	0.03	0.91	0.09
2	50	1.0	0.06	1.0	0.01	1.0	0.06
3	50	0.98	0.16	0.98	0.06	1.0	0.12

TABLE IV: Comparison between the accuracy computed from the discriminator D and the respective accuracy from the AC classifier fed with 10,000 samples generated from GANs with ADAM optimizer

Table III shows the discriminator D accuracy and the baseline AC accuracy computed based on an input of $10,000$ samples generated by each configuration of GANs that use RMSProp, whereas Table IV shows classification results of $10,000$ samples generated by GANs implemented using the ADAM optimizer.

It is evident from Table III that although RMSprop implementations presented high values of discriminator D accuracy, they showed generally poor confidence values computed with the baseline classifier (AC) based on real data. This situation is highlighted in Figure 5a-5c, which shows that the similarity between the generated synthetic data and the real data used for training AC is very low. In fact, Figure 5a-5c shows only a few high values but with low repeatibility. The best configuration for the RMSProp optimizer was obtained with dataset size = $1,000$ and batch size = 400, reaching confidence values of 0.95, 0.44 and 0.63 (respectively with discriminator D accuracy of 1.0, 0.99 and 0.97).

Figure 5d-5f highlights the benefit of using the ADAM classifier. In fact, in this case, more solutions presented high confidence values and good repeateability. From Figure 5d-5f it becomes clear that the dataset size was crucial - the solutions with the best confidence values were obtained with a small dataset size $(1,000)$. For the ADAM implementation, the configuration that ensured high accuracy and repeateability was obtained with the same configuration that achieved best results for RMSProp (dataset size = $1,000$ and batch size = 400). With these settings, the GAN showed respectively 0.77, 0.98 and 0.60 for AC accuracy, and 1.0, 0.98 and 0.98 for discriminator D accuracy.

It is to be noted that there seemed to be a relation between the number of training epochs and the result with the best confidence values. In Figure 4, the best configuration with RMSProp was obtained by training the model in about $500,000$ epochs, meanwhile for the ADAM optimizer, the best configuration was achieved with only $150,000$ epochs, which is more than 3.3 times faster. The number of training epochs for the best ADAM configuration represented also the lowest value among the ADAM configurations with the same batch size.

VI. Conclusions and Future work

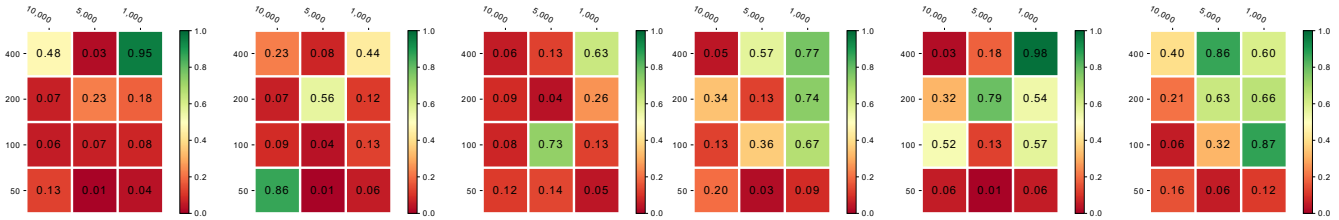
In image based data augmentation the similarity among the synthetic and the real samples can be easily verified by a normal visualization. Nevertheless, when the dataset represents features (for example in Bitcoin classification problem) the validation of the similarity among the data results complicated. In this scenario, our paper analyses how GAN configuration parameters - optimizer function, size of the dataset and batch size - affect the training of a GAN able to generate synthetic Bitcoin address samples for data augmentation. Our approach allowed us to determine the best setting for the GAN, thus ensuring high similarity between synthetic and the real samples of a critical Bitcoin class that typically presents with few samples (Mining Pool).

Our results showed that GANs implemented with the ADAM optimizer found solutions faster than GANs using the RMSProp optimizer. Moreover, GANs using the ADAM optimizer presented high repeatibility in terms of training epochs and in terms of confidence value (baseline model accuracy testing with synthetic data; our measure for similarity between synthetic and real data), which were used as a metric for similarity between synthetic and real samples. It is to be noted that decreasing the dataset size for GANs with the ADAM optimizer helped find solutions with high confidence values, while smaller batch size negatively affected confidence values. On the other hand, the introduced approach presents limitations in the generation of synthetic samples using larger datasets, in fact in these cases the performed similarity is very low, regardless of the batch size and the optimizer.

Results of this study demonstrate that with the correct configuration, GANs represent a robust and efficient tool allowing for generating good synthetic data that is similar to real Bitcoin address data. In this way, GANs could be used to resolve the imbalance problem related to Bitcoin address datasets. In future work, we aim to demonstrate that the found settings generate high-quality synthetic data for other classes that are affected by the class imbalance problem (such as the Service and the Mixer) as well. Ultimately, it will be interesting to use these synthetic data to train a new general classifier and test it with real data across several classes in order to measure the effect of Bitcoin data augmentation with GANs. Based on our results, we expect a significant improvement of classification results for underrepresented classes, which could eventually improve anomaly detection within the Bitcoin network.

ACKNOWLEDGMENTS

This work was partially funded by the European Commission through the Horizon 2020 research and innovation program, as part of the "TITANIUM" project (grant agreement No 740558).



(a) RMSProp first repetition (b) RMSProp second repetition (c) RMSProp third repetition (d) ADAM first repetition (e) ADAM second repetition (f) ADAM third repetition

Fig. 5: Confidence values (measure for similarity between synthetic and real data) calculated with the baseline model AC fed with $10,000$ synthetic samples from GANs. X-axis shows influence of dataset size; Y-axis shows influence of batch size.

References

[1] Bartoletti, M., Pes, B., Serusi, S.: Data mining for detecting bitcoin ponzi schemes. In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 75–84. IEEE (2018)

[2] Boshmaf, Y., Al Jawaheri, H., Al Sabah, M.: Blocktag: design and applications of a tagging system for blockchain analysis. In: IFIP International Conference on ICT Systems Security and Privacy Protection. pp. 299–313. Springer (2019)

[3] Chawla, N., Japkowicz, N., Kolcz, A.: Editorial: special issue on learning from imbalanced data sets, sigkdd explorations 6 (1): 1–6 (2004)

[4] Crosby, M., Pattanayak, P., Verma, S., Kalyanaraman, V., et al.: Blockchain technology: Beyond bitcoin. Applied Innovation **2**(6-10), 71 (2016)

[5] Douzas, G., Bacao, F.: Effective data generation for imbalanced learning using conditional generative adversarial networks. Expert Systems with applications **91**, 464–471 (2018)

[6] Fernández, A., García, S., Galar, M., Prati, R.C., Krawczyk, B., Herrera, F.: Learning from imbalanced data sets. Springer (2018)

[7] Foley, S., Karlsen, J.R., Putniņš, T.J.: Sex, drugs, and bitcoin: How much illegal activity is financed through cryptocurrencies? The Review of Financial Studies **32**(5), 1798–1853 (2019)

[8] Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., Greenspan, H.: Synthetic data augmentation using gan for improved liver lesion classification. In: 2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018). pp. 289–293. IEEE (2018)

[9] Goodfellow, I.: Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160 (2016)

[10] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)

[11] Harlev, M.A., Sun Yin, H., Langenheldt, K.C., Mukkamala, R., Vatrapu, R.: Breaking bad: De-anonymising entity types on the bitcoin blockchain using supervised machine learning. In: Proceedings of the 51st Hawaii International Conference on System Sciences (2018)

[12] Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1125–1134 (2017)

[13] Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)

[14] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[15] Lei, K., Xie, Y., Zhong, S., Dai, J., Yang, M., Shen, Y.: Generative adversarial fusion network for class imbalance credit scoring. Neural Computing and Applications pp. 1–12 (2019)

[16] Liang, J., Li, L., Chen, W., Zeng, D.: Targeted addresses identification for bitcoin with network representation learning. In: 2019 IEEE International Conference on Intelligence and Security Informatics (ISI). pp. 158–160. IEEE (2019)

[17] Lin, Y.J., Wu, P.W., Hsu, C.H., Tu, I.P., Liao, S.w.: An evaluation of bitcoin address classification based on transaction history summarization. In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 302–310. IEEE (2019)

[18] Liu, X.Y., Wu, J., Zhou, Z.H.: Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **39**(2), 539–550 (2008)

[19] Mariani, G., Scheidegger, F., Istrate, R., Bekas, C., Malossi, C.: Bagan: Data augmentation with balancing gan. arXiv preprint arXiv:1803.09655 (2018)

[20] Merino, T., Stillwell, M., Steele, M., Coplan, M., Patton, J., Stoyanov, A., Deng, L.: Expansion of cyber attack data from unbalanced datasets using generative adversarial networks. In: International Conference on Software Engineering Research, Management and Applications. pp. 131–145. Springer (2019)

[21] Minaee, S., Abdolrashidi, A.: Finger-gan: Generating realistic fingerprint images using connectivity imposed gan. arXiv preprint arXiv:1812.10482 (2018)

[22] Nagarajan, V., Kolter, J.Z.: Gradient descent gan optimization is locally stable. In: Advances in Neural Information Processing Systems. pp. 5585–5595 (2017)

[23] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep., Manubot (2019)

[24] Ranshous, S., Joslyn, C.A., Kreyling, S., Nowak, K., Samatova, N.F., West, C.L., Winters, S.: Exchange pattern mining in the bitcoin transaction directed hypergraph. In: International Conference on Financial Cryptography and Data Security. pp. 248–263. Springer (2017)

[25] Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: Proceedings of the IEEE international conference on neural networks. vol. 1993, pp. 586–591. San Francisco (1993)

[26] Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)

[27] Shin, H.C., Tenenholtz, N.A., Rogers, J.K., Schwarz, C.G., Senjem, M.L., Gunter, J.L., Andriole, K.P., Michalski, M.: Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In: International Workshop on Simulation and Synthesis in Medical Imaging. pp. 1–11. Springer (2018)

[28] Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning **4**(2), 26–31 (2012)

[29] Wang, K., Gou, C., Duan, Y., Lin, Y., Zheng, X., Wang, F.Y.: Generative adversarial networks: introduction and outlook. IEEE/CAA Journal of Automatica Sinica **4**(4), 588–598 (2017)

[30] Wu, E., Wu, K., Cox, D., Lotter, W.: Conditional infilling gans for data augmentation in mammogram classification. In: Image Analysis for Moving Organ, Breast, and Thoracic Images, pp. 98–106. Springer (2018)

[31] Zhang, Y., Wang, D.: A cost-sensitive ensemble method for class-imbalanced datasets. In: Abstract and applied analysis. vol. 2013. Hindawi (2013)

[32] Zola, F., Bruse, J.L., Eguimendia, M., Galar, M., Orduna Urrutia, R.: Bitcoin and cybersecurity: temporal dissection of blockchain data to unveil changes in entity behavioral patterns. Applied Sciences **9**(23), 5003 (2019)

[33] Zola, F., Eguimendia, M., Bruse, J.L., Urrutia, R.O.: Cascading machine learning to attack bitcoin anonymity. In: 2019 IEEE International Conference on Blockchain (Blockchain). pp. 10–17. IEEE (2019)