UCLouvain

École polytechnique de Louvain

Tabular Data Synthesis using Generative Adversarial Networks

An application to Table Augmentation

Author: **Edouard Couplet**
Supervisors: **John Lee, Michel Verleysen**
Reader: **Anne-Sophie Collin**
Academic year 2020–2021
Master [120] in Data Sciences Engineering

Abstract

In this master thesis, we design an efficient tabular data synthesizer and study the use of synthetic data for table augmentation. While tabular data is the most common data modality, qualitative tabular data is not always easy to obtain or access, hence the need for synthetic data. Nevertheless, because of its high complexity, synthesizing tabular data is not an easy task. The main difficulties lie in the simultaneous processing of numerical and categorical columns, and in the modeling of the intricate relationships between them. State-of-the-art tabular data synthesizers are based on generative adversarial networks (GANs). In particular, CTGAN [1] introduces several techniques to deal with complex multi-modal numerical columns and uses one-hot encoding to represent categorical attributes. However it struggles to effectively capture associations between columns. In this thesis, we propose an enhanced version of CTGAN with a novel encoding-decoding structure as an alternative to one-hot encoding. Through rigorous evaluation, we show that it significantly improves the quality of the synthesized data. We notably achieve a 55% increase in machine learning efficacy and obtain encouraging results for data augmentation in the context of imbalanced learning.

Acknowledgements

I would like to thank my two advisors, John LEE and Michel VERLEYSEN, for their precious guidance throughout the realization of this thesis and for their enthusiasm during our punctual meetings.

I would also like to give special thanks to Camille DRAGUET for her constant understanding and support, for her very helpful proofreading work, and especially for making sure that I did not turn into a sleep-deprived zombie while writing the manuscript.

Finally, a warm thank you to Anne-Sophie COLLIN for having accepted to be part of the jury as a reader.

Contents

Introduction

Data has always been a valuable resource, and this is even more true nowadays with the relentless progress of artificial intelligence and machine learning (ML). With cleverly designed ML algorithms, crucial knowledge can be extracted from data and this knowledge is at the heart of many important decision-making processes, whether in industry, politics or in the scientific community.

As any resource, data needs to be stored in an organized way. A very direct and efficient way to proceed is with tables: these are the well-known data structures composed of rows and columns, such as Excel spreadsheets for instance. Data stored under this format is called tabular data and is probably the most common data modality. Nevertheless, qualitative tabular data is not easy to come by. On one hand, because it is valuable, it is usually owned by private entities which can restrict and/or monetize its access. On the other hand, when the data is public, it is governed—and rightly so—by strict privacy policies that may greatly limit its usefulness. Besides, even if qualitative data is available, quantity can also be an issue. Indeed, ML algorithms typically need a very large amount of data to ensure acceptable performance and there are many domains where only few data is available. This can be the case in healthcare for example, with data related to patients suffering from rare conditions, or in survey data, when the population sample is relatively small. One possible way to address these accessibility and quantity issues is to leverage synthetic data, *i.e.*, artificially created data that presents the same properties as some given existing data.

However, the production of qualitative synthetic tabular data is not an easy task. This is mainly due to the difficulty of modelling the complex relationships between the columns of a table, but also to the wide variety of data types that these different columns may contain. Categorical data is hard to deal with due to its very nature, and numerical data can be hard to process when it follows complex non-Gaussian-like distributions. As a result, there is still no unanimously accepted generative model for tabular data synthesis. This is quite different for image data synthesis. Indeed, since they were introduced by I. J. GoodFellow et al. [2] in 2014, generative adversarial networks (GANs) have literally taken over the field of image generation. Their incredible success is due to their ability of accurately modelling the underlying distribution of a given set of real images, and then sampling from this distribution to produce realistic looking images that could convincingly belong to the same set. A question that begs to be asked is then: since images are nothing more to a computer than big tables of numbers corresponding to pixel values, couldn't we apply GANs to tabular data?

Yes, we could. It is not without any difficulties, but we can definitely use GANs to synthesize tabular data, and this is exactly what we will do in this thesis.

Of course, we are not the first ones to have had this idea and over the past couple of years, a few approaches using GANs for tabular data synthesis have been proposed in the literature. One state-of-the-art approach is CTGAN [1]. It generates synthetic rows of a given table using a conditional Wassertein GAN framework and introduces several methods such as *mode-specific normalization* to deal with complex numerical columns. Moreover, CTGAN deals with categorical columns by encoding their values as one-hot vectors. Such an encoding is simple and efficient, yet not very meaningful. In particular, it does not consider how one category relates to another.

Our goal is therefore to improve the CTGAN synthesizer. Our key contribution is a novel approach for encoding categorical values inspired by word embeddings from the domain of language modelling. We also make a few additions based on techniques used in some other state-of-the-art methods such as the extended conditional vector, the *feature matching loss*, and the *prediction loss*. We rigorously compare our enhanced version of CTGAN with the original CTGAN on several datasets and several metrics. These include metrics that measure the statistical similarity of the synthetic data to the real data, and machine learning efficacy metrics that measure how well the synthetic data can be used in place of the real data in a machine learning application. We show that our enhanced CTGAN outperforms the original CTGAN on all but one metric (on which they perform about the same). In particular, it increases the machine learning efficacy of the synthetic data by 55%.

Finally, we want to see if GAN-based tabular data synthesizers can be used for tabular data augmentation. To this end, we design several scenarios where we augment datasets with different proportion of synthetic data and then train classifiers or regressors on the augmented data depending on the learning task at hand. In general, we were not able to obtain better results on the test data for models trained on augmented data. However, we have obtained encouraging results that tend to show that our enhanced CTGAN could be used efficiently to oversample minority classes in the context of imbalanced learning.

This manuscript is structured into 6 chapters. In the first chapter, we give some general information on tabular data and data synthesis. In chapter 2, we explain how GANs work both from a theoretical and practical point of view. We introduce the vanilla GAN, the conditional GAN and the Wassertein GAN. In chapter 3, we expose the challenges related to the synthesis of tabular data with GANs and briefly look at how existing approaches tackle these challenges. Based on our observations, we introduce our own approach. In chapter 4, we develop in detail the CTGAN model and our enhanced CTGAN model. In chapter 5, we explain our experimental set up and our evaluation metrics both for data synthesis and data augmentation. In the last chapter, we compare and discuss the results of CTGAN and our enhanced CTGAN for the task of tabular data synthesis. Moreover, we analyze how the enhanced CTGAN model performs in the context of data augmentation.

Chapter 1

Tabular data and data synthesis

1.1 Tabular data

Tabular data is data that is structured into rows and columns. Each row represents some *thing* and contains bits of information about it. These bits of information are captured in some defined attributes and recorded in cells, each of which contains the value of one attribute. Tabular data is only useful if all *things* are of the same nature and described by the same attributes : it follows that each row has the same number of cells and that cells within a same column represent the same attribute. Medical records, for example, are often stored in tabular form : each row corresponds to a patient and contains health and demographic information. Some common attributes are the patient's name, address, weight, blood pressure, heart rate, etc.

Such a tabular structure allows to store and process a large amount of data and, more notably, a wide variety of data. This versatility implies that tabular data can have various properties, depending on what pieces of information we want to store and process.

1.1.1 Table-wise properties

A table can have any shape, there is no constrain on the number of rows nor the number of columns (except for the fact that if a table has only one row or one column, it cannot really be identified as a table anymore). When dealing with census data, for example, there is a good chance that the population sample is very large relative to the number of attributes recorded. In this case, the number of rows will be much greater than the number of columns. In contrast, when performing gene expression analysis with human subjects, the sample size will be relatively small because genetic profiling is very expensive. Moreover, the number of attributes will be extremely large, as one attribute typically corresponds to one gene (Homo Sapiens has about 25000 of them). As a result, the number of rows will be much smaller than the number of columns.

Besides, a table can be incomplete and contain empty cells. In the case of census data, this may happen if pieces of information such as the birth-date or the income are not available for some individuals of the sample population. It may also be that some part of the data is unreliable and is therefore intentionally suppressed. In the example of gene expression analysis, it may not be possible to reliably measure the expression of a gene because of sub-optimal experimental conditions (or because of some other reason). Scientists may therefore decide to exclude unreliable measurements to preserve the quality of the data.

A simple way to solve this issue is to replace the missing values with the mean or the median value of the corresponding attribute. This trick is computationally efficient but far from ideal[1]. Fortunately, there exists a large literature on the subject with more elaborate and far better solutions.

Also, note that tabular data is not always in the form of a single table, as this may not be sufficient to effectively store and process real-world data, nor to capture its complexity. Most modern databases contain multiple tables which are organized and interact with one another according to some conceptual model[2].

1.1.2 Row-wise properties

Rows can be dependent on one another. This is the case when working with time-series data such as sensor data or market stock prices. When put in tabular form, each row contains data related to a given point t in time. The values of the attributes in the row corresponding to time t may then be dependent on the values of the same attributes in some other rows corresponding to time points anterior to t. If the rows are ordered chronologically - which would make a lot of sense - these correspond to previous rows in the table.

When there are no clear reasons to believe that rows of some tabular data are dependent on one another, we generally assume that the rows are independent, even if true independence is rarely met in real world settings. According to this assumption, all rows can be thought of as sampled from the same joint distribution of their attributes.

1.1.3 Column-wise properties

Columns can be all independent of one another but that would make data far less interesting. In fact, finding relationships and patterns among attributes is usually the main purpose of tabular data analysis and the reason why we collect tabular data in the first place, besides keeping a trace of relevant bits of information. The power of data comes from the knowledge than can be extracted from it, and that knowledge is hidden somewhere in the columns and in the way they interact together.

However, this knowledge is not easy to extract because columns can interact in infinitely many ways, and most of them are non trivial. Moreover, columns can contain data of many different types depending on the attribute that is represented. The most basic distinction is made between categorical data and numerical data.

Categorical data relates to qualitative attributes. The values that describe an attribute are called categories and can be represented as strings or integers which, in this particular case, have no strict mathematical meaning.

[1] In particular, it introduces a bias towards the mean/median value of the corresponding attribute.

[2] One such widely used conceptual model is the relational model. Tables are called relations and are defined as a set of related tuples (rows). Relationships between tables are identified and implemented using foreign keys.

Categorical data is said to be ordinal if the categories can be ranked or ordered. For example, suppose a survey where subjects have to give their opinion with respect to several proposition by choosing among five possible options : "I strongly agree","I agree", "I am neutral", "I disagree" and "I strongly disagree". In this case, it makes sense to rank the possible values on a qualitative scale that represents the agreement of the subject with the proposition; the survey data would therefore be qualified as ordinal data. If categories can only be identified by names and cannot be ordered in some way that makes sense, then the data is said to be nominal. Examples are a person's gender, religion, etc.

Numerical data relates to quantitative attributes, *i.e.*, attributes that can be measured in terms of numbers. If there is a bijection between the set of all possible values that can describe the attribute and the set of natural numbers, the data is said to be discrete and is typically represented by integers. Some examples are a person's age, number of years of education, number of children, etc. If there exists no such bijection, the data is said to be continuous and is typically represented by real numbers. Examples are a person's height or weight.

One of the attributes can be designated as the target attribute. Machine learning tasks often consist in predicting the target attribute given other attributes.

1.2 Data synthesis

Data synthesis is the action of creating data. This new data can either be created without any connection to some already existing data or, on the contrary, in close relation to some given real world data. In the first case, one can easily generate somewhat arbitrary data using the adequate statistical tools. This can be very useful, for example, to test predictive models in specific scenarios without having to find real data that matches the required conditions, which can be difficult or even sometimes impossible. The second case is more tricky and perhaps of greater interest. The goal is to generate realistic synthetic data based on some given example data. By realistic, we mean that the generated data must have, if not equal, similar properties as the example data. Several approaches of various complexities exist to achieve this goal.

A first and simple approach is to take the original data and add small perturbations to it. The assumption is that because perturbations are small, the properties of the original data will be preserved. This approach is used with success on image data. Indeed, we can take the image of a dog, add some Gaussian noise, and we will obtain a new image which still convincingly represents a dog. The perturbation could be a change of hue or luminosity, or take the form of simple geometric transformations such as a translation or a rotation; the transformed picture would still be that of a dog. However, this is not so straight forward with tabular data : how do we know for sure that we do not denature a row when perturbing it ? Besides, what would it mean to add a small perturbation to categorical data ? And in particular, to nominal data ? We could imagine only applying perturbations to numerical data but even so, the produced synthetic table would seem to be a mere copy of the original one. It may then not meet some privacy standards nor be useful for data augmentation, thereby loosing its interest.

A second approach is to combine points from the original data to produce new synthetic data points. With tabular data, this means combining attributes of some rows to produce new synthetic rows. A popular example of this approach is the Synthetic Minority Oversampling Technique, or SMOTE [3]. It works by identifying, using the euclidean distance, the k-closest neighbors of some original data points in the high dimensional space of attributes. New synthetic data points are then created on the lines joining the original points to one of their k-closest neighbors. SMOTE is designed to be used in the context of imbalanced classification[3] but techniques implementing the same principle can be used in many other situations. However, similarly to the perturbation approach, it is not so clear how to deal with categorical data. Also, there is no guarantee that combining two values of a numerical attribute will produce a realistic value for this same attribute. Consider a given attribute as a random variable and suppose that this variable is distributed as a bi-modal Gaussian distribution with clearly separated modes, that is, the probability density function is close to zero between the two modes. In this case, combining a value from one mode with a value from the other may produce some unrealistic result.

A third and more elaborate approach consists in modelling the distribution of the original data $\mathbf{x} \sim p_d(\mathbf{x})$ and then sampling from the modeled distribution to produce new synthetic samples $\tilde{\mathbf{x}} \sim p_\theta(\tilde{\mathbf{x}})$. This approach has the advantage to work both with numerical and categorical data, and to provide realistic synthetic samples on the condition that p_θ is a good approximation of p_d. Many methods based on this approach have been proposed in the literature; they mostly vary in how they construct p_θ. State of the art methods typically involve deep learning. Some common examples are variational auto encoders (VAEs)[4], deep belief networks (DBNs)[5] and generative adversarial networks (GANs)[2]. GANs can be trained without the need to explicitly define p_θ and as such, they have very few design restrictions.

This last approach is the one we follow in the present thesis. In particular, we learn a synthetizer $\mathcal{S}$ based on some existing tabular data with distribution p_d. The main component of this synthetizer is the generator network of a GAN. This generator implicitly models a distribution p_G and the goal will be to train this GAN such that p_G captures all the complexity of p_d, that is the intricate relationships between the column of the given tabular data.

[3]Suppose a binary classification problem. Tabular data is imbalanced if a majority of rows belongs to one category of the class attribute while a minority belongs to the other. The former are referred to as the majority class and the latter as the minority class. This imbalance can cause the classifier to be biased towards the majority class. The goal of SMOTE is therefore to produce extra synthetic rows belonging the the minority class in order to prevent this bias.

Chapter 2

Generative adversarial networks

GANs were first introduced by Ian J. Goodfellow and colleagues in 2014 [2]. Since then, countless variations have been explored. In this chapter, we will first describe in detail the original version of GANs, which we refer to as vanilla GAN. In particular, we will try to show why it works in theory but not so perfectly in practice. We will then study a variation called Wassertein GAN [6] which mitigates the shortcomings of the original version and which we will use in our tabular data synthesizer.

2.1 Vanilla GAN

In its simplest form, a GAN can be seen as two players competing in a game. The first player is a generative model G, called the generator. Given some noise variable $\mathbf{z}$ as input, it aims to create samples $\tilde{\mathbf{x}}_d \sim p_G$ that come from the same (unknown) distribution as some real target data $\mathbf{x}_d \sim p_d$. The second player is a discriminative model D, called the discriminator. It recieves samples $\mathbf{x}$ and tries to determine if they come from G or from the real data. The goal of G is to trick D into thinking that its samples are real while the goal of the D is to not be fooled. As the latter learns to better discriminate between real and fake generated samples, the former is encouraged to produce sample that are more convincingly real. This pushes p_G to resemble p_d. Eventually, we want to reach a point where D is incapable of distinguishing fake from real data, that is, a point where $p_G = p_d$ and therefore where G can produce samples $\tilde{\mathbf{x}}_d \sim p_d$.

More precisely, each player is represented by a function which is differentiable both with respect to its inputs and its parameters. On one hand, the generator is represented by a function $G(\mathbf{z}; \theta_G)$ which defines a mapping from the noise space to the data space and is implemented as a neural network with parameters θ_G. The input noise $\mathbf{z}$ is sampled from a defined prior distribution p_z (usually a uniform or Gaussian distribution) and the output $\tilde{\mathbf{x}}_d = G(\mathbf{z})$ corresponds to a sample from p_G. On the other hand, the discriminator is represented by a function $D(\mathbf{x}; \theta_D)$ which defines a mapping from the data space to a probability space and is implemented as a neural network with parameters θ_D. Its input $\mathbf{x}$ comes either from the real data or from G and its output $\hat{y} = D(\mathbf{x})$ represents the probability that $\mathbf{x}$ comes from the real data rather than G. When we reach a point in training where $p_G \approx p_d$, D has no better choice to guess whether $\mathbf{x}$ is real or fake; we therefore have $D(x) = \frac{1}{2}$. In the original paper (and in the present thesis), both neural networks of G and D are multi layer perceptrons (MLPs).

Figure 2.1 shows the vanilla GAN framework. Note that we can assign a label y to the input $\mathbf{x}$ of D, where y indicates whether $\mathbf{x}$ comes from p_d ($y = 1$) or from p_G ($y = 0$). The discriminator can then be interpreted as a binary classifier which aims to predict the label y of its input $\mathbf{x}$. Usually, a binary classifier does not directly predict the label but rather outputs a probability for that label with respect to one of the two classes. In our setting, it would mean that $D(\mathbf{x})$ represents the probability that $\mathbf{x}$ has label $y = 1$. This is coherent whith what we have explained in the previous paragraph : if $D(\mathbf{x})$ is close to 1, D thinks that it is likely for $\mathbf{x}$ to have label $y = 1$ and therefore that $\mathbf{x}$ comes from p_d rather than from p_G. On the contrary, if $D(\mathbf{x})$ is close to 0, D thinks that it is *not* likely for $\mathbf{x}$ to have label $y = 1$ or, in other words, that $\mathbf{x}$ likely has label $y = 0$ and therefore that it comes from p_G rather than from p_d. Finally, if $D(\mathbf{x})$ remains close to $\frac{1}{2}$, it means that D cannot really decide whether $\mathbf{x}$ has label $y = 1$ or $y = 0$ and therefore that we are reaching a point where $p_G \approx p_d$.

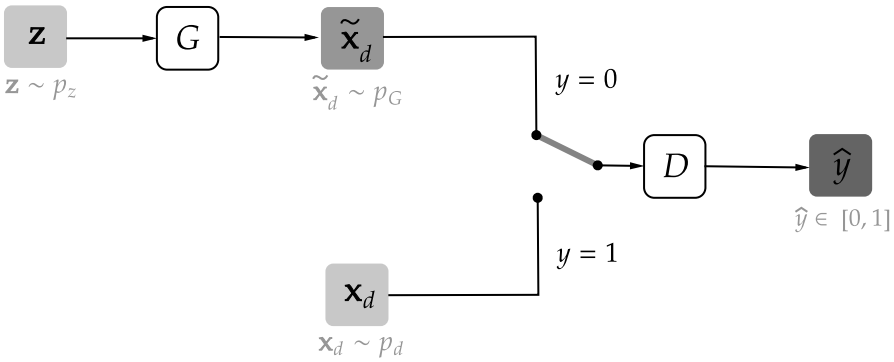


Figure 2.1: Vanilla GAN framework

2.1.1 A minimax game

If we want G to produce realistic samples, D must be trained to *maximize* the probability of assigning the correct label to $\mathbf{x}$, that is $y = 1$ to $\mathbf{x}_d$ and $y = 0$ to $\tilde{\mathbf{x}}_d$, while G must be trained in an adversarial way to *minimize* the probability that D assigns the correct label $y = 0$ to $\tilde{\mathbf{x}}_d$. Since G wants to *minimize* what D wants to *maximize*, it becomes clear that G and D are competing in a two player minimax game with a certain value function $V(G, D)$:

$$\min_G \max_D V(G, D) \tag{2.1}$$

In practice, during training, the generator and the discriminator both minimize a loss function. Because the discriminator must maximize $V(G, D)$, we can write its loss function as $\mathcal{L}_D = -V(G, D)$. The loss function of the generator, for a fixed (optimal) discriminator D, is then given by $\mathcal{L}_G = -\mathcal{L}_D = V(G, D)$.

We will now derive the expression of $V(G, D)$. To do this, we will first express the loss of discriminator then use the relation $V(G, D) = -\mathcal{L}_D$. Because D is interpreted as a binary classifier, it seems sensible to use binary cross entropy to measure the discrepancy between a prediction $\hat{y} = D(\mathbf{x})$ and the corresponding label y of $\mathbf{x}$:

$$BCE(\hat{y}, y) = -\big(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})\big) \tag{2.2}$$

Next, suppose that the discriminator receives as input a single $\mathbf{x}_d$ and a single $\tilde{\mathbf{x}}_d$; because the objective of D is to predict the correct label both when $\mathbf{x}$ comes from the real data and when $\mathbf{x}$ comes from G, we can write :

$$\mathcal{L}_D = BCE\big(D(\mathbf{x}_d), 1\big) + BCE\big(D(\tilde{\mathbf{x}}_d), 0\big) \tag{2.3}$$

Which gives, using equation 2.2:

$$\mathcal{L}_D = -\log\big(D(\mathbf{x}_d)\big) - \log\big(1 - D(\tilde{\mathbf{x}}_d)\big) \tag{2.4}$$

Since we must actually consider multiple $\mathbf{x}_d$ and $\tilde{\mathbf{x}}_d$, and because $D(\mathbf{x})$ corresponds to a probability, the discriminator will have to minimize the expectation of the previously calculated loss:

$$\mathcal{L}_D = -\mathbb{E}_{x_d \sim p_d}[\log D(\mathbf{x}_d)] - \mathbb{E}_{\tilde{x}_d \sim p_G}[\log(1 - D(\tilde{\mathbf{x}}_d))] \tag{2.5}$$

We then apply the change of variable $\tilde{\mathbf{x}}_d = G(\mathbf{z})$ to obtain:

$$\mathcal{L}_D = -\mathbb{E}_{x_d \sim p_d}[\log D(\mathbf{x}_d)] - \mathbb{E}_{z \sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{2.6}$$

The loss of the generator is simply obtained by flipping the signs. Besides, the first term disappears as it does not depend on G:

$$\mathcal{L}_G = \mathbb{E}_{z \sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{2.7}$$

Now that we have an expression for $\mathcal{L}_D$ (and $\mathcal{L}_G$), we can fill in equation 2.1 to obtain the complete formulation of the two player minimax game as described in [2]:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x_d \sim p_d}[\log D(\mathbf{x}_d)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{2.8}$$

Finally, lets check that this formulation corresponds to what we want for D and G. The discriminator has to maximize the two terms of $V(G, D)$ as they both depend on D. Since $D(\mathbf{x}) \in [0, 1]$, we have that $\log(D(\mathbf{x})) \in]-\infty, 0]$. It follows that the first term is maximal when $D(\mathbf{x}_d) = 1$, which corresponds to assigning the correct label $y = 1$ to $\mathbf{x}_d$. We can reason similarly for the second term: it is maximal when $D(G(\mathbf{z})) = 0$, which correspond to assigning the correct label $y = 0$ to $G(\mathbf{z}) = \tilde{\mathbf{x}}_d$. We thus observe that D does indeed maximize the probability of assigning the correct label to $\mathbf{x}$.

The generator has to minimize only one term: $\mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$. It is minimal when $D(G(\mathbf{z})) = 1$ which corresponds to D assigning the *incorrect* label $y = 1$ to $G(\mathbf{z}) = \tilde{\mathbf{x}}_d$. Therefore, G does minimize the probability of D assigning the correct label $y = 0$ to $\tilde{\mathbf{x}}_d$.

2.1.2 Conditional variation

Suppose $\mathbf{x}_d$ are samples drawn from the famous MNIST dataset [7]. This dataset is a collection of 28×28 black and white images representing the digits zero through nine. Each $\mathbf{x}_d$ is thus a vector of $\mathbb{R}^{784}$ containing the pixel values of one image. Each $\mathbf{x}_d$ also has a class label $c \in \{0, 1, ..., 8, 9\}$ corresponding to the digit it represents. Moreover, suppose that we have trained a vanilla GAN such that we can generate convincingly realistic images of digits. In this case, even if we can produce samples $\tilde{\mathbf{x}}_d \sim p_d$, we cannot choose specifically the class of the generated samples. In other words, we cannot arbitrarily generate an image of a given digit.

Depending on the situation, we might want to direct the generation process such that G produces a sample from a given class, or more generally, a sample from a given mode of the data distribution. This can be done by conditioning both G and D on some vector $\mathbf{c}$ indicating in some way the mode to generate. This vector is called the conditional vector. Such a conditional GAN framework is represented on Figure 2.2. The objective function of the two player minimax game becomes :

$$\min_G \max_D V(G, D) = \mathbb{E}_{x_d \sim p_d}[\log D(\mathbf{x}_d|\mathbf{c})] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(\mathbf{z}|\mathbf{c})))] \tag{2.9}$$

The idea is that if the generator produces samples $\tilde{\mathbf{x}}_d$ that do not match the mode given by $\mathbf{c}$, the discriminator will more easily classify them as false, and so even if $\tilde{\mathbf{x}}_d \sim p_d$. This mechanism thus encourages G to produce realistic samples of given modes.

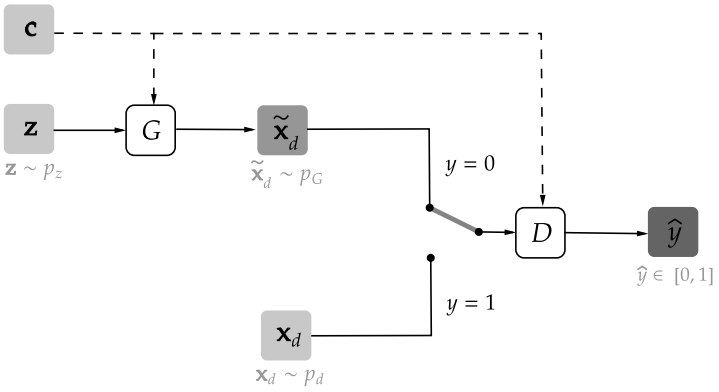


Figure 2.2: Conditional GAN framework

Interestingly, the idea of a conditional GAN was already mentioned in the original paper[2] and was formalized soon after by M. Mirza and S.Osindro in [8]. The authors successfully trained a conditional GAN on the MNIST dataset to produce images of specific digits. They conditioned G and D on the class labels encoded as one hot vectors. Hence $\mathbf{c}$ is a 10-dimensional vector with a 1 at the position corresponding to the class label and 0 elsewhere. Practically, they implemented G and D as MLPs. In the generator, they first mapped $\mathbf{z}$ and $\mathbf{c}$ to individual hidden layers before feeding both activated outputs to a second common hidden layer and then to the rest of the network. In the discriminator, $\mathbf{x}$ and $\mathbf{c}$ are treated similarly.

2.1.3 Training procedure

So far, we have derived the loss functions of G and D and showed how they relate to the formulation of a two players minimax game. However, we have not yet explained how to train a GAN from an algorithmic point of view. The training procedure, as described in [2], is given in Algorithm 1.

Since G and D are implemented as (deep) neural networks, the learning occurs through gradient back-propagation. The training takes place in several iterations; at each iteration, we first train the discriminator then the generator. In theory, D should be trained to optimality in order to provide more reliable gradients to G. This would be computationally prohibitive so instead D in trained only for k steps. Its parameters θ_D are updated using the gradient $\nabla_{\theta_D}\mathcal{L}_D(\theta_G, \theta_D)$. G is trained in a single step and its parameters θ_G are updated using the gradient $\nabla_{\theta_G}\mathcal{L}_G(\theta_G, \theta_D)$. The updates can be made following any standard gradient-based learning rule. The most simple approach is vanilla gradient descent: at some iteration j, we compute $\theta_j = \theta_{j-1} - \alpha\nabla_{\theta_j}$ where α is the learning rate. In [2], the authors use momentum based gradient descent.

Note that because training is inherently a discrete numerical process, we use the discrete expectation in the expressions of $\mathcal{L}_D$ and $\mathcal{L}_G$ given in equations 2.6 and 2.7. Therefore, given a batch of m noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ and a batch of m data samples $\{\mathbf{x}_d{}^{(1)}, ..., \mathbf{x}_d{}^{(m)}\}$, we have :

$$\mathcal{L}_D(\theta_G, \theta_D) = \frac{1}{m}\sum_{i=1}^{m}\Big(-\log\big(D(\mathbf{x}_d{}^{(i)}; \theta_D)\big) - \log\big(1 - D(G(\mathbf{z}^{(i)}; \theta_G); \theta_D)\big)\Big) \tag{2.10}$$

$$\mathcal{L}_G(\theta_G, \theta_D) = \frac{1}{m}\sum_{i=1}^{m}\Big(\log\big(1 - D(G(\mathbf{z}^{(i)}; \theta_G); \theta_D)\big)\Big) \tag{2.11}$$

Algorithm 1: Batch stochastic gradient descent training of GANs

for number of training iterations **do**
  Training of the discriminator:
  for k steps **do**
    Sample a batch of m noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from noise prior p_z
    Sample a batch of m data samples $\{\mathbf{x}_d{}^{(1)}, ..., \mathbf{x}_d{}^{(m)}\}$ from p_d
    Update D by descending its stochastic gradient $\nabla_{\theta_D}\mathcal{L}_D$
  end
  Training of the generator:
  Sample a batch of m noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from noise prior p_z
  Update G by descending its stochastic gradient $\nabla_{\theta_G}\mathcal{L}_G$
end

Figure 2.3 illustrates nicely the training procedure of GANs on a one-dimensional theoretical example. In this example, $\mathbf{x}_d$ are samples drawn from a uni-modal Gaussian distribution. In (a), we have reached a state where training has nearly converged, *i.e.*, we have p_G similar to p_d. The discriminator is relatively accurate but not yet optimal given p_G and p_d. We must thus first train D until we obtain an optimal classifier (b). Next, we update the generator (c). We can observe that p_G is pushed towards p_d, meaning that samples $\tilde{\mathbf{x}}_d \sim p_G$ will more likely be classified as real data by D. Finally, after a number of iterations, training fully converges and we reach a point where $p_G = p_d$ and thus D is unable to distinguish $\tilde{\mathbf{x}}_d$ from $\mathbf{x}_d$ ($D(x) = \frac{1}{2}$).

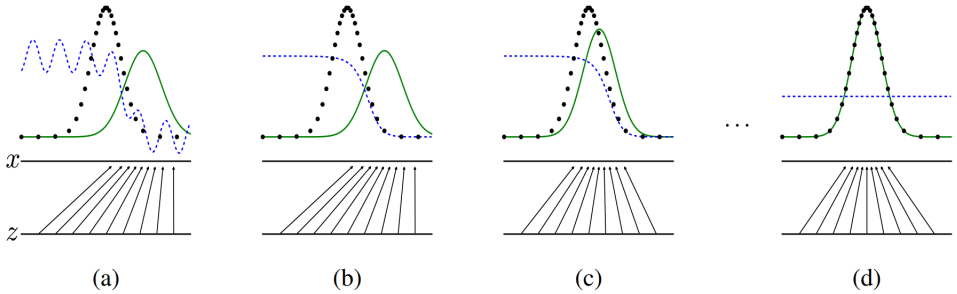


Figure 2.3: GAN training (in theory). The black dotted line represents p_d (p_d is generally unknown, we only have samples from it). The green solid line represents p_G, the distribution implicitly modeled by the generator. The blue dashed line represents the output of the discriminator $D(x)$. The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows represent the mapping $x = G(z)$. Reproduced from I.J. Goodfellow et al. [2].

Figure 2.4 represents the same one-dimensional example but in practice, with simulated data. One major difference is that we do not train the discriminator in (b) until it is optimal, instead, we train it for k steps as specified in Algorithm 1. Also, we observe that we do not reach a point of perfect convergence in (d). Nevertheless, for this simple example, the practice seems to match the theory quite well. This is unfortunately far from being a general rule. In the next sections, we will explore why GANs work fine in theory, but fail in many practical cases. More importantly, we will also look at some solutions.

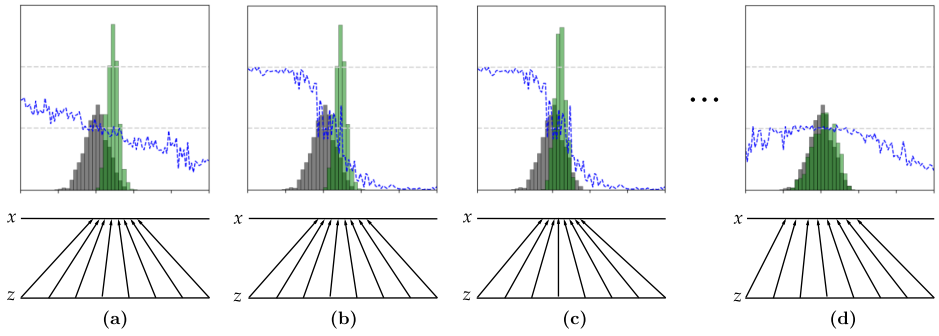


Figure 2.4: GAN training (in practice). The black histogram represents samples from the simulated data $x_d \sim p_d$. The green histogram represents generated samples $\tilde{x}_d \sim p_G$.

2.1.4 Theoretical backround

The main purpose of the original GAN paper [2], other than to present a groundbreaking generative model, is to convince the reader that the idea of GANs is mathematically sound. Indeed, a significant part of the paper is dedicated to proving that, in theory and under some assumptions, GANs work. The authors develop their argument in two points: first, they show that the global minimum of the generator's training criterion is achieved if and only if $p_G = p_d$. Then, they demonstrate that Algorithm 1 converges to this minimum.

We will briefly analyse these two points as they provide important insights on how GANs work. In order to obtain the desired results, an assumption is made that G and D have infinite capacity, this means in particular that the generator can recover any p_G. We thus consider that we are working directly in the space of probability density functions rather than in the space of network parameters.

Global minimum

Based on equation 2.5 and the relation $V(G, D) = -\mathcal{L}_D$, we can write :

$$\begin{aligned} V(G, D) &= \mathbb{E}_{x_d \sim p_d}[\log D(\mathbf{x}_d)] + \mathbb{E}_{\tilde{x}_d \sim p_G}[\log(1 - D(\tilde{\mathbf{x}}_d))] \\ &= \int_{\mathcal{X}_d} p_d(\mathbf{x}_d) \cdot \log D(\mathbf{x}_d) d\mathbf{x}_d + \int_{\tilde{\mathcal{X}}_d} p_G(\tilde{\mathbf{x}}_d) \cdot \log(1 - D(\tilde{\mathbf{x}}_d)) d\tilde{\mathbf{x}}_d \end{aligned} \tag{2.12}$$

where $\mathcal{X}_d$ is the domain of $\mathbf{x}_d$ and $\tilde{\mathcal{X}}_d$ the domain of $\tilde{\mathbf{x}}_d$. Since $\mathcal{X}_d$ and $\tilde{\mathcal{X}}_d$ actually correspond to the same domain $\mathcal{X}$, we have:

$$V(G, D) = \int_{\mathcal{X}} \big(p_d(\mathbf{x}) \cdot \log D(\mathbf{x}) + p_G(\mathbf{x}) \cdot \log(1 - D(\mathbf{x}))\big) d\mathbf{x} \tag{2.13}$$

Recall that the discriminator has to maximize $V(G, D)$. The integral in equation 2.13 is maximal if D is such that the integrand is maximal for all $\mathbf{x} \in supp(p_G) \cup supp(p_d) \subseteq \mathcal{X}$[1]:

$$\begin{aligned} 0 &= \frac{\partial}{\partial D} \big(p_d(\mathbf{x}) \cdot \log D(\mathbf{x}) + p_G(\mathbf{x}) \cdot \log(1 - D(\mathbf{x}))\big) \\ 0 &= \frac{p_d(\mathbf{x})}{D(\mathbf{x})} - \frac{p_G(\mathbf{x})}{1 - D(\mathbf{x})} \end{aligned}$$

For a given G, the optimal discriminator D^*_G is thus given by:

$$D^*_G = \frac{p_d(\mathbf{x})}{p_d(\mathbf{x}) + p_G(\mathbf{x})} \tag{2.14}$$

Now, recall that the generator has to minimize $\max_D V(G, D)$. Putting equations 2.13 and 2.14 together, we obtain that G has to minimize:

$$V(G, D^*_G) = \int_{\mathcal{X}} \left[p_d(\mathbf{x}) \cdot \log \left(\frac{p_d(\mathbf{x})}{p_d(\mathbf{x}) + p_G(\mathbf{x})} \right) + p_G(\mathbf{x}) \cdot \log \left(\frac{p_G(\mathbf{x})}{p_d(\mathbf{x}) + p_G(\mathbf{x})} \right) \right] d\mathbf{x} \tag{2.15}$$

[1] $supp(f)$ is the support of f. It is defined as the part of the domain of f where f is non-zero : $supp(f) = \{x \in X | f(x) \neq 0\}$

Which we can reformulate as:

$$V(G, D_G^*) = \int_{\mathcal{X}} p_d(\mathbf{x}) \cdot \log\left(\frac{p_d(\mathbf{x})}{\frac{1}{2}(p_d(\mathbf{x}) + p_G(\mathbf{x}))}\right) d\mathbf{x} + \int_{\mathcal{X}} p_G(\mathbf{x}) \cdot \log\left(\frac{p_G(\mathbf{x})}{\frac{1}{2}(p_d(\mathbf{x}) + p_G(\mathbf{x}))}\right) d\mathbf{x}$$
$$\underbrace{-\log(2) \cdot \int_{\mathcal{X}} p_d(\mathbf{x}) d\mathbf{x} - \log(2) \cdot \int_{\mathcal{X}} p_G(\mathbf{x}) d\mathbf{x}}_{=-\log(4)}$$

Using the definitions[2] of the Kullback–Leibler divergence (KL) and the Jensen–Shannon divergence (JS), this gives us:

$$V(G, D_G^*) = \quad KL\big(p_d \parallel \frac{1}{2}(p_d + p_G)\big) + KL\big(p_G \parallel \frac{1}{2}(p_d + p_G)\big) - \log(4) \tag{2.16}$$

$$V(G, D_G^*) = \quad 2 \cdot JS(p_d \parallel p_G) - \log(4) \tag{2.17}$$

We observe that GANs actually minimize the JS divergence between p_G and p_d. This is a relevant criterion to optimize for pushing p_G towards p_d and thus achieving $\tilde{\mathbf{x}}_d \sim p_d$. This is what makes vanilla GANs a mathematically sound idea. In particular, equation 2.17 has a unique minimum when $JS(p_d \parallel p_G) = 0$, that is, when $p_G = p_d$. We have thus found our first result according to which the global minimum of the generator's training criterion is reached if and only if $p_G = p_d$. We will now give an intuition on why Algorithm 1 converges to this global minimum.

Convergence of the training procedure

Under the assumption that G and D have infinite capacity, we are able to train D to optimality in the inner loop of Algorithm 1. The training procedure therefore consists in optimizing the criterion $V(G, D_G^*)$ which we now know has a global optimum in $p_G = p_d$. This criterion can be seen as a convex function u of p_G. Algorithm 1 is then equivalent to applying gradient descent on u: given sufficiently small updates of p_G, we will converge to its global minimum, *i.e.*, p_G will converge to p_d.

[2]The Kullback–Leibler divergence, also known as relative entropy, is a measure of how one probability distribution differs from another reference probability distribution. Let $p(x)$ be a given distribution and $q(x)$ the reference distribution, both defined on a domain $\mathcal{X}$. The KL divergence between p and q is defined as:

$$KL(p \parallel q) = \int_{\mathcal{X}} p(x) \cdot \log(\frac{p(x)}{q(x)}) dx$$

The KL divergence is an unbounded positive quantity; it is 0 only when $p = q$. It is possibly infinite when there are points such that $q(x) = 0$ and $p(x) > 0$. Note that it is *not* symmetric.

The Jensen–Shannon divergence is another measure of the distance between two probability distributions. It is bounded and symmetric. Given two probability distributions p and q, the JS divergence can be defined in function of $KL(p \parallel m)$ and $KL(q \parallel m)$ where $m = \frac{1}{2}(p + q)$:

$$JS(p \parallel q) = \frac{1}{2} \cdot KL(p \| m) + \frac{1}{2} \cdot KL(q \| m)$$

The JS divergence is 0 when $p = q$ and maxes-out to $\log(2)$ in the case where p and q have disjoint supports or supports that lie on low dimensional manifolds.

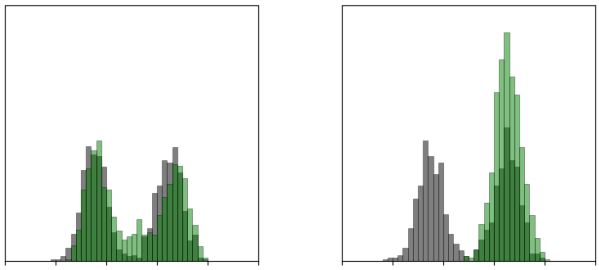
2.1.5 GANs in practice

We have shown, under some specific assumptions, that GANs converge and that the generator can recover exactly the distribution of the data p_d. Doing so, we have made the strong assumption that D and G have infinite capacity. Consequently, we assumed to work directly in the space of probability density functions p_G. However in practice, we only have indirect access to p_G through the parameters θ_G of G. Moreover, it is usually impossible to train D to optimality. Hence, the proofs of global optimality and convergence do not hold in a general and practical parametric setting. In other words, there is no guarantee that GAN actually converge to $p_G = p_d$

Does this imply that GANs are only a theoretical curiosity? Of course not, as their huge popularity seems to indicate. Besides, we have shown in Figure 2.4 that a Vanilla GAN can behave very similarly to the theory on a toy example. This is due in part to the fact that neural networks, even if they do not have infinite capacity, still have an impressive capability of representing a wide range of functions. Nevertheless, neural nets remain hard to train on complex real data, especially in the context of a two player game where there is not one but two adversarial learning objectives. Indeed, while the two players can reach an equilibrium, they can also undo each other's progress, resulting in massive instabilities. Such unstable behaviors can take many forms. When working with vanilla GANs, two big issues related to non-convergence are typically reported: mode collapse and vanishing gradients.

Mode collapse

Most real world data is highly complex and multi-modal. Mode collapse happens when the generator produces samples from only one mode (complete collapse) or few modes (partial collapse) of the data distribution. In theory, nothing prevents such a behavior; in fact, it actually seems quite intuitive. Consider data distributed as a bi-modal 1D Gaussian as an illustration: in the course of training, suppose that G learns that it can fool D when producing convincing images of one of the two mode. It will therefore receive positive feedback and be encouraged to produced more samples of that mode. Eventually, it may be trapped in some kind of positive feedback loop and only produce samples from this mode. Such a collapse is represented on Figure 2.5 (b).



(a) Convergent behavior (b) Mode collapse

Figure 2.5: Convergent behavior vs. mode collapse. The black histogram represents samples from the target data $x_d \sim p_d$. The green histogram represents generated samples $\tilde{x}_d \sim p_G$. In (a), G is close to convergence. In (b), G has collapsed on a single mode.

Because D only receives samples of the same mode from G, it will at some point learn to systematically classify these samples as fake. G may then simply switch mode and this process will repeat it self over and over, thereby preventing the model from properly converging.

Although it can be explained rather intuitively, there is still no clear consensus on what exactly in the GAN framework causes G to collapse. However, this does not preclude finding techniques to mitigate the problem. One of them is batch discrimination. It was introduced by T. Salimans et al. in [9]. The idea is to allow the discriminator to make use of some additional information provided by the combination of all samples in the batch, rather than just seeing one sample at the time. Suppose that G is on the verge of collapsing; when looking at a whole batch of generated samples, D can immediately spot that all the samples are similar and, since a batch of real data would not exhibit this kind of property, it classifies each of them as fake. This will help to avoid the collapse of G. Such a thing would not be as simple if D could only process each sample independently from one another, with no additional information on the batch.

The authors propose a specific form batch discrimination in their paper but it is, as they mention, a quite general concept which can be implemented in many ways. In fact, we will describe in a later chapter a kind of batch discrimination technique known as packing.

Vanishing gradients

Vanishing gradients are a very problematic issue when training neural networks in general because they greatly hinder the learning process. In GANs, we speak of vanishing gradients when the generator receives a very weak or null gradient from the discriminator. This typically occurs early in training when G cannot yet produce convincingly real samples. Such a situation is represented on Figure 2.6. D is very good at discriminating between $\mathbf{x}_d$ and $\tilde{\mathbf{x}}_d$: we have $D(\mathbf{x}_d) = 1$ for almost all $\mathbf{x}_d$ and $D(\tilde{\mathbf{x}}_d) = D(G(\mathbf{z})) = 0$ for almost all $\mathbf{z}$. It follows that the term $\mathbb{E}_{z\sim p_z}[\log(1 - D(G(\mathbf{z})))]$ in $\mathcal{L}_G$ will be close to 0 and so will its gradient.

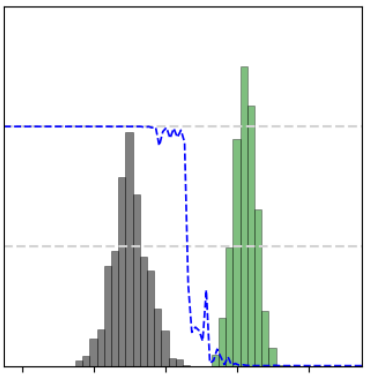


Figure 2.6: Vanishing Gradient. The black histogram represents samples from the target data $x_d \sim p_d$. The green histogram represents generated samples $\tilde{x}_d \sim p_G$. The blue dashed line represents the output of the discriminator $D(x)$.

If the problem is that D is too good, couldn't we just simply train it less ? Unfortunately, it is not that simple. Indeed, we actually want to train D close to optimality: it makes the training criterion of G a better approximation of $JS(p_G \| p_d)$, which we would like to minimize to obtain $p_G \approx p_d$. However, M. Arjosky and L. Bottou have shown in [10] that, if the distributions p_G and p_d have supports that are disjoint or lie on low dimensional manifolds, the optimal D of a vanilla GAN will be perfect[3] and its gradient will be zero everywhere in the union of $supp(p_G)$ and $supp(p_d)$. As the authors provide strong evidence that $supp(p_G)$ and $supp(p_d)$ are indeed concentrated on low dimensional manifolds, we are in a tricky situation : if we train D close to optimality, we will have a meaningful loss criterion for G but we will face vanishing gradients, and if we train D less, we will have stronger gradients but inaccurate updates of G. Both cases are undesirable and constitute obstacles to the convergence of the learning process.

Another solution, which was already mentioned by I.J. Goodfellow et al. in the original GAN paper, consists in modifying $\mathcal{L}_G$ such that G minimizes $\mathbb{E}_{z \sim p_z}[-\log D(G(\mathbf{z}))]$ instead of $\mathbb{E}_{z \sim p_z}[\log(1 - D(G(\mathbf{z})))]$. This new loss matches the same objective of G, *i.e.*, to minimize the probability that its samples are classified as fake by D, while providing stronger gradient to G early in learning and so even in case of a perfect discriminator. This is illustrated on figure 2.7. However, Vanilla GANs using this variation of $\mathcal{L}_G$ still experience important instabilities in practice. M. Arjosky and L. Bottou also show in [10] that, given an optimal D, optimizing this criterion is equivalent to minimizing $KL(p_G \| p_d) - 2 \cdot JS(p_G \| p_d)$. They hypothesize that the instabilities in the updates of G are in part due to the opposite sign of the divergences in this criterion. Indeed it seems to provide contradictory informations : minimizing the positive KL divergence term pushes the distributions towards each other while minimizing the negative JS divergence term pushes them apart.

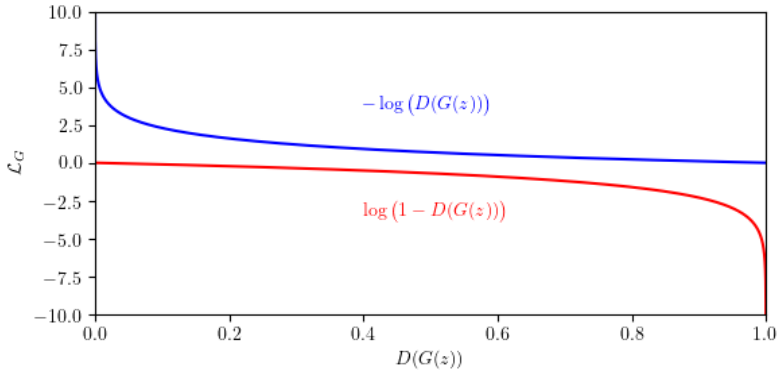


Figure 2.7: Variation of $\mathcal{L}_G$ for the vanilla GAN

None of the presented solutions are satisfactory enough. As the problem of vanishing gradients seems closely related to the loss functions of G and D, and in particular to the KL and JS divergences, M. Arjosky and L. Bottou suggest to use a different loss function which is not based on former divergences but rather on the Wassertein divergence. In a follow up paper of [10], they apply their own recommendations and present a new GAN framework : the Wassertein GAN [6].

[3] $D(\mathbf{x}) = 1$, $\forall \mathbf{x} \in supp(p_d)$ and $D(\mathbf{x}) = 0$, $\forall \mathbf{x} \in supp(p_G)$

2.2 Wassertein GAN

The Wassertein GAN (WGAN) was introduced in [6] by M. Arjosky et al. as a new framework that alleviates the issues of non-convergence suffered by the original vanilla GAN. In particular, it brings a solution to the vanishing gradient problem. Indeed, recall that vanilla GANs optimize the JS divergence between p_G and p_d and that, although it is sound from a theoretical point of view, it has some important shortcomings in practice. Notably, $JS(p_G \| p_d)$ is not well behaved (it maxes-out) when the supports of p_G and p_d are disjoint or lie on a low dimensional manifold, which is most often the case for real world data. This leads to zero gradient almost everywhere. If we want to avoid this issue, we must not train D until optimality but then we only minimize a mere approximation of $JS(p_G \| p_d)$, which is also undesirable. Moreover, we have seen that modifying the loss function of G to $\mathbb{E}_{z \sim p_z}[-\log D(G(\mathbf{z}))]$ is not a viable solution either.

Because of these shortcomings, mainly related to the JS divergence, M. Arjosky et al. propose to base instead the objective criterion of GANs on the Wassertein divergence. As the JS divergence, the Wassertein divergence is a measure of the distance between two distributions and is arguably better behaved than the former in the context of GANs. The WGAN framework is presented in Figure 2.8.

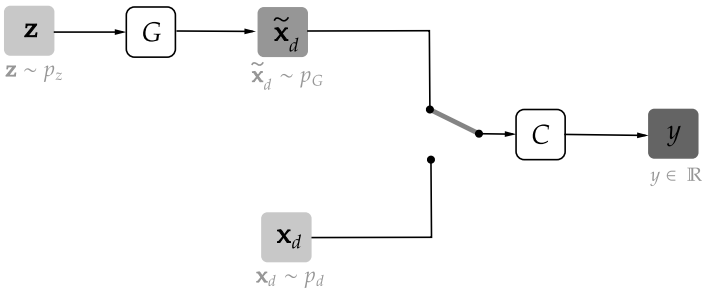


Figure 2.8: WGAN framework

In this framework, the discriminator still yields an appreciation of how real a sample $\mathbf{x}$ is, but this appreciation does not have to be a probability anymore. Hence, D does not play a proper discriminating role and the authors of the WGAN paper refer to it as the "critic" instead of the "discriminator". However, for means of simplicity and to remain consistent with previously established notations, we will keep referring to it as the Discriminator D. In theory, for an optimal D, the generator G minimizes $W(p_G, p_d)$. The WGAN framework has better stability and better convergence properties than the original vanilla GAN framework. Notably, using the Wassertein divergence as a training criterion alleviates the vanishing gradient problem.

As we will make use of such a WGAN framework to build our tabular data synthesizer, we want to get some more intuition on how the Wassertein divergence works, why it is well behaved in the context of GANs, and why it solves the vanishing gradient problem. We will thus develop this in the next few pages. Also, we will show how it is implemented in practice.

2.2.1 Wassertein divergence

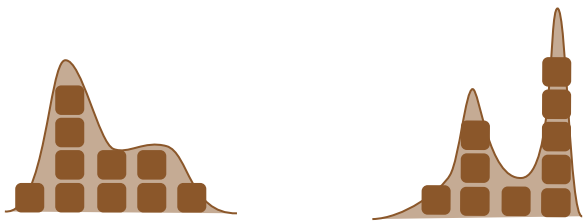
The wassertein divergence between two probability distributions P_x and P_y is expressed as follow :

$$W(P_x, P_y) = \inf_{\gamma \in \Pi(P_x, P_y)} \mathbb{E}_{(x,y)\sim\gamma}\big[\|x - y\|\big] \tag{2.18}$$

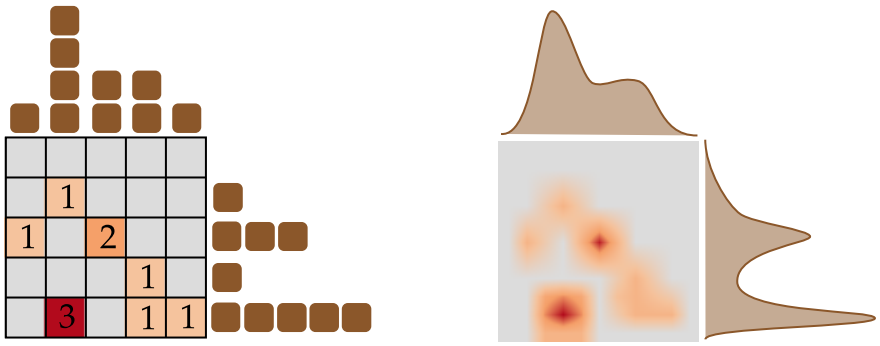
Where $\Pi(P_x, P_y)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively P_x and P_y. Since a distribution is defined by how much "mass" is attributed to each point, $\gamma(x, y)$ can be understood as how much "mass" needs to be transported from x to y in order to transform P_x into P_y. The Wassertein divergence is in fact strongly related to optimal transport and is famously know as the Earth-Mover distance. Lets therefore use this "earth-moving" analogy to explain the intuition behind equation 2.18.

Suppose we are a team of construction workers and our boss asks us to turn a pile of earth X into a pile of earth Y. At first, he just wants us to change the shape of the pile, not rebuild it in a different place. We have a shovel at our disposal, and as we are the type of persons who like to do things efficiently, we want to minimize the work w we will have to do in order to turn pile X into pile Y. This work, or effort, can be defined as the mass m (in metric tons) of earth to be moved times the distance d (in meters) on which it has to be moved : $w = m \cdot d$. This is an optimization problem so lets take a piece of paper, a pen, a warm cup of coffee and start to think about it.

To make our life easier, we discretize the pile into blocks of 1 cubic meter of earth, and assume a density of 1 such that each block weighs a metric ton. We then organize these blocks in columns such that the initial pile X can be seen as a 1D object described by P_x. In particular, $P_x(x)$ gives the mass in tons, or the number of blocks, in the column at position x. Similarly, the transformed pile Y can be represented by P_y where $P_y(y)$ indicates the number of blocks in the columns at position y.



Transforming P_x into P_y can thus be reduced to moving blocks of earth with the shovel from certain columns to others. Therefore, on our piece of paper, we try to device a plan $\gamma(x, y)$ that tells us how many blocks we want to take from the column x in P_x and move to column y in P_y. Note that there is 5 columns in P_x while only 4 in P_y, we will thus consider that the first column of P_y contains no block. An example of such a plan γ is represented in Figure 2.9 (a) in the form of a 5×5 grid. The content of the cell at position (x, y), starting from the upper left corner, gives the number of blocks to move from x to y. For example, the digit 3 at position $(2, 5)$ means that we must move 3 blocks from the second column to the fifth column. The numbers on the diagonal represent the blocks that do not need to be moved.



(a) (b)

Figure 2.9: Intuition for the Wassertein divergence. (a) Discrete representation of $\gamma(x, y)$. (b) Continuous representation of $\gamma(x, y)$

Observe that the sum of $\gamma(x, y)$ over a column x is always equal to the number of blocks in the corresponding column x of P_x and that the sum of $\gamma(x, y)$ over a row y is always equal to the number of blocks in the corresponding column y of P_y. We have:

$$P_x(x) = \sum_y \gamma(x, y) \tag{2.19}$$

$$P_y(y) = \sum_x \gamma(x, y) \tag{2.20}$$

Next, recall the relation $w = m \cdot d$: the effort required for moving some blocks of column x to a given column y is the mass of the blocks to move, which is given by $\gamma(x, y)$, times the distance over which they are displaced, which we will naturally take as the euclidean distance $\|x - y\|$. The effort of moving all the blocks from one column x to P_y is then : $\sum_y \gamma(x, y) \cdot \|x - y\|$. Note that if one block remains in the column, the corresponding term in the sum will be null since $\|x - y\| = 0$. If we take this last result and sum over all the columns x, we obtain the effort for transforming P_x into P_y:

$$w(P_x, P_y) = \sum_x \sum_y \gamma(x, y) \cdot \|x - y\| \tag{2.21}$$

Finally, we have to find the plan γ that minimizes this expression of the total effort. The search space for γ is limited. Indeed, for the plan to make any sense, it has to satisfy the conditions expressed in equations 2.19 and 2.20. This simply means that every block leaving a column of P_x must land somewhere in P_y and every block arriving in a column of P_y must come from somewhere in P_x.

Now that we have find a way to proceed for discretized piles of earth, adapting the reasoning to the continuous case (see Figure 2.9) is straight forward: the effort required to move all the earth that comes from a slice of P_x, of width $\mathrm{d}x$ and located at position x, to some new location in P_y is given by $\int_y \gamma(x, y) \cdot \|x - y\| \mathrm{d}y$, and the total effort of repeating this operation for all slices of P_x is computed as:

$$w(P_x, P_y) = \int_x \int_y \gamma(x, y) \cdot \|x - y\| \, \mathrm{d}y \, \mathrm{d}x \tag{2.22}$$

The conditions in equations 2.19 and 2.20 become :

$$P_x(x) = \int_y \gamma(x, y)\,\mathrm{d}y \tag{2.23}$$

$$P_y(y) = \int_x \gamma(x, y)\,\mathrm{d}x \tag{2.24}$$

If we interpret P_x, P_y and γ as distributions, we see that our plan makes sense only if P_x and P_y are the marginal distributions of the joint distribution γ. Our objective is therefore to find the plan γ that minimizes equation 2.22 among the space of all joint distributions that have P_x and P_y as marginals. The minimal effort $W(P_x, P_y)$ of transforming P_x into P_y is obtained as $W(P_x, P_y) = \inf w(P_x, P_y)$:

$$W(P_x, P_y) = \inf_{\gamma \in \Pi(P_x, P_y)} \int_x \int_y \gamma(x, y) \cdot \|x - y\|\,\mathrm{d}y\,\mathrm{d}x \tag{2.25}$$

Given the mathematical definition of expectation, equation 2.25 is totally equivalent to equation 2.18. The Wassertein divergence, or Earth-mover distance, can thus be understood as the minimal effort required to move one pile of earth into another one. In our analogy, we have considered a case where P_x and P_y are uni-dimensional and have a common support. Of course, it can be generalized to more complex multi-dimensional distributions, and most importantly, to distributions with supports that are disjoint or lie on low dimensional manifolds. Indeed, imagine a case where our hypothetical boss asks us to transform the pile X into the same pile Y as before, but located 150m away. In terms of distributions, this would mean that P_x and P_y have disjoint supports. However, this would not cause $W(P_x, P_y)$ to max out as the JS divergence would, instead, the term $\|x - y\|$ in the integrand of equation 2.25 will simply be increased by 150.

The ability of the Wassertein divergence to capture, through its term $\|x-y\|$, the distance between distributions that have disjoint supports is what makes it such a good candidate for an optimization criterion in GANs.

2.2.2 WGAN loss

Because of its nice properties, we would like for G to minimize the Wassertein divergence between p_G and p_d :

$$W(p_d, p_G) = \inf_{\gamma \in \Pi(p_d, p_G)} \mathbb{E}_{(\mathbf{x}_d, \tilde{\mathbf{x}}_d) \sim \gamma} \big[\|\mathbf{x}_d - \tilde{\mathbf{x}}_d\| \big] \tag{2.26}$$

This is a priori not easy as the infinimum over $\gamma \in \Pi(p_d, p_G)$ is highly intractable. Fortunately, as a result of the Kantorovich-Rubinstein duality [11], we can express the Wassertein divergence as:

$$W(p_d, p_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}_d \sim p_d} \big[f(\mathbf{x}_d) \big] - \mathbb{E}_{\tilde{\mathbf{x}}_d \sim p_G} \big[f(\tilde{\mathbf{x}}_d) \big] \tag{2.27}$$

Where the supremum is taken over all 1-Lipschitz function $f : \mathcal{X} \to \mathbb{R}$. We can then approximate f with D such that the minimax game between G and D becomes :

$$\min_G \max_{\|D\|_L \leq 1} \quad \mathbb{E}_{\mathbf{x}_d \sim p_d} \big[D(\mathbf{x}_d) \big] - \mathbb{E}_{\mathbf{z} \sim p_z} \big[D(G(\mathbf{z})) \big] \tag{2.28}$$

In a practical training situation, we use the discrete expectations. Given a batch of m noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ and a batch of m data samples $\{\mathbf{x}_d{}^{(1)}, ..., \mathbf{x}_d{}^{(m)}\}$, we can thus express $\mathcal{L}_D$ and $\mathcal{L}_G$ as:

$$\mathcal{L}_D(\theta_G, \theta_D) = -\frac{1}{m}\sum_{i=1}^{m}\Big(D(\mathbf{x}_d{}^{(i)}; \theta_D) - D(G(\mathbf{z}^{(i)}; \theta_G); \theta_D)\Big) \tag{2.29}$$

$$\mathcal{L}_G(\theta_G, \theta_D) = \frac{1}{m}\sum_{i=1}^{m}\Big(-D(G(\mathbf{z}^{(i)}; \theta_G); \theta_D)\Big) \tag{2.30}$$

For an optimal D, equation 2.28 tells us that G minimizes $W(p_d, p_G)$. Because $W(p_d, p_G)$ is well behaved (it does not max out) even when the supports of p_G and p_d lie on low dimensional manifolds, we can actually train D until or near optimality without it leading to vanishing gradients. This is the big advantage of WGAN over vanilla GAN. In fact, in theory, we actually want to train D to optimality: a better D means a better estimates of $W(p_d, p_G)$ and more reliable gradients back-propagated to G. In practice however, it may be to computationally heavy to learn an optimal D so we do not necessarily do so.

With vanishing gradients out of the way, the learning process is much more stable than in vanilla GANs and G has better convergence properties. In Figure 2.10, we see how the gradient vanishing problem is solved on a toy example. Indeed, we observe that the output of the WGAN discriminator has well behaved gradients for all $\mathbf{x}$. In contrast, the discriminator of the vanilla GAN leads to zero gradients for all $\mathbf{x} \in supp(p_G) \cup supp(p_d)$.

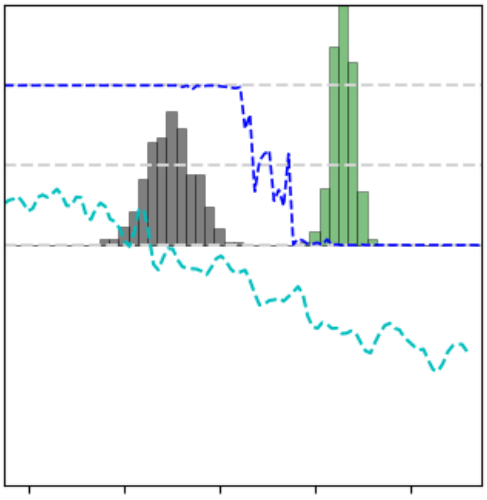


Figure 2.10: Vanishing gradient solved. The black histogram represents samples from the target data $x_d \sim p_d$. The green histogram represents generated samples $\tilde{x}_d \sim p_G$. The blue dashed line represents $D(x)$ for the vanilla GAN. The cyan dashed line represents $D(x)$ for the WGAN.

One important thing that we have not considered yet is the constraint on D : it has to be 1-Lipschitz otherwise none of all this works. In particular, G will not optimize $W(p_d, p_G)$. In our case, for D to be 1-Lipschitz means that the norm of its gradient must always be at most 1 : $\|\nabla_{\mathbf{x}} D(\mathbf{x})\| \leq 1, \forall \mathbf{x}$.

In the WGAN paper [6], the authors use weight clipping to enforce the Lipschitz constraint on the discriminator. After each gradient upgrade of D, the weights θ_D are clipped to an interval $[-c, c]$ where c is the clipping parameter. However they state themselves that "weight clipping is clearly a terrible way to enforce a Lipschitz constraint".

Indeed, I. Gulrajani et al. show in [12] that weight clipping pushes all the weights to extreme values of the interval $[-c, c]$ and that a poorly chosen c can then lead to vanishing or exploding gradients. As an alternative to enforce the Lipschitz contraint, they propose to add a penalty on the norm of the gradient in the loss of D :

$$\mathcal{L}_D = -\big(\mathbb{E}_{\mathbf{x}_d \sim p_d}\big[D(\mathbf{x}_d)\big] - \mathbb{E}_{\tilde{\mathbf{x}}_d \sim p_G}\big[D(\tilde{\mathbf{x}}_d)\big]\big) + \lambda \cdot \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{x}}}\big[\big(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\| - 1\big)^2\big] \tag{2.31}$$

Note that this penalty actually pushes $\|\nabla_{\mathbf{x}} D(\mathbf{x})\|$ to values close to 1. This is motivated by the fact that an optimal D in a WGAN framework has a gradient of norm 1 almost everywhere in $supp(p_d) \cup supp(p_G)$. Besides, because the unit gradient norm constraint cannot be enforced everywhere for computational reasons, it is enforced only on certain $\hat{\mathbf{x}} \sim p_{\hat{x}}$ that are random linear combinations of $\mathbf{x}_d$ and $\tilde{\mathbf{x}}_d$:

$$\hat{\mathbf{x}} = \epsilon \cdot \mathbf{x}_d + (1 - \epsilon) \cdot \tilde{\mathbf{x}}_d \qquad \epsilon \sim \mathcal{U}(0, 1) \tag{2.32}$$

Other methods for enforcing the Lipschitz constraint have been proposed in the GAN literature, notably spectral normalization [13]. However, as we did not use such methods to regularize the WGAN framework in our tabular data synthesizer, they are outside the scope of this thesis. In particular, we will enforce the constraint through gradient penalty.

Also, note that while the WGAN loss was initially designed for solving the vanishing gradient problem, it proved to have other advantages. Among other things, the authors of [6] have empirically shown that it is better correlated with the quality of the generated samples and that it drastically reduces mode collapse.

Chapter 3

Tabular data synthesis with GANs

3.1 Task formalization

To generate synthetic tabular data, we need a synthesizer $\mathcal{S}$ which is learned from existing tabular data. In our case, the main component of this synthesizer is the generator network of a GAN. $\mathcal{S}$ takes as input a table $\mathbf{T}$ and outputs a synthetic table $\tilde{\mathbf{T}}$ with similar properties. In the scope of this work, we limit the possible inputs to single tables with independent rows and no missing values.

$\mathbf{T}$ has n rows, k numerical columns and l categorical columns. In total, it has $k + l = m$ columns[1]. Each column corresponds to a random variable $X_{d,j}$, $j \in \{1, ..., m\}$. Together, all random variables follow a joint probability distribution p_d. This is what we referred to in the previous chapter as the distribution of the original data. Each row $\mathbf{x}_d^{(i)}$, $i \in \{1, ..., n\}$, corresponds to one sample $\mathbf{x}_d$ of this unknown distribution : $\mathbf{x}_d \sim p_d$. We have $\mathbf{x}_d^{(i)} = \{x_{d,1}^{(i)}, ..., x_{d,m}^{(i)}\}$.

In the course of training, $\mathcal{S}$ builds (implicitly) a model p_G. The learning objective is to achieve $p_G = p_d$. When training is completed, $\tilde{\mathbf{T}}$ is obtained by sampling rows from the constructed model: $\tilde{\mathbf{x}}_d \sim p_G$. The quality of the synthetic table $\tilde{\mathbf{T}}$ can then be assessed by some evaluation function $f(\mathbf{T}, \tilde{\mathbf{T}})$.

When synthesizing data in the context of data augmentation, $\mathbf{T}$ is separated (row-wise) into $\mathbf{T}_{train}$ and $\mathbf{T}_{test}$ and $\mathcal{S}$ is trained on $\mathbf{T}_{train}$ only. The augmented table is obtained by concatenating the synthetic rows to the training table : $\mathbf{T}_{aug} = \mathbf{T}_{train} \oplus \tilde{\mathbf{T}}$. Depending on the type of problem that must be solved, a specific model can then be trained on $\mathbf{T}_{aug}$ and evaluated on $\mathbf{T}_{test}$.

[1]Throughout this manuscript, we have also used the letter m to denote the batch size. Normally, the context is sufficient to determine which of the two quantities is involved.

3.2 Challenges

As we have seen in Chapter 1, tabular data has unique properties that make it difficult to synthesize. Moreover, in Chapter 2, we discussed the obstacles that GANs have to overcome in practice to properly learn the distribution of some given data. It is therefore not surprising that the synthesis of tabular data with GANs presents many challenges. Such challenges have been identified by L. Xu et al. in [1] and need to be kept in mind when designing an efficient synthesizer.

Mixed data types : this is probably the most challenging property of tabular data when it comes to data synthesis. Notably, the generator of a GAN can only produce continuous outputs and thus GANs cannot handle categorical attributes unless they are encoded in some way. Most often, categorical attributes are simply one-hot-encoded, which is only a half-solution. Indeed, G can learn, through a softmax activation function, to generate a probability distribution over the categories of a given attribute, but it is hard for G to output vectors filled with exact 0's and an exact 1 for the represented category. The problem is that if G cannot generate proper one-hot vectors, D can simply tell the difference between fake generated data and real data. Therefore, in addition to being capable of treating categorical and numerical data simultaneously, a tabular data synthesizer will have to make use of a more refined categorical encoding, or an activation function that can produce better approximations of one-hot vectors.

Non-Gaussian distributions : as with most machine learning algorithms that treat multiple continuous attributes which can have different scales and units, we want to normalize and/or standardize our data before feeding it into the GAN. The ranges we want for the continuous attributes depend on their corresponding activation functions in the last layer of G: for example, $[0, 1]$ for sigmoid activations, $[-1, 1]$ for tanh activations, etc. When the values of the attributes do not follow a Gaussian-like distribution, as it is often the case in complex real world tabular data, normalization and standardization is not straight forward. For example, applying minmax normalization on a highly skewed distribution may result in most values very close to the limits of the minmax interval where the gradient of tanh and sigmoid functions are very weak, yet G cannot learn efficiently with weak gradients.

Multi-modal distribution : in addition to being not necessarily Gaussian-like, tabular data can also be highly multi-modal. Since GANs suffer from mode collapse, capturing all modes is naturally a big challenge.

Highly imbalanced categorical columns : categorical columns, or attributes, may be strongly imbalanced, *i.e.* most rows belong to a major category while only a few belong to one or more minor categories. In this case, the minor categories are often those that contain the most relevant information. Unfortunately, because they are poorly represented, they lack training opportunities, which may lead to mode drop-outs that are likely to go unnoticed by the discriminator. An efficient tabular data synthesizer must thus be able to deal with such imbalanced attributes, for examples by giving more weight to minor categories. However, the difficulty is that G must still recover the original distribution of the data, not a distribution where minor categories are over-represented.

3.3 Existing approaches

Although still limited in comparison to the literature on GAN-based image generation, the literature on GANs for tabular data synthesis is starting to grow and several approaches corresponding to our task (as formalized in section 3.1) have been proposed over the past couple of years.

MedGAN has been proposed in [14] by E. Choi et al. to generate electronic health records (EHRs). As with most of the approaches we will present, MedGAN's main objective is to create realistic synthetic data while preserving the confidentiality of the original data, so it is not necessarily focused on data augmentation. Nevertheless, MedGAN (and other approaches) can be applied in this context. It deals with categorical attributes by training a vanilla auto-encoder on the original data. The trained encoder allows categorical variables to be mapped to a continuous latent space, while the decoder allows this mapping to be reversed, *i.e.*, the original representation to be recovered from the continuous representation. The generator of MedGAN can thus produce samples directly in the continuous latent space defined by the the auto-encoder and these samples are then decoded before being passed to the discriminator, alongside samples from the original data. In addition, the authors propose a method they call *minibatch averaging* to cope with mode collapse, and a method combining batch normalization, ReLU activations and skip connections to enhance the training of G. All G, D, the encoder and the decoder are implemented using MLPs.

The design of MedGAN has inspired many other approaches for tabular data synthesis. CorrGAN [15] uses the same architecture but has an additional term in the reconstruction loss of the auto-encoder to encourage the decoder to preserve the correlation between the different attributes. A. Torfi and E. Fox propose instead in [16] to use convolutional layers both in the GAN and the auto-encoder to better capture this correlation. R. Camino et al. propose in [17] to improve the MedGAN architecture by using the WGAN framework with gradient penalty (WGAN-GP) [12]. In addition, they suggest to use a separate dense layer for each categorical variable at the end of the decoder, then apply a gumbel-softmax[2] activation [18] on the output of each of these layers. This means that the decoder can produce convincing one-hot-encoded vectors, which in turn makes the discriminator's task more difficult.

CrGAN-Cnet [19] takes a different approach to model airline passenger name records. It integrates cross networks [20] in G and D to better capture the interactions between the attributes and uses the Cramer distance as an optimization criterion for the GAN [21]. There is no auto-encoder, G uses standard softmax activations for categorical attributes. However, to avoid easy discrimination between generated samples and proper one-hot-encoded real samples, an embedding layers is added before the first layer of D. CrGAN-Cnet is also designed to handle missing values.

[2]The gumbel-softmax activation can produce vectors $\{0,1\}^d$ for an attribute with d categories, while still being differentiable. We will develop this further in Chapter 4.

Table-GAN [22] uses the architecture of DCGAN [23], the famous deep convolutional GAN that revolutionized image generation. To this end, it transforms rows of tabular data into square matrices and processes them as images. Moreover, it introduces an auxiliary classifier that is trained on the original data to predict the classes of the real samples. This classifier is then used to predict the class of the generated samples and feedback is provided to G through the *classification loss*. This encourages G to produce samples that are semantically coherent. Table-GAN adds another term to the loss function of G : the *information loss*. This is an adaptation of what the authors of [9] call *feature matching* and consists in enforcing the generator to produce samples that match some specific statistic of the real data, *e.g.*, the expected value of the features of some intermediate layer of D.

TGAN [24] has yet another architecture: it uses Long Short Term Memory (LSTM) cells in G to synthesize each column one by one. LSTMs are very popular for text generation; the idea to use them for tabular data is to better capture the relation between attributes, as they would successfully do with words. G has softmax activations for the categorical attributes so noise is added to these one-hot-encoded attributes such that the task of D is non trivial. Also, TGAN introduces *mode-specific normalization* for numerical attributes which addresses the challenge of normalizing non Gaussian-like distributions. It consist in fitting a Gaussian mixture on a numerical attribute and then normalizing the values of the attribute with respect to the corresponding modes of the mixture. Finally, TGAN adds a term in the vanilla loss function of G that directly penalizes the sum of KL divergences between each synthesized column and its real counterpart.

So far, non of the presented tabular data synthesizers explicitly tackles the issue of highly imbalanced categorical columns. L. Xu et al. take up the challenge and in a follow up paper of [24], they present a novel approach called CTGAN [1]. In particular, they switch from a vanilla GAN framework to a conditional WGAN-GP framework. WGAN-GP brings more stability and better convergence properties while the new conditional nature means that G can be encouraged to produce samples from a particular category. Therefore, the authors propose a training method which, by encouraging the generator to produce samples from the minor categories, alleviates the issue of imbalanced categorical attributes. They call this method *training by sampling*. Moreover, they drop the LSTM cells in G in favor of an easier to train MLP and do not add noise to one-hot-encoded categorical attributes but instead use gumbel- softmax activations. Also, they keep mode specific normalization and integrate a PACGAN [25] structure in D which uses packing to prevent mode collapse. The authors show that CTGAN outperforms other methods such as MedGAN and Table-GAN.

Because of the impressive results of CTGAN, other state-of-the-art methods for tabular data synthesis have adopted the conditional WGAN-GP framework. One of them is CWGAN [26]. It was designed specifically for oversampling the minority class in the context of imbalanced learning and uses a similar structure to [19] with cross networks in G and D, and an input embedding layer in D for categorical attributes. Moreover, CWGAN has the particularity of conditioning the numerical outputs of G on its own categorical outputs. The idea is to allow G to model more easily relationships between numerical attributes and particular realisations of categorical attributes. The authors call this technique *self conditioning*. They also use an auxiliary classifier as in [22].

According to our knowledge, the most recent approach that uses the conditional WGAN-GP framework is CTAB-GAN [27]. Its core operating principles are very similar to those of the CTGAN. In particular, it leverages *mode-specific normalization* with Gaussian mixtures and uses the same *training by sampling* procedure. Nonetheless, the authors make some key contributions. First, G and D are 1d convolutional networks instead of standards MLPs. Second, they add two terms to the loss of G as in [22]: the *information loss* and the *classification loss* (CTAB-GAN thus has an auxiliary classifier). Third, they design an encoder that can handle mixed type attributes, *i.e.*, attributes that have both a numerical component and some specific discrete values. Finally, they propose to use a simple yet effective log transform on skewed numerical attributes. They compare CTAB-GAN to MedGAN, Table-GAN, CTGAN and CWGAN, and show that it outperforms all of them (sometimes by a wide margin) both in the quality and the usability of the generated data.

3.4 Our approach

Among all existing approaches, CTAB-GAN seems to be the most performant and appropriate for our task. We would thus like to use it as a foundation for our own tabular data synthesizer. However, at the time of writing this thesis, the paper presenting CTAB-GAN is still under review and the authors have not provided many details regarding the implementation, which makes it difficult to replicate the model in practice. On the other hand, the authors of [1] have developed a python library[3] with an open-source implementation of CTGAN. Since CTAB-GAN is largely based on CTGAN, we will stick to the main architecture of CTGAN as a basis for our tabular data synthesizer and then try to enhanced it with some features of state-of-the-art models and also with a couple of original contributions. In particular, we will

- use the conditional WGAN-GP framework and implement G and D as MLPs. We use a PACGAN structure for D in order to prevent mode collapse.
- apply *mode specific normalization* to numerical attributes and design a more flexible Gaussian mixture model to imitate the behavior of the mixed type encoder of CTAB-GAN. We will also apply a log transform to highly skewed numerical attributes.
- design a novel and semantically meaningful encoding-decoding scheme for categorical attributes such that they can be represented in a continuous latent space and thus be modeled more easily by G. The produced embedding will also allow to better capture the relationships between all types of attributes, categorical and/or numerical.
- add *feature matching* terms to the loss function of G; one of the terms corresponds to the *information loss*.
- train an auxiliary network (classifier or regressor) to learn the target attribute of the table and encourage G to produce coherent sample rows.
- follow the *training by sampling* procedure to deal with imbalanced categorical attributes.

In chapter 4, we develop in detail the architecture of the CTGAN synthesizer and our enhanced version of it.

[3] https://sdv.dev/CTGAN

Chapter 4

GAN-based models

4.1 Baseline CTGAN

The global architecture of the tabular data synthesizer CTGAN [1] is represented on Figure 4.1. During the training phase, a data sampler S samples a batch of m conditional vectors $\mathbf{c}$ and a batch of m rows $\mathbf{x}_d \sim p_d(\mathbf{x}_d|\mathbf{c})$ from the original data such that the i^{th} row $\mathbf{x}_d^{(i)}$ matches the condition given by $\mathbf{c}^{(i)}$. Note that we do not have to satisfy $p_d(\mathbf{x}_d|\mathbf{c}) = p_d(\mathbf{x}_d)$ and therefore, in the context of an imbalanced classification problem, we can sample $\mathbf{c}$ such that minority classes are favoured. The batch $\mathbf{x}_d$ is passed to a transformer T which applies a series of reversible transformations to the data such that it becomes usable : $T(\mathbf{x}_d) = \mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{c})$. The batch of conditional vectors $\mathbf{c}$ is passed as input to the generator together with a batch of m noise vectors $\mathbf{z} \sim p_z$, where p_z is a multi-dimensional Gaussian distribution. G outputs a batch of m synthetic rows $\tilde{\mathbf{x}}_t$. The goal for G is to achieve $p_G(\tilde{\mathbf{x}}_t|\mathbf{c}) = p_t(\mathbf{x}_t|\mathbf{c})$ and thus that $\tilde{\mathbf{x}}_t^{(i)}$ matches $\mathbf{c}^{(i)}$. $\mathbf{x}_t$ and $\tilde{\mathbf{x}}_t$ are then passed to D which gives an appreciation of how real the samples are, given $\mathbf{c}$. G and D are optimized using the WGAN-GP loss.

When the model is trained, we can sample an arbitrary batch $\mathbf{c}$ depending on the type of rows we want to synthesize and apply the inverse of T to the output $\tilde{\mathbf{x}}_t$ of G in order to recover $\tilde{\mathbf{x}}_d$: $T^{-1}(\tilde{\mathbf{x}}_t) = \tilde{\mathbf{x}}_d$. If $\mathbf{c}$ is sampled such that $\mathbf{x}_d \sim p_d(\mathbf{x}_d)$ and if the model has properly converged, we should have $\tilde{\mathbf{x}}_d \sim p_d$.

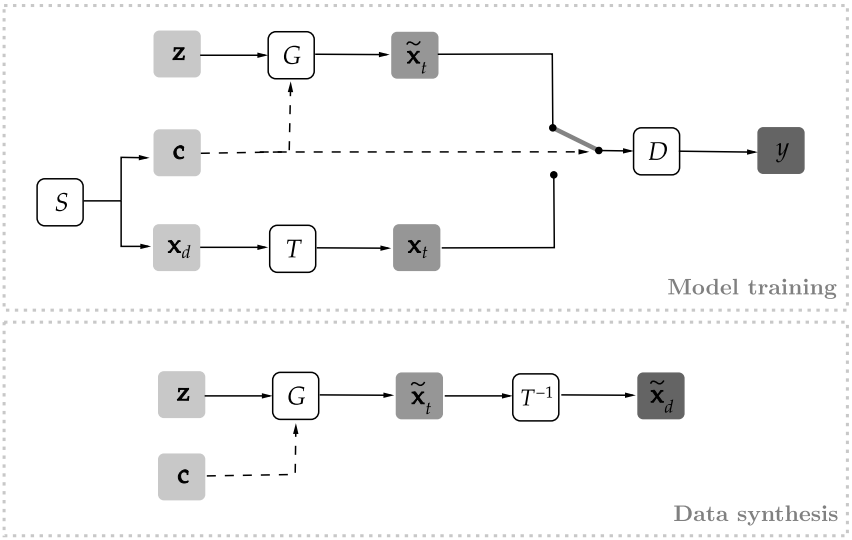


Figure 4.1: Global architecture of CTGAN

4.1.1 Data transformation

The goal of the transformer T is to make data easier to process by G and D. Indeed, given the diversity of tabular data, a given table is usually not directly exploitable. Typically, we want to represent the categorical columns in some way that allows computation and normalize the numerical columns such that they all have similar ranges (often $[0, 1]$ or $[-1, 1]$).

Suppose a table $\mathbf{T}$ with m columns : k numerical columns and l categorical columns. Recall that each column corresponds to a random variable X_j and lets assume that indices from 1 to k refer to numerical columns and indices from $k+1$ to m refer to categorical columns. The transformation process for such a table $\mathbf{T}$ is represented on Figure 4.2.

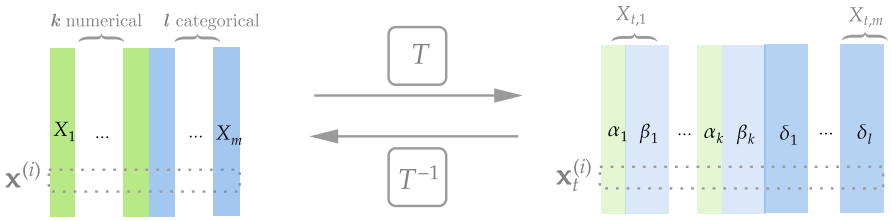


Figure 4.2: Data transformation in CTGAN

Categorical columns

In CTGAN, the values of each categorical column X_j are represented as one-hot vectors $\boldsymbol{\delta}_{j-k}$ of dimension equal to $|X_j|$, the number of possible values for the considered column. We have $\boldsymbol{\delta}_1$ corresponding to X_{k+1}, $\boldsymbol{\delta}_2$ to X_{k+2}, etc. $\boldsymbol{\delta}_l$ corresponds to X_m.

Numerical columns

Due to the difficulty of normalizing complex columns with multi-modal non Gaussian-like distributions, CTGAN introduces *mode-specific normalization*. As its name suggests, the technique consists in normalizing the values in a numerical columns with respect to specific modes of the column's distribution. A given value is thus represented by a combination of a scalar normalized value α and a one-hot vector $\boldsymbol{\beta}$ indicating the mode with respect to which it has been normalized.

To identify the modes in the distribution of a numerical column j, CTGAN fits a variational Gaussian mixture model (VGM) to X_j. The learned mixture is given by:

$$\mathbb{P} = \sum_q \omega_q \cdot \mathcal{N}(\mu_q, \sigma_q) \tag{4.1}$$

where ω_q, μ_q and σ_q are respectively the weight, the mean and the standard deviation of the Gaussian that fits mode q. Next, for each value $x_j^{(i)}$ in X_j, a mode q^* is sampled among all modes of the mixture. For a given $x_j^{(i)}$, each mode q is sampled with a probability proportional to $\rho_q = \omega_q \mathcal{N}(x_j^{(i)}; \mu_q, \sigma_q)$.

Finally, for each $x_j^{(i)}$, $\alpha_j^{(i)}$ and $\boldsymbol{\beta}_j^{(i)}$ are computed as :

$$\alpha_j^{(i)} = \frac{x_j^{(i)} - \mu_{q^*}}{4\sigma_{q^*}} \tag{4.2}$$

$$\boldsymbol{\beta}_j^{(i)} = \mathbf{e}_{q^*} \tag{4.3}$$

where $\mathbf{e}_{q^*}$ is a vector filled with zeros and a one at position q^*. Note that $x_j^{(i)}$ is scaled by 4σ. This is to have 99.99% of the normalized values between -1 and 1. To make sure that 100% of the values effectively fall into that range, CTGAN clips all α's to $[-1, 1]$.

Figure 4.3 illustrates the principles of *mode-specific normalization* on a simulated column X_j. The VGM model identifies 3 modes (a) and the learned mixture is thus:

$$\mathbb{P} = \sum_{q=1}^{3} \omega_q \cdot \mathcal{N}(\mu_q, \sigma_q) \tag{4.4}$$

For a given $x_j^{(i)}$, a mode q^* is sampled among mode 1, mode 2 and mode 3 (b). Since we have that $\rho_3 >> \rho_2 > \rho_1$, mode 3 is much more likely to be picked than the two others. This is the case here and so $\alpha_j^{(i)}$ and $\boldsymbol{\beta}_j^{(i)}$ are computed as :

$$\alpha_j^{(i)} = \frac{x_j^{(i)} - \mu_3}{4\sigma_3} \tag{4.5}$$

$$\boldsymbol{\beta}_j^{(i)} = \mathbf{e}_3 = [0, 0, 1] \tag{4.6}$$

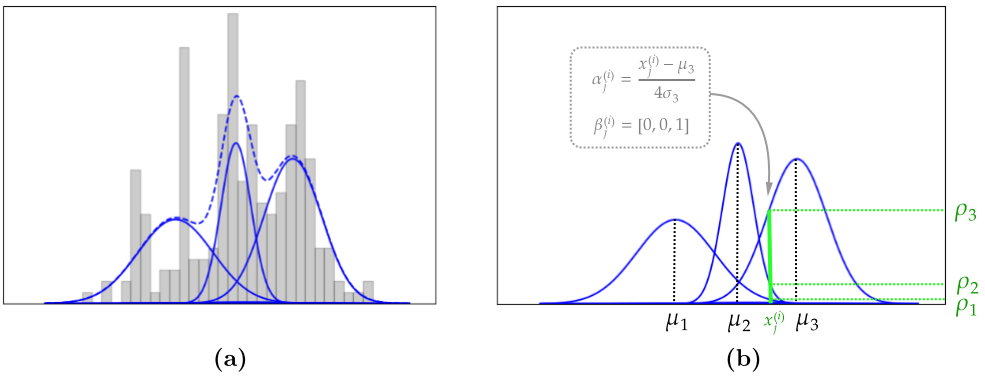


Figure 4.3: Mode specific normalization. The black histogram represents the distribution of a simulated column. The blue solid curves represent the different components of the Gaussian mixture. The blue dashed curve is the sum of all components.

When all categorical and numerical columns are transformed, a row i becomes:

$$T(\mathbf{x}_d^{(i)}) = \mathbf{x}_t^{(i)} = \alpha_1^{(i)} \oplus \boldsymbol{\beta}_1^{(i)} \oplus \cdots \oplus \alpha_k^{(i)} \oplus \boldsymbol{\beta}_k^{(i)} \oplus \boldsymbol{\delta}_1^{(i)} \oplus \cdots \oplus \boldsymbol{\delta}_l^{(i)} \tag{4.7}$$

4.1.2 Conditional vector

Recall that CTGAN uses a conditional GAN framework to address the problem of imbalanced categorical columns. Indeed, in a vanilla GAN framework, we randomly sample the training batch $\mathbf{x}_d$ from $p_d(\mathbf{x}_d)$ and thus rows with minor categories are under-represented, as they also are in the original data. However, in a conditional framework, we do not sample $\mathbf{x}_d$ directly from $p_d(\mathbf{x}_d)$ but instead from $p_d(\mathbf{x}_d|\mathbf{c})$. Therefore, we can sample a batch of conditional vectors $\mathbf{c}$ such that all categories get a fair representation in the training data.

Suppose that, during training, we want G to produce synthetic rows $\tilde{\mathbf{x}}_d$ with the v^*th category of the j^*th attribute. As the data is transformed by T, we actually want G to produce sample rows $\tilde{\mathbf{x}}_t$ with the v^*th component of $\tilde{\boldsymbol{\delta}}_{j^*}$ equal to 1. We would thus like $\mathbf{c}$ to express the condition $(\tilde{\boldsymbol{\delta}}_{j^*})_{v^*} = 1$. To do this, CTGAN defines for each $\tilde{\boldsymbol{\delta}}_j$ a corresponding *mask* vector $\mathbf{m}_j$ of same dimension $|X_j|$. The vth component in $\mathbf{m}_j$ refers to category v of X_j. For a given $\mathbf{m}_j$, each component $(\mathbf{m}_j)_v$ is computed as:

$$(\mathbf{m}_j)_v = \begin{cases} 1 & \text{if } j = j^* \text{ and } v = v^* \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

The conditional vector $\mathbf{c}$ is then given by $\mathbf{m}_1 \oplus \cdots \oplus \mathbf{m}_l$. A toy example is given in Figure 4.4 (b). Consider a table with only 2 categorical columns : column 1 has possible categories $\{A, B\}$ and column 2 has possible categories $\{U, V, W\}$. Suppose that we want to generate samples with the 2^{nd} category V of the 2^{nd} column. In this case, $\mathbf{c}$ has to express the condition $(\tilde{\boldsymbol{\delta}}_2)_2 = 1$. Following equation 4.13, we obtain $\mathbf{m}_1 = [0, 0]$, $\mathbf{m}_2 = [0, 1, 0]$ and $\mathbf{c} = \mathbf{m}_1 \oplus \mathbf{m}_2 = [0, 0, 0, 1, 0]$.

Figure 4.4: Conditional vector of CTGAN

In practice, to provide more training opportunities on minor categories, CTGAN leverages *training by sampling*. This is a procedure followed by S to sample a batch of conditional vectors $\mathbf{c}$ and the corresponding training rows $\mathbf{x}_d$ such that each category in the training data is represented proportionally to the logarithm of its frequency in the original data. This greatly reduces the frequency imbalance between major and minor categories. The procedure for a single $\mathbf{c}$ is described as follow in [1] :

1. Create l zero-filled *mask* vectors $\mathbf{m}_j$ (one *mask* for each of the l categorical columns).
2. Randomly select a categorical column X_{j^*} out of all categorical columns, with equal probability.
3. Construct a PMF (probability mass function) across the categories of X_{j^*} such that the probability mass of each category is the logarithm of its frequency in X_{j^*} .
4. Selected randomly a category v^* according to the PMF above.
5. Set the v^*th component of $\mathbf{m}_{j^*}$ to 1.
6. Compute $\mathbf{c}$ as $\mathbf{m}_1 \oplus \cdots \oplus \mathbf{m}_l$.

4.1.3 Generator

The network architecture of the generator of CTGAN is represented on Figure 4.5. G is a fully connected MLP with two hidden layers. The input $\mathbf{h}_1$ of the second hidden layer is obtained by concatenating the output of the first hidden layer with its input $\mathbf{h}_0 = \mathbf{z} \oplus \mathbf{c}$. We thus have : $\mathbf{h}_1 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|\to 256}(\mathbf{h}_0))) \oplus \mathbf{h}_0$. Similarly, $\mathbf{h}_2$ is obtained as $\texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|+256\to 256}(\mathbf{h}_1))) \oplus \mathbf{h}_1$. This kind of skip connections allows both a better gradient flow and a better crossing of the features.

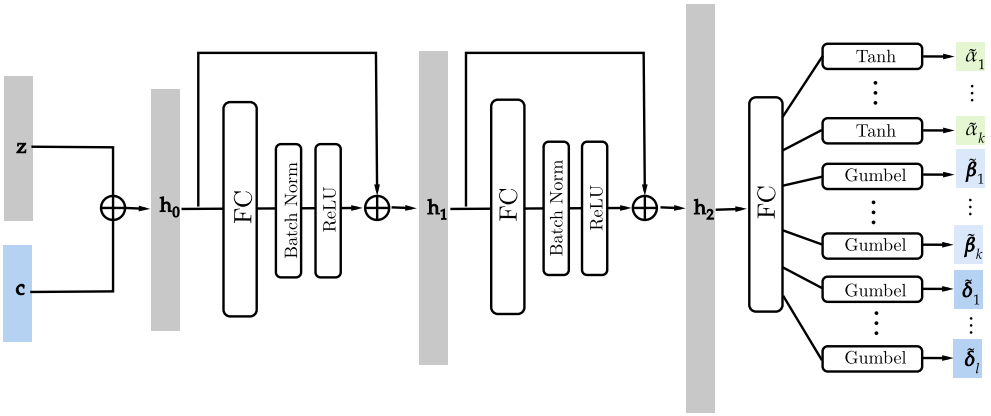


Figure 4.5: Architecture of G in CTGAN

Note that G has an individual output dense layer with the adequate activation for each component of $\tilde{\mathbf{x}}_t$: `tanh` for the scalar values $\tilde{\alpha}$ and `Gumbel softmax`[1] for the vectors $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\delta}}$.

In CTGAN, the loss of G has two terms : $\mathcal{L}_G = \mathcal{L}_{WGAN} + \mathcal{L}_{cond}$. The term $\mathcal{L}_{WGAN}$ is the traditional loss of G in a WGAN-GP framework and the term $\mathcal{L}_{cond}$ is the average cross entropy CE between $\tilde{\boldsymbol{\delta}}_{\boldsymbol{j^*}}$ and $\mathbf{m}_{j^*}$. Therefore, $\mathcal{L}_{cond}$ encourages[2] G to produce synthetic rows that match the condition given by $\mathbf{c}$. For a batch of m noise vectors $\mathbf{z}$ and a batch of m conditional vectors $\mathbf{c}$, we have:

$$\mathcal{L}_{WGAN} = \frac{1}{m} \sum_{i=1}^{m} \Big(- D(G(\mathbf{z}^{(i)}|\mathbf{c}^{(i)})) \Big) \tag{4.9}$$

$$\mathcal{L}_{cond} = \frac{1}{m} \sum_{i=1}^{m} \Big(CE(\mathbf{m}_{j^*}^{(i)}, \tilde{\boldsymbol{\delta}}_{j^*}^{(i)}) \Big) \tag{4.10}$$

[1] A regular `softmax` can only produce a distributed representation over the categories of an attribute and would thus need an additional `argmax` sampling operation to produce proper one-hot vectors. This sampling operation is not differentiable and blocks the gradient flow. On the other hand, `Gumbel softmax` makes use of a clever reparameterization trick [18] to provide a differentiable approximation of `argmax` and therefore, it is preferred over regular `softmax` by CTGAN to generate one-hot encoded vectors $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\delta}}$.

[2] Due to the conditional nature of the framework, G is already encouraged to produce synthetic rows that match the condition given by $\mathbf{c}$. However, adding the term $\mathcal{L}_{cond}$ enforces it more directly.

4.1.4 Discriminator

The network architecture of the discriminator of CTGAN is represented on Figure 4.6. Like G, D is a fully connected MLP with two hidden layers. However, it has no skip connections, no batch normalization and uses `Dropout` [28] and `LeakyReLU` activations instead of standard `ReLU`. Besides, D has a PACGAN [25] structure: the input $\mathbf{h}_0$ to the first layer is the concatenation of p samples $\mathbf{x}_t$ and the p corresponding conditional vectors $\mathbf{c}$. This is known as *packing* and is a form of batch discrimination [9]. It allows D to look at multiple samples simultaneously and it thus helps preventing mode collapse.

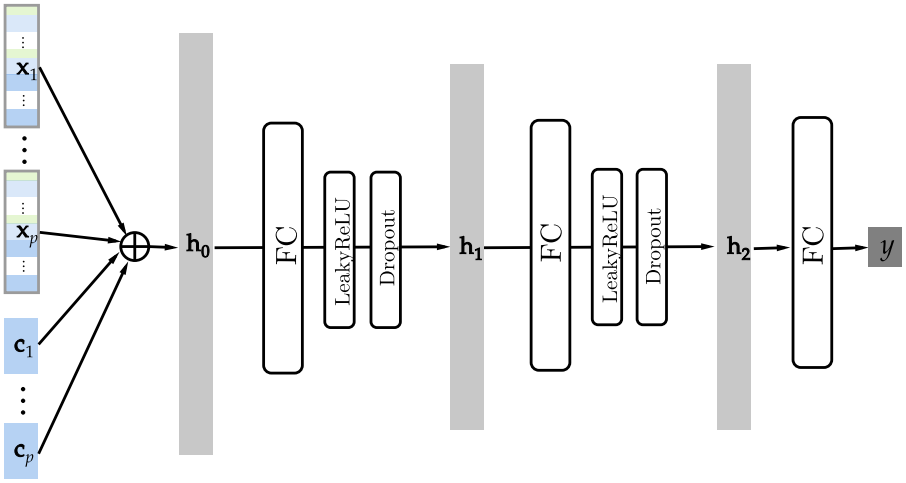


Figure 4.6: Architecture of D in CTGAN

The output of the first hidden layer is $\mathbf{h}_1 = \texttt{Drop}(\texttt{Leaky}(\texttt{FC}_{|10\mathbf{c}|+10|\mathbf{z}|\to 256}(\mathbf{h}_0)))$. Similarly, $\mathbf{h}_2$ is obtained as $\texttt{Drop}(\texttt{Leaky}(\texttt{FC}_{256\to 256}(\mathbf{h}_1)))$. There is no activation function in the output layer, y is simply obtained as $\texttt{FC}_{256\to 1}(\mathbf{h}_2)$ and represents an assessment of how real the input samples are.

The loss function of D is the standard loss function in a conditional WGAN-GP framework. For input batches of m samples $\mathbf{x}_t$ and m samples $\tilde{\mathbf{x}}_t$, $\mathcal{L}_D$ is defined as:

$$\mathcal{L}_D = -\frac{1}{m/p}\sum_{i=1}^{m/p}\left(D({}_p\mathbf{x}_t^{(i)}|\,{}_p\mathbf{c}^{(i)}) - D({}_p\tilde{\mathbf{x}}_t^{(i)}|\,{}_p\mathbf{c}^{(i)})\right) + \lambda\cdot\sum_{i=1}^{m/p}\left(\|\nabla_{{}_p\hat{\mathbf{x}}^{(i)}} D({}_p\hat{\mathbf{x}}^{(i)}|\,{}_p\mathbf{c}^{(i)})\| - 1\right)^2 \tag{4.11}$$

where ${}_p\mathbf{x}_t$, ${}_p\tilde{\mathbf{x}}_t$ and ${}_p\mathbf{c}$ are respectively the concatenation of p samples $\mathbf{x}_t$, $\tilde{\mathbf{x}}_t$ and $\mathbf{c}$, and where ${}_p\hat{\mathbf{x}}$ is obtained as:

$${}_p\hat{\mathbf{x}} = \epsilon\cdot\,{}_p\mathbf{x}_t + (1-\epsilon)\cdot\,{}_p\tilde{\mathbf{x}}_t \qquad \epsilon\sim\mathcal{U}(0,1) \tag{4.12}$$

4.2 Enhanced CTGAN

The global architecture of our tabular data synthesizer is represented in Figure 4.7. It is naturally very close to the architecture of CTGAN, but with some key additions.

Our most significant contribution is the new encoding-decoding structure. The idea is to provide a more meaningful representation of the categories in the categorical columns and of the modes in the numerical columns. Indeed, recall that CTGAN represents those respectively as one-hot vectors $\boldsymbol{\delta}$ and one-hot vectors $\boldsymbol{\beta}$. These one-hot vectors actually carry very little information relative to their dimension. Moreover, CTGAN transforms all columns independently from one another which means that it is entirely up to G to capture all the intricate relationships between the different columns. We designed an encoder that maps $\boldsymbol{\delta}$ and $\boldsymbol{\beta}$ to a common latent space. As a result, the encoded representations $\boldsymbol{\delta}_e$ and $\boldsymbol{\beta}_e$ will be able to capture relationships between all types of columns and thus carry more meaningful information than simple one-hot vectors. In a way, we are "pre-chewing" the work of G. Besides, since the latent space is continuous, it is also easier to model for G.

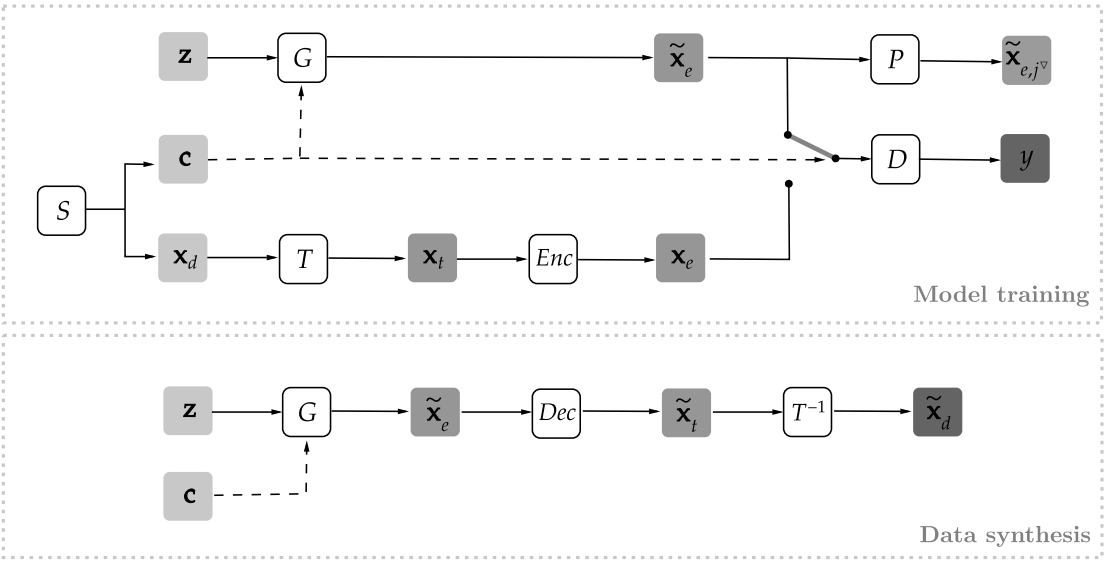


Figure 4.7: Global architecture of our enhanced CTGAN

The encoder is trained on real transformed data $\mathbf{x}_t$. It is positioned after the transformer T and outputs $\mathbf{x}_e = Enc(\mathbf{x}_t)$. Given a batch of m noise vectors $\mathbf{z}$ and a batch of m conditional vectors $\mathbf{c}$, G produces a batch of m encoded outputs $\tilde{\mathbf{x}}_e$. The samples $\mathbf{x}_e$ and $\tilde{\mathbf{x}}_e$ are then passed to D as usual. The samples $\tilde{\mathbf{x}}_e$ are also passed to an auxiliary network which we call the Predictor P. P is trained on the original encoded data to predict the target attribute $\mathbf{x}_{e,j^\triangledown}$ of any sample $\mathbf{x}_e$ given all its other attributes. If P can then recover the target attribute $\tilde{\mathbf{x}}_{e,j^\triangledown}$ of generated data $\tilde{\mathbf{x}}_e$ given its other attributes, it means that G produces samples that are coherent. To encourage this, P provides feedback to G through the *prediction loss*. We finally train a decoder to recover $\mathbf{x}_t$ given some real encoded data $\mathbf{x}_e$. The decoder is mainly necessary when we want to synthesize data $\tilde{\mathbf{x}}_d$ in the same format as the original table. Indeed, G produces $\tilde{\mathbf{x}}_e$ and then $\tilde{\mathbf{x}}_d$ is obtained as $T^{-1}(Dec(\tilde{\mathbf{x}}_e))$.

4.2.1 Data transformation

The purpose of all transformations is to represent the data in a usable way in order to get the most out of it. Nevertheless, each transformation may also introduce some approximation and/or bias with respect to the original data and as we compose many transformations, there is a risk that these biases get amplified. With our encoder, we add yet another layer of transformations; we would thus like the previous transformations made by T yo be as accurate and qualitative as possible.

Regarding categorical columns, we do not modify T: we just use one-hot encoding as in CTGAN. Concerning numerical columns, recall that CTGAN fits a VGM model on each one of them in order to identify the modes of its distribution and then apply *mode-specific normalization*. However, in the course of our experiences, we have observed that the VGM model of CTGAN has some trouble fitting (1) long-tail or highly skewed distributions and (2) complex distributions with many dense and local modes.

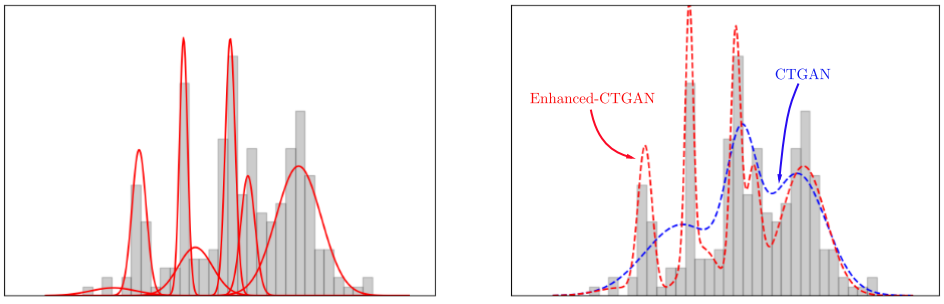
To address the first issue, we apply a log transform on highly skewed numerical columns, as in CTAB-GAN [27]. Let X_j be such a column with lower bound l_b. We replace each of its values $x_j^{(i)}$ by :

$$\begin{cases} \log(x_j^{(i)}) & \text{if } l_b > 0 \\ \log(x_j^{(i)} - l_b + \epsilon) & \text{if } l_b \leq 0, \text{ where } \epsilon > 0 \end{cases} \tag{4.13}$$

To address the second issue, we modify the VGM model such that it can fit Gaussian mixtures recursively. For a given column, the procedure is as follow:

1. Fit a first Gaussian mixture. Suppose that this mixture has k modes.
2. Divide the data samples into k sets corresponding to the modes of the mixture in 1.
3. For each set, fit a new Gaussian mixture if the considered set includes a sufficient number of samples and if the standard deviation of theses samples is above a certain threshold (this is to avoid over-fitting).
4. Repeat step 2 and 3 until the weights of the Gaussian components are below a defined threshold or the maximum recursion depth is attained.

As Figure 4.8 shows, this recursive scheme makes the VGM model more flexible and enables it to capture more local modes.



(a) (b)

Figure 4.8: Recursive VGM model of Enhanced CTGAN

4.2.2 Encoder - Decoder

As we have explained, after transforming the data with T, we obtain rows $\mathbf{x}_t$ with some numerical values α and a bunch of one-hot vectors $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$. These one-hot vectors are surely usable for learning, as demonstrated by the good performance of CTGAN and CTAB-GAN, but we feel that they could contain more rich information that could benefit the tabular synthesizer. Such information could be how one category relates to some other category of a different categorical attribute, or maybe to some mode of a numerical attribute. It could also be how the modes of different numerical attributes are related to one another. The goal of the encoder, as illustrated in Figure 4.9, is thus to provide meaningful data representations $\boldsymbol{\beta}_e$ and $\boldsymbol{\delta}_e$ that encode useful information. The goal of the decoder is to recover the one hot vectors $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$ from $\boldsymbol{\beta}_e$ and $\boldsymbol{\delta}_e$. Note that in the following explanations, we will use the term "category" loosely to designate any value of an attribute that is expressed as a one-hot vector: it may thus refer to actual categories of categorical attributes, to modes of numerical attributes, or to both.

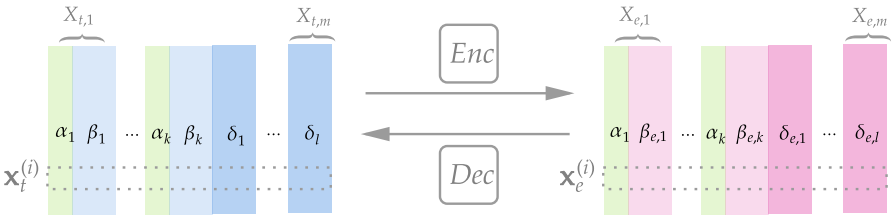


Figure 4.9: Encoding-decoding scheme in our enhanced CTGAN

Our Encoder is an adaptation of Word2Vec [29], a very popular word embedding scheme in the domain of language modelling. The idea is that if two *target* words often appear in a same *context*, then the words should have a similar vector representation. In the case of tabular data, this would mean that if two *target* categories occur with the same *context* categories in many rows of the data, their vector representations $\boldsymbol{\delta}_e$ (or $\boldsymbol{\beta}_e$) should be similar. To construct such similar representations, we will train a MLP with a single hidden layer to perform some artificial task and then use the values of the neurons in the hidden layer as an embedding for the input of the network. We will call this model M. Given an input *target* category $\mathbf{t}$, the task of M is to output for each category the probability that it is a *context* category of $\mathbf{t}$, *i.e.*, the probability that it appears in the same row as $\mathbf{t}$. The intuition is the following: if two *target* categories $\mathbf{t}$ that have similar *context* categories $\mathbf{c}$ are both fed to M, the neural network should produce similar outputs. One way it can do that is if it learns similar inner representation for $\mathbf{t}$ on the hidden layer. Because any *target* category can also be considered as a *context* category, the network should also produce similar representations for $\mathbf{c}$ and $\mathbf{t}$.

The encoding workflow is represented on Figure 4.10. To train M, we first need training pairs $(\mathbf{t}, \mathbf{c})$ composed of a *target* category and one of its *context* category. M requires $\mathbf{t}$ and $\mathbf{c}$ to be one-hot vectors of dimension equal to the total number of categories n_{cat} across all attributes. To create these training examples, we start from the table obtained at the output of the transformer T (a). From this table, we keep only the one-hot encoded columns and thus obtain the table Z_0 (b). We then build the table $Z = Z_0^T Z_0$ (c). Since Z_0 is a complete disjunctive table, $Z = Z_0^T Z_0$ is a Burt table.

Z contains all the contingency tables of the attributes taken in pairs. For instance, let us suppose an original table with only 7 rows (d). Its attribute X_i has possible categories $\{A, B\}$ and its attribute X_j has possible categories $\{U, V, W\}$. The block $Z_{i,i}$ contains on its diagonal the frequencies of the categories A and B in the original table: A appears in 3 rows, while B appears in the 4 others. The block $Z_{i,j}$ gives the conditional frequencies of the categories of attribute X_j, given the categories of attribute X_i. We can see in (d) that the category U appears in one row with B, the category V in 3 rows with B, etc.

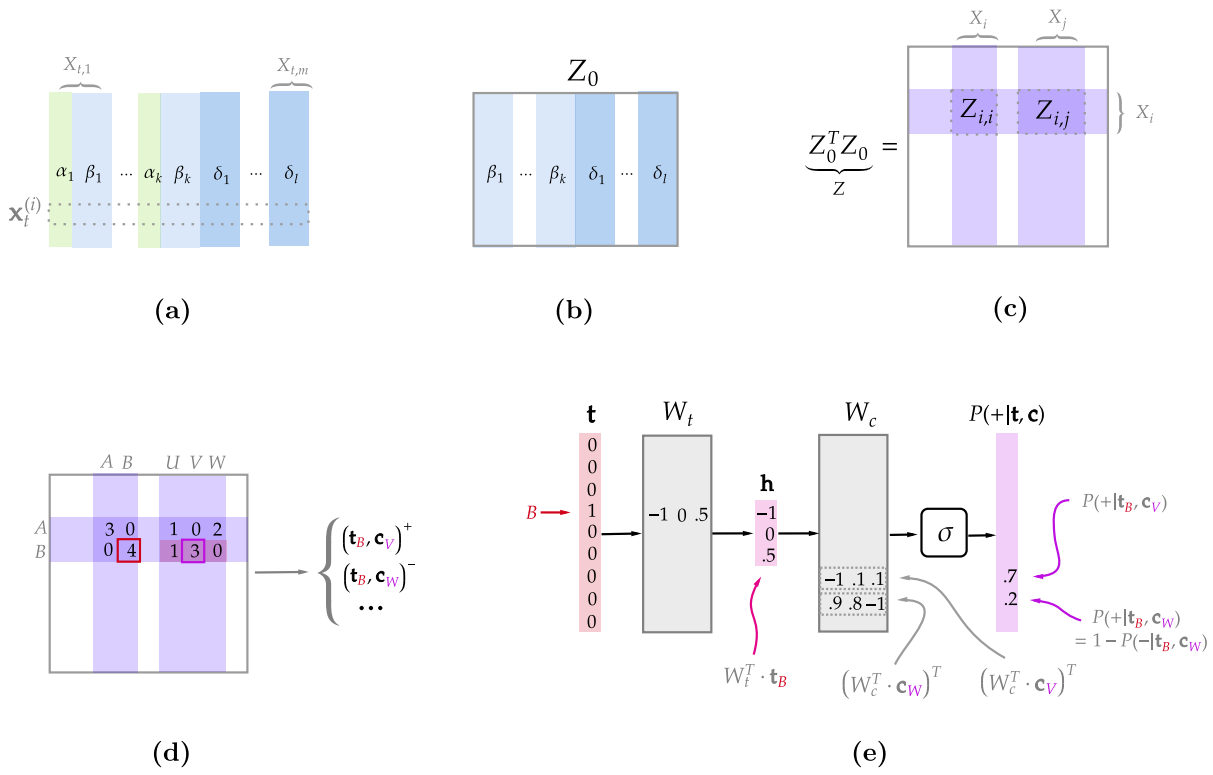


Figure 4.10: Caption

Once the Burt table is constructed, it is easy to create training pairs $(\mathbf{t}, \mathbf{c})$. Note that for a more efficient training of M, we actually want to create both positive pairs where $\mathbf{c}$ is indeed a *context* category of $\mathbf{t}$ and negative pairs where $\mathbf{c}$ never appears in the same row as $\mathbf{t}$. The procedure to sample such training pairs is as follow:

1. Randomly select a *target* attribute among all possible attributes, with equal probability. Suppose we have selected X_i.
2. Construct a PMF across the categories of X_i such that the probability mass of each category is the logarithm of its frequency. The frequencies are given on the diagonal of $Z_{i,i}$. We take the logarithm to reduce potential imbalance between minor and major categories.
3. Select randomly a *target* category according to the PMF above. In our example, suppose we have selected the category B. The target vector $\mathbf{t}_B$ will thus be a one-hot vector of dimension $n_{cat} \times 1$ indicating category B.

4. Randomly select a *context* attribute among all possible attributes, with equal probability. Suppose we have selected X_j.
5. Construct a conditional PMF across the categories of X_j such that the probability mass of each category is the logarithm of its frequency given the selected *target* category. The conditional frequencies are given on the row of $Z_{i,j}$ corresponding to the *target* category.
6. Select randomly a positive *context* category according to the PMF above. In our example, we have selected the category V as its conditional frequency given B was the highest. The *context* vector $\mathbf{c}_V$ will thus be a one-hot vector of dimension $n_{cat} \times 1$ indicating category V.
7. Select randomly a negative *context* category. This is a category for which the probability mass of the conditional PMF is zero. In our example, we have selected the category W.

Let us now detail what happens when we feed our example positive pair $(\mathbf{t}_B, \mathbf{c}_V)^+$ and negative pair $(\mathbf{t}_B, \mathbf{c}_W)^-$ to M. An illustration is given in Figure 4.10 (e). In any cases, the input vector is $\mathbf{t}_B$. It is multiplied by a weight matrix W_t^T to give the vector $\mathbf{h}$. If we feed $\mathbf{t}_B$ to a trained model, $\mathbf{h}$ will be our encoded representation for category B. The weight matrix W_t has size $n_{cat} \times n_{enc}$; each of its rows corresponds to a category and n_{enc} is the dimension we want our encoding to have. Since $\mathbf{t}$ is one-hot encoded, $\mathbf{h} = W_t^T \mathbf{t}_B$ is just the transpose of the row in W_t that corresponds to category B. This means that when the model is fully trained, we can use W_t as a look-up table where each row gives the encoding of one *target* category.

Next, $\mathbf{h}$ is multiplied by a second weight matrix W_c of size $n_{cat} \times n_{enc}$. Similarly to W_t, each row of W_c represents the encoding of a *context* category. This vector $W_c\mathbf{h}$ is directly passed through a sigmoid activation function to obtain a final output vector of size $n_{cat} \times 1$ with all components between 0 and 1. In particular, the component of the output vector associated with the *context* category V is computed as the sigmoid of the scalar product between the encoded representation of $\mathbf{t}_B$ and the encoded representation of $\mathbf{c}_V$: $\sigma((W_c^T \mathbf{c}_V)^T \cdot \mathbf{h}) = \sigma((W_c^T \mathbf{c}_V)^T \cdot (W_t^T \mathbf{t}_B))$.

This scalar product is a measure of similarity (an unscaled cosine) between $W_c^T \mathbf{c}_V$ and $W_t^T \mathbf{t}_B$ and thus, when passed through a sigmoid, it can be interpreted as the probability that $W_c^T \mathbf{c}_V$ is similar to $W_t^T \mathbf{t}_B$, or in other words, the probability $P(+|\mathbf{t}_b, \mathbf{c}_V)$ that V is a context category of B. The component of the output vector associated with the negative *context* category W is $P(+|\mathbf{t}_b, \mathbf{c}_W) = 1 - P(-|\mathbf{t}_b, \mathbf{c}_W) = \sigma((W_c^T \mathbf{c}_W)^T \cdot (W_t^T \mathbf{t}_B))$

The goal of M is to maximize $P(+|\mathbf{t}_b, \mathbf{c}_V)$ and $P(-|\mathbf{t}_b, \mathbf{c}_W)$. If we generalize this result for a batch of positive training pairs $(\mathbf{t}^{(i)}, \mathbf{c}^{(i)})^+$, $i \in \{1, ..., m\}$, and a batch of negative training pairs $(\mathbf{t}^{(j)}, \mathbf{c}^{(j)})^-$, $j \in \{1, ..., n\}$, the loss function can be written as:

$$\mathcal{L}_M = -\frac{1}{m} \sum_{i=1}^{m} \log \left(P(+|\mathbf{t}^{(i)}, \mathbf{c}^{(i)}) \right) - \frac{1}{n} \sum_{j=1}^{n} \log \left(P(-|\mathbf{t}^{(j)}, \mathbf{c}^{(j)}) \right) \tag{4.14}$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \log \left(\sigma\big((W_c^T \mathbf{c}^{(i)})^T \cdot (W_t^T \mathbf{t}^{(i)})\big) \right) - \frac{1}{n} \sum_{j=1}^{n} \log \left(\sigma\big((W_c^T \mathbf{c}^{(j)})^T \cdot (W_t^T \mathbf{t}^{(j)})\big) \right) \tag{4.15}$$

Note that taking the logarithm of the probabilities does not change the training objective.

The hardest part of learning the encoder is naturally to train M. Once M is trained on the transformed original data, we can just select either W_t or W_c to serve as a look-up table whenever we need to encode some one-hot vector of a category $\boldsymbol{\delta}$ or of a mode $\boldsymbol{\beta}$. In our enhanced CTGAN, we use W_t. Finally, we normalize the encoded columns to $[-1, 1]$.

Because of the architecture of M and of how we sample its training examples, we see that the encoded representation $\boldsymbol{\delta}_e$ of one category or the encoded representation $\boldsymbol{\beta}_e$ of one mode will depend to some extend on all other categories and modes and will thus contain useful information on how one category or mode relates to the other. This is what makes such a representation more meaningful, at least in our opinion, than one-hot vectors.

Now that we have a functioning encoder, we still need to build its counter-part: the decoder. The architecture of the decoder is almost identical to the architecture of the generator in CTGAN (fully connected MLP with skip connections, Batch-norm and ReLU activations in the hidden layers, and tanh/Gumbel activation in the output layer). This makes sense since both the decoder and CTGAN's G must generate samples $\tilde{\mathbf{x}}_t \sim p_t$. However, their inputs and goals are very different. Whereas CTGAN's G is fed with $\mathbf{z} \oplus \mathbf{c}$ and must try to output convincingly real samples $\tilde{\mathbf{x}}_t$ given $\mathbf{c}$, the decoder in our enhanced CTGAN receives as input encoded rows of the original data $\mathbf{x}_e = Enc(\mathbf{x}_t)$ and must try to output $\tilde{\mathbf{x}}_t = \mathbf{x}_t$. In fact, because the values $\alpha_1 \cdots \alpha_k$ in $\mathbf{x}_t$ are not affected by the encoder and thus do not need to be decoded either, we only pass reduced rows $[\boldsymbol{\beta}_{e,1} \oplus \cdots \oplus \boldsymbol{\beta}_{e,k} \oplus \boldsymbol{\delta}_{e,1} \oplus \cdots \oplus \boldsymbol{\delta}_{e,l}]$ to the decoder. Its goal is then to have $\tilde{\boldsymbol{\beta}}_j = \boldsymbol{\beta}_j$ for $j \in \{1, ..., k\}$ and $\tilde{\boldsymbol{\delta}}_j = \boldsymbol{\delta}_j$ for $j \in \{1, ..., l\}$.

One way to measure the discrepancy between one-hot vectors is cross-entropy CE. For a batch of m reduced training samples $\mathbf{x}_e$, the loss function of the decoder is thus given by:

$$\mathcal{L}_{dec} = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{k} \sum_{j=1}^{k} \left(CE(\tilde{\boldsymbol{\beta}}_j^{(i)}, \boldsymbol{\beta}_j^{(i)}) \right) + \frac{1}{l} \sum_{j=1}^{l} \left(CE(\tilde{\boldsymbol{\delta}}_j^{(i)}, \boldsymbol{\delta}_j^{(i)}) \right) \right) \tag{4.16}$$

4.2.3 Conditional vector

As in CTAB-GAN [27], we extend the conditional vector to modes of numerical data. This can be beneficial if some of the numerical columns also present strong "imbalance", *i.e.*, most of the samples belong to one major mode and only a few to other minor modes. It is all the more interesting if the relevant information for predicting the target attribute lies in the under-represented modes. Therefore, we define a *mask* vector $\mathbf{m}$ for each attribute, numerical or categorical. The *mask* $\mathbf{m}_j$ corresponds to $\boldsymbol{\beta}_j$ if $j \leq k$ and to $\boldsymbol{\delta}_{j-k}$ if $j > k$. This is illustrated on Figure 4.11 (a).

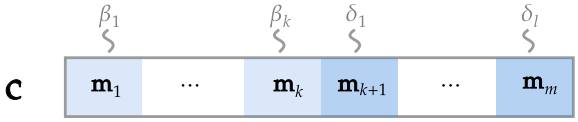


Figure 4.11: Extended conditional vector for our enhanced CTGAN

4.2.4 Generator

Because G does not have to generate one-hot vectors anymore, we only use `tanh` activations in the output layer. For the rest, we keep the same generator architecture as CTGAN but we modify and add terms in the loss function. The loss of G in our enhanced model is defined as :

$$\mathcal{L}_G = \mathcal{L}_{WGAN} + \mathcal{L}_{cond} + \mathcal{L}_{fm} + \mathcal{L}_P \tag{4.17}$$

Where $\mathcal{L}_{WGAN}$ is the traditional WGAN-GP loss, $\mathcal{L}_{cond}$ is the term that penalizes G if it produces samples that do not match $\mathbf{c}$, $\mathcal{L}_{fm}$ is the feature-matching loss and $\mathcal{L}_P$ is the feedback from the predictor. In the following explanation, all the losses are defined given a batch of m noise samples and a batch of m conditional vectors $\mathbf{c}$.

$\boldsymbol{\mathcal{L}_{WGAN}}$:

The only difference with CTGAN is that G outputs samples $\tilde{\mathbf{x}}_e$ and not $\tilde{\mathbf{x}}_t$.

$$\mathcal{L}_{WGAN} = \frac{1}{m}\sum_{i=1}^{m}\Big(- D(G(\mathbf{z}^{(i)}|\mathbf{c}^{(i)}))\Big) = \frac{1}{m}\sum_{i=1}^{m}\Big(- D(\tilde{\mathbf{x}}_e{}^{(i)}|\mathbf{c}^{(i)})\Big) \tag{4.18}$$

$\boldsymbol{\mathcal{L}_{cond}}$:

This term was introduced by CTGAN. Recall from equation 4.10 that we had to compute $CE(\mathbf{m}_{j^*}^{(i)}, \tilde{\boldsymbol{\delta}}_{j^*}^{(i)})$ with $j^* \in \{1, ..., l\}$. Now, because we have extended the conditional vector to all attributes, not only the l categorical ones, we also have to extend $\mathcal{L}_{cond}$ to all attributes. In particular, when $\mathbf{m}_{j^*}^{(i)}$ refers to a numerical attribute, we will have to compute $CE(\mathbf{m}_{j^*}^{(i)}, \tilde{\boldsymbol{\beta}}_{j^*}^{(i)})$ with $j^* \in \{1, ..., k\}$. Note that the indices are defined in a slightly different way with the addition of the k numerical attributes. To be rigorous, when $\mathbf{m}_{j^*}^{(i)}$ refers to a categorical attribute, we thus compute $CE(\mathbf{m}_{j^*}^{(i)}, \tilde{\boldsymbol{\delta}}_{j^*-k}^{(i)})$ with $j^* \in \{k+1, ..., m\}$. Also, as we have just mentioned, G outputs $\tilde{\mathbf{x}}_e$ and not $\tilde{\mathbf{x}}_t$, meaning that we only have access to the encoded $\tilde{\boldsymbol{\beta}}_{e,j^*}^{(i)}$ and $\tilde{\boldsymbol{\delta}}_{e,j^*-k}^{(i)}$. To obtain the one-hot $\tilde{\boldsymbol{\beta}}_{j^*}^{(i)}$ and $\tilde{\boldsymbol{\delta}}_{j^*-k}^{(i)}$, we will thus first need to feed $\tilde{\mathbf{x}}_e$ to our decoder. We then have:

$$\mathcal{L}_{cond} = \frac{1}{m}\sum_{i=1}^{m}\Big(CE(\mathbf{m}_{j^*}^{(i)}, \tilde{\mathbf{x}}_{t,j^*}^{(i)})\Big) \quad \text{where } \tilde{\mathbf{x}}_{t,j^*}^{(i)} = \begin{cases} \tilde{\boldsymbol{\beta}}_{j^*}^{(i)} & \text{if } j^* \leq k \\ \tilde{\boldsymbol{\delta}}_{j^*-k}^{(i)} & \text{if } j^* > k \end{cases} \tag{4.19}$$

$\boldsymbol{\mathcal{L}_{fm}}$:

The goal of the feature matching loss is to encourage G to produce synthetic data that matches some given features of the real data, hence the name. The loss is itself composed of 4 simple terms : $\mathcal{L}_{fm} = \mathcal{L}_{mean-D} + \mathcal{L}_{sd-D} + \mathcal{L}_{mean-G} + \mathcal{L}_{sd-G}$.

The two first terms $\mathcal{L}_{mean-D}$ and $\mathcal{L}_{sd-D}$ correspond to the *information loss* used in [22] and [27]. The idea is to compare features extracted from the last layer of the discriminator. Indeed, these are the features on the basis of which D decides if its inputs are rather real or synthetic. We would thus naturally like G to output synthetic data that presents similar features as the real data when passed to the last layer of D. Of course, this should happen on its own if the GAN converges, but we want to enforce it more explicitly.

We can thus encourage G to produce data that will match some specific statistics of the extracted features, for instance, the mean and the standard deviation. This is exactly the role of $\mathcal{L}_{mean-D}$ and $\mathcal{L}_{sd-D}$. They are defined as:

$$\mathcal{L}_{mean-D} = \left\| \frac{1}{m}\sum_{i=1}^{m} D_{\mathbf{h}_2}(\tilde{\mathbf{x}}_e^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} D_{\mathbf{h}_2}(\mathbf{x}_e^{(i)}) \right\| \tag{4.20}$$

$$\mathcal{L}_{sd-D} = \left\| \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(D_{\mathbf{h}_2}(\tilde{\mathbf{x}}_e^{(i)}) - \overline{D_{\mathbf{h}_2}(\tilde{\mathbf{x}}_e)}\right)^2} - \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(D_{\mathbf{h}_2}(\mathbf{x}_e^{(i)}) - \overline{D_{\mathbf{h}_2}(\mathbf{x}_e)}\right)^2} \right\| \tag{4.21}$$

where $D_{\mathbf{h}_2}$ represents the extracted features $\mathbf{h}_2$ at the last hidden layer of D and $\overline{D_{\mathbf{h}_2}(\mathbf{x})} = \frac{1}{m}\sum_{i=1}^{m} D_{\mathbf{h}_2}(\mathbf{x}^{(i)})$.

$\mathcal{L}_{mean-G}$ and $\mathcal{L}_{sd-G}$ follow the same idea but instead of extracting the features from the last layer of D, we take the features directly at the output of G. In fact, we encourage G to produce synthetic data with attributes that have the same mean and standard deviation as attributes of the real data. Note that this only acts columns-wise, it does not necessarily encourage G to produce rows issued from the joint distribution of the real columns. $\mathcal{L}_{mean-G}$ and $\mathcal{L}_{sd-G}$ are expressed as follow:

$$\mathcal{L}_{mean-G} = \left\| \frac{1}{m}\sum_{i=1}^{m} \tilde{\mathbf{x}}_e^{(i)} - \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_e^{(i)} \right\| \tag{4.22}$$

$$\mathcal{L}_{sd-G} = \left\| \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\tilde{\mathbf{x}}_e^{(i)} - \overline{\tilde{\mathbf{x}}_e}\right)^2} - \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{x}_e^{(i)} - \overline{\mathbf{x}_e}\right)^2} \right\| \tag{4.23}$$

$\boldsymbol{\mathcal{L}_P}$:

This terms is similar to the *classification loss* used in [22], [26] and [27]. The idea is to encourage G to produce samples that are coherent in the sense that the target attribute can be predicted from the others (when possible). To do this, we first train a predictor network P on the original encoded data $\mathbf{x}_e$. P is a small MLP that must output the target attribute $\mathbf{x}_{e,j^\triangledown}$ given as input all other attributes of $\mathbf{x}_e$. If the target attribute is numerical, $\mathbf{x}_{e,j^\triangledown} = \alpha_{e,j^\triangledown} \oplus \boldsymbol{\beta}_{e,j^\triangledown}$; if the target attribute is categorical, $\mathbf{x}_{e,j^\triangledown} = \boldsymbol{\delta}_{e,j^\triangledown-k}$. As $\mathbf{x}_{e,j^\triangledown}$ is encoded in a numerical fashion and because similar attributes should have similar encodings, we can use a simple MSE loss to train P:

$$\mathcal{L}_P = \frac{1}{m}\sum_{i=1}^{m} \left\| \hat{\mathbf{x}}_{e,j^\triangledown}^{(i)} - \mathbf{x}_{e,j^\triangledown}^{(i)} \right\|^2 \tag{4.24}$$

where $\hat{\mathbf{x}}_{e,j^\triangledown}$ is the prediction of P. When training our enhanced CTGAN, P is already trained and we use it on the generated data such that the feedback given to G is in fact:

$$\mathcal{L}_P = \frac{1}{m}\sum_{i=1}^{m} \left\| \hat{\mathbf{x}}_{e,j^\triangledown}^{(i)} - \tilde{\mathbf{x}}_{e,j^\triangledown}^{(i)} \right\|^2 \tag{4.25}$$

4.2.5 Discriminator

We do not bring any changes to the discriminator of CTGAN.

Chapter 5

Experimental setup

5.1 Datasets

We compare CTGAN and our enhanced CTGAN on 4 different datasets which contain a good mix of categorical and numerical attributes. Because we want to test our models in the context of data augmentation and also due to a very limited computational capacity, we choose to work with relatively small datasets (≤ 1000 samples):

- **Adult :** this is a reduced version of the *Adult income* dataset from the United States Census Bureau. The associated task is to predict whether a given adult earns more or less than 50K a year based on attributes such as education level, race, gender, etc. We sampled 1000 rows from the original dataset and grouped some categories of categorical attributes together such that each possible category is represented in at least 5 rows.
- **Heart :** this is the famous Cleveland Heart Disease dataset from the UCI Repository. The goal is to predict whether or not a patient suffers from a cardio vascular disease using 14 attributes such has resting blood pressure, chest pain type, etc.
- **Hcv :** this dataset also comes from the UCI Repository. It contains laboratory values of blood donors together with some demographic values. The target attribute is multi-class and represents the diagnostic associated with each blood sample (normal, suspect, Hepatitis, Fibrosis or Cirrhosis).
- **Student :** the goal here is to predict the final grade of students given a multitude of demographic, social and school related attributes. The associated learning task is thus regression and not classification as for the previous datasets.

The main features of each datasets are summarized in table 5.1.

	# samples	# attributes	# categorical	# numerical	Learning task
Adult	1000	15	9	6	Binary classification
Heart	303	14	9	5	Binary classification
Hcv	589	14	2	12	Multi-class classification
Student	649	30	27	3	Regression

Table 5.1: Main features of the datasets

5.2 Implementation

For CTGAN, we use the model exactly as described in [1]. The authors have provided a `pytorch` implementation of their tabular synthesizer in an open-source python library `CTGAN`[1]. This library is itself part of a broader ecosystem of libraries dedicated to synthetic data generation called the Synthetic Data Vault[2] (SDV). Notably, we also make good use of the library `SDMetrics`[3] which provides a number of metrics to evaluate the quality of synthetic data. The SDV project is carried by the *DATA to AI Lab* team at MIT.

The implementation of our enhanced CTGAN is entirely based on the `pytorch` implementation of the original CTGAN. We duplicated the `CTGAN` library and modified/added the necessary scripts to match our model design. We kept the same parameters for the common part of the architecture to provide a fair comparison. These parameters include numbers of layers in G and in D, layer dimensions, learning rates, etc.

5.3 Evaluation procedure

Recall that we have two objectives: improve CTGAN and use the improved model to augment tabular data. To assess whether theses objectives have been successfully met, we first evaluate the quality of the synthetic data produced by our enhanced CTGAN and draw a comparison with the results obtained by the original CTGAN. Secondly, we use the synthesized data to augment the original data and we evaluate the performance of ML models trained on this augmented data.

We evaluate the tabular synthesizers $\mathcal{S}$ (CTGAN and enhanced CTGAN) on each dataset using a 4-fold cross validation. The procedure is illustrated on Figure 5.1. Let $\mathbf{T}$ be the table corresponding to one of the datasets. $\mathbf{T}$ is randomly divided in 4 equal folds (row-wise). 3 folds form the training data $\mathbf{T}_{train}$ while the remaining fold is the test data $\mathbf{T}_{test}$. We train $\mathcal{S}$ on $\mathbf{T}_{train}$ and then generate $\tilde{\mathbf{T}}_{train}$. We can now evaluate the quality of the synthesized data $\tilde{\mathbf{T}}_{train}$ using 4 types of metrics :

- **Statistical similarity metrics :** these metrics measure the difference between some statistical properties of $\tilde{\mathbf{T}}_{train}$ and $\mathbf{T}_{train}$. The metrics can apply on each column individually, on a couple of columns or directly on the whole table.
- **Likelihood metrics :** these metrics evaluate the (log) likelihood of $\tilde{\mathbf{T}}_{train}$ given $\mathcal{M}$, a model fitted on the real data $\mathbf{T}_{train}$.
- **Detection metrics :** these metrics reflect the ability of a trained classifier to distinguish between rows of $\tilde{\mathbf{T}}_{train}$ and rows of $\mathbf{T}_{train}$.
- **Machine learning efficacy metrics :** these metrics compare the performance of a model $\mathcal{M}_1$ trained on the synthetic data $\tilde{\mathbf{T}}_{train}$ and a model $\mathcal{M}_2$ trained on the real data $\mathbf{T}_{train}$.

[1] https://github.com/sdv-dev/CTGAN
[2] https://sdv.dev/
[3] https://github.com/sdv-dev/SDMetrics

Next, to assess the impact of data augmentation, we create $\mathbf{T}_{aug} = \mathbf{T}_{train} \oplus \tilde{\mathbf{T}}_{train}$. Note that we can add any numbers of rows to $\mathbf{T}_{train}$, we do not necessarily have to add the whole synthetic table $\tilde{\mathbf{T}}_{train}$. We then train a model $\mathcal{M}$ on $\mathbf{T}_{aug}$ and evaluate it on $\mathbf{T}_{test}$. This is to compare with a model $\mathcal{M}$ trained on $\mathbf{T}_{train}$ and evaluated in the same conditions on $\mathbf{T}_{test}$.

Overall, to obtain significant results, we train $\mathcal{S}$ three times at each iteration of the 4-fold cross-validation and sample three $\tilde{\mathbf{T}}_{train}$ for each trained $\mathcal{S}$. Our final results are thus the average taken over $4 \times 3 \times 3 = 36$ measurements.

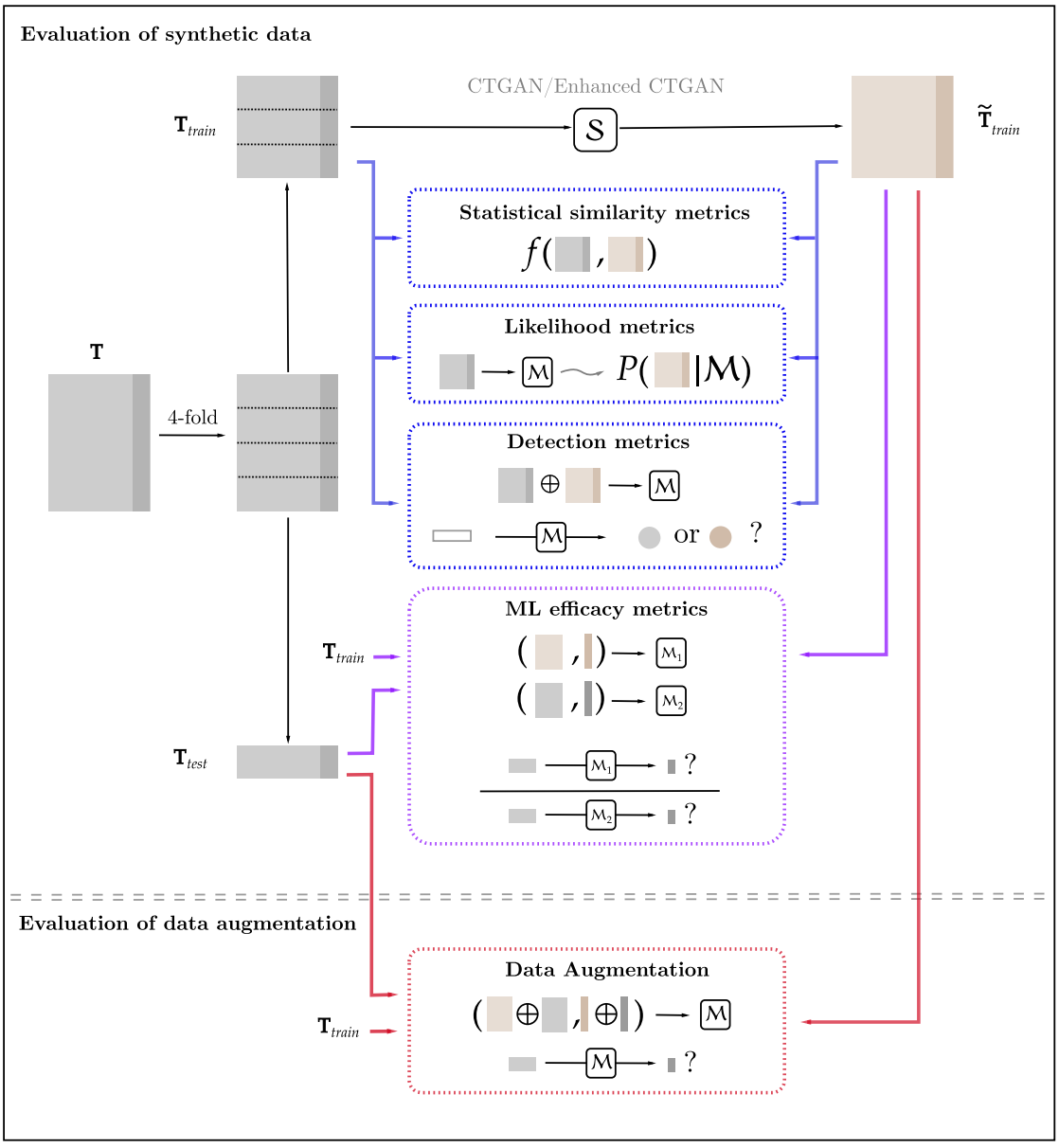


Figure 5.1: Evaluation procedure for synthesized and augmented data

5.3.1 Statistical similarity metrics

Kolmogorov-Smirnov (`KST`) : this metric compares the distribution of numerical columns using a two-sample Kolmogorov-Smirnov test. The statistic of this test, known as the D-statistic, indicates the maximal distance between the empirical CDFs of the distributions. The metric outputs 1 minus the D statistic such that 1 is the best possible value and corresponds to identical empirical distributions. The metric is computed for each column in $\tilde{\mathbf{T}}_{train}$ and its corresponding column in $\mathbf{T}_{train}$, and then averaged over all the columns.

Chi-Squared (`CST`) : this metric compares the distribution of categorical columns using a Chi-Squared goodness of fit test. When applied to a given column of $\tilde{\mathbf{T}}_{train}$ and its corresponding column in $\mathbf{T}_{train}$, the p-value of this test can be interpreted as the probability that the two columns follow the same distribution (the maximum value is 1). As `KST`, the metric is computed for each column and then averaged over all the columns.

Continuous KL (`CKL`) : this metric computes the KL divergence between the joint probability distribution of two numerical columns in $\tilde{\mathbf{T}}_{train}$ and the joint probability distribution of the two corresponding columns in $\mathbf{T}_{train}$. It does this for all possible pairs of columns, takes the average value and then outputs the normalized value $1/(1 + \overline{KL})$. The minimal value is thus zero when $\overline{KL} = \infty$ and the maximal value is 1 when $\overline{KL} = 0$.

Discrete KL (`DKL`) : this metric is exactly the same as `CKL` but applied to categorical columns.

Correlation (`C`) : the goal of this metric is to measure the difference between the correlation matrix $\tilde{M}$ of the numerical columns in $\tilde{\mathbf{T}}_{train}$ and the correlation matrix M of the same columns in $\mathbf{T}_{train}$. First, we compute the Frobenius norm of the matrix difference : $d = \|M - \tilde{M}\|_F$. Then, in order to have a metric between 0 and 1, we normalize the output as $1 - (d/d_{max})$ where d_{max} is the maximum value for d given M.

Theil's U (`TU`) : this metric is the adaptation of the correlation metric `C` for categorical columns. To measure the association between a two categorical columns X and Y, we use Theil's U. It is also known as the uncertainty coefficient $U(X|Y)$ and is defined as :

$$U(X|Y) = \frac{H(X) - H(X|Y)}{H(X)} \tag{5.1}$$

where H denotes the entropy. We see that if X is completely defined by Y (maximal association), $H(X|Y) = 0$ and thus $U(X|Y) = 1$. On the contrary, if X is independent from Y, $H(X|Y) = H(X)$ and $U(X|Y) = 0$. By applying Theil's U, we can construct the association matrices $\tilde{M}$ and M and then compute the normalized output of the metric exactly as for the numerical case.

Correlation ratio (`CR`) : this metric is similar to the correlation metric `C` and to the Theil's U metric `TU`. It measures the difference in mixed associations of numerical and categorical columns between $\tilde{\mathbf{T}}_{train}$ and $\mathbf{T}_{train}$. To compute the association between a numerical column X and a categorical column Y, we use the correlation ration η.

Lets denote each observation of X as x_{yi} where y indicates the category of Y it belongs to, and i its index in that category. Let n_y be the number of observations x in category y, $\overline{x}_y$ the mean of category y and $\overline{x}$ the global mean of column X. The correlation ratio is given by:

$$\eta = \sqrt{\frac{\sum_y n_y(\overline{x}_y - \overline{x})^2}{\sum_{y,i}(x_{y,i} - \overline{x})^2}} \tag{5.2}$$

If there is no association between X and Y, the mean $\overline{x}_y$ in each category y will be equal to the global mean and we see that $\eta = 0$. An extreme case where $\eta = 1$ could be when observations within categories are the same, but different across all categories. We can now compute η for all possible combination of a categorical and a numerical column, and thus construct $\tilde{M}$ and M. The final output of the metric is computed exactly as with `C` and `TU`.

5.3.2 Likelihood metrics

Gaussian mixture log likelihood (`GMLL`) : this metric fits a multi-dimensional Gaussian mixture model to all numerical columns of $\mathbf{T}_{train}$ and computes the likelihood of $\tilde{\mathbf{T}}_{train}$ given this model. It then returns the logarithm of this likelihood.

Bayesian network log likelihood (`BNLL`) : this metric is the counterpart of `GMLL` for categorical columns. It fits a Bayesian network model to all categorical columns of $\mathbf{T}_{train}$ and computes the likelihood of $\tilde{\mathbf{T}}_{train}$ given this model. It then returns the logarithm of this likelihood.

5.3.3 Detection metrics

Logistic detection (`LRD`) : to compute this metric, we assign labels "real" to rows of $\mathbf{T}_{train}$ and labels "fake" to rows of $\tilde{\mathbf{T}}_{train}$. A logistic regression model is then trained on $\mathbf{T}_{train} \oplus \tilde{\mathbf{T}}_{train}$ to predict the label of a given row. The best possible scenario is when the model cannot distinguish between "real" and "fake" rows and thus has 0 zero accuracy. The metric outputs 1 minus the accuracy of the model such that 1 is the best value.

Support vector detection (`SVCD`) : this metric does exactly the same as `LDR` but trains a support vector machine classifier instead of a logistic regression model.

5.3.4 Machine learning efficacy metrics

Machine learning efficacy (`ME`) : this metric is the aggregate of many other metrics. Indeed, we train several classifiers or regressors on the real data $\mathbf{T}_{train}$ depending on the nature of the target attribute. We then evaluate the models on $\mathbf{T}_{test}$ with adequate metrics and denote s_{real} the average score across all evaluated metrics and models. We repeat the operation but train the models on $\tilde{\mathbf{T}}_{train}$ instead of $\mathbf{T}_{train}$ and denote the average result s_{fake}. The machine learning efficacy metric is then s_{fake}/s_{real} and represents to which extent we can substitute real data with synthetic data for machine learning tasks. If `ME` is equal to 1, it means that we obtain equally good performance when training the models on synthetic data as when training on real data.

For datasets with a categorical target attribute (Adult, Heart, Hcv), we use the following classifiers from the `sklearn`[4] library : `LogisticRegression`, `KNeighborsClassifier`, `RandomForestClassifier` and `MLPClassifier`. Furthermore, we evaluate them with the following metrics : `Accuracy`, `F1-score`, `ROC-AUC` (area under the receiver operating characteristic curve) and `PR-AUC` (area under the precision-recall curve). `ROC-AUC` and `PR-AUC` give a more general idea of the classification performance as they do not depend on a given classification threshold. `F1-score` and `PR-AUC` are more suited than `Accuracy` and `ROC-AUC` in imbalance classification problems (the last two may be over optimistic in such a context).

For the only dataset with a numerical target attribute (Student), we use the following regressors: `Ridge`, `KNeighborsRegressor`, `RandomForestRegressor` and `MLPRegressor`. Furthermore, We evaluate their performance with mean square error `MSE` and mean absolute error `MAE`.

For all models, we simply use the default parameters of `sklearn` as we are more interested in comparing performance when training on synthetic data versus training on real data than in overall raw performance.

5.3.5 Data augmentation

The process of evaluating the impact of data augmentation is very similar to evaluating machine learning efficacy in the sense that we use the same classifiers/regressors models to predict the target attribute and the same metrics to evaluate them. However, in addition to evaluating models trained on $\tilde{\mathbf{T}}_{train}$ and models trained on $\mathbf{T}_{train}$ (which will act as reference points), we also and especially evaluate models trained on augmented data $\mathbf{T}_{aug} = \mathbf{T}_{train} \oplus \tilde{\mathbf{T}}_{train}$. Besides, we do not look at the ratio of aggregated scores s_{aug}/s_{real} and we compute the average score only over the models to keep a sense of which metric is impacted (or not) by the augmentation of data. Moreover, recall that $\tilde{\mathbf{T}}_{train}$ and $\mathbf{T}_{train}$ can have different sizes in $\mathbf{T}_{aug}$. Therefore, we test several scenarios as represented in Figure 5.2. In (a), we have $|\tilde{\mathbf{T}}_{train}| = 0.5 \cdot |\mathbf{T}_{train}|$; in (b) we have the standard case $|\tilde{\mathbf{T}}_{train}| = |\mathbf{T}_{train}|$; in (c) we have $|\tilde{\mathbf{T}}_{train}| = 2 \cdot |\mathbf{T}_{train}|$; finally, in (d), we sample $\tilde{\mathbf{T}}_{train}$ such that the target attribute is balanced in $\mathbf{T}_{aug}$ (only applicable when the target attribute is categorical).

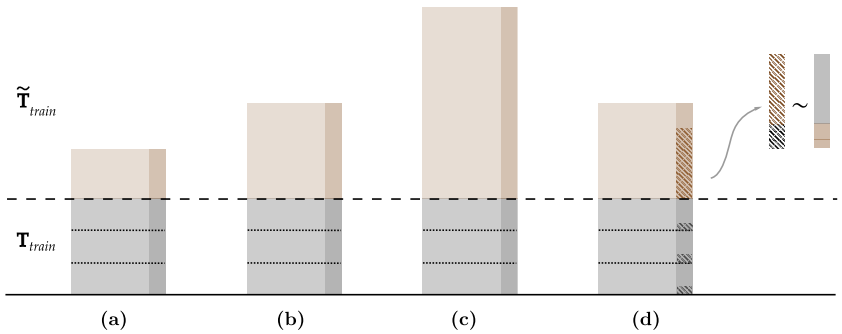


Figure 5.2: Various scenarios for data augmentation.

[4]https://scikit-learn.org/stable/index.html

Chapter 6

Results

In this chapter, we evaluate both CTGAN and our enhanced CTGAN according to the procedure described in Chapter 5. We first report the results strictly related to data synthesis and compare the quality of the synthetic data generated by the two models. Then, we analyze the impact of augmenting data with the enhanced model.

6.1 Data synthesis

Table 6.1 contains the scores of CTGAN and our enhanced CTGAN for all evaluated metrics and for each dataset. The colour expresses whether our model improves on CTGAN (greenish colour) or not (reddish colour). Note that this is a visual aid only and is thus not based on any precisely defined quantitative scale. From the looks of the table, it is clear that our model is indeed an enhanced version of CTGAN. It outperforms the latter on all datasets and for all metrics except one, for which it achieves similar results: the `CST` metric. Recall that this metric is a Chi-squared goodness of fit test performed independently on each categorical column of the synthetic data and its corresponding column in the real data. We thus observe that CTGAN is already very good at modelling the distributions of individual categorical columns. However, it may struggle to synthesized realistic data from numerical columns with complex distributions. Such columns are better handled by our enhanced CTGAN, as indicated by the particularly big increases in the score of `KST` (and the scores of `CKL` and `GMLL`). We illustrate this on Figure 6.1.

			Adult		Heart		Hcv		Student	
	Num	Cat	CTGAN	Ours	CTGAN	Ours	CTGAN	Ours	CTGAN	Ours
BNLL		×	-15.315	-12.463	-8.482	-6.866	-1.899	-1.107	-24.922	-22.667
GMLL	×		-2.1E+10	-2.8E+09	-42.094	-14.508	-98.651	-70.473	-1.3E+03	-116.077
LRD	×	×	0.543	0.638	0.360	0.863	0.232	0.647	0.383	0.501
SVD	×	×	0.443	0.901	0.178	0.768	0.134	0.630	0.277	0.469
CST		×	0.985	0.981	0.964	0.959	0.938	0.948	0.962	0.952
KST	×		0.637	0.896	0.705	0.903	0.685	0.913	0.712	0.817
CKL	×		0.612	0.791	0.337	0.590	0.445	0.747	0.418	0.681
DKL		×	0.755	0.807	0.908	0.951	0.790	0.961	0.941	0.943
C	×		0.937	0.956	0.847	0.895	0.881	0.906	0.932	0.947
TU		×	0.879	0.896	0.932	0.957	0.992	0.995	0.974	0.976
CR	×	×	0.918	0.943	0.788	0.854	0.852	0.883	0.902	0.923
ME	×	×	0.546	0.892	0.599	0.952	0.606	0.770	0.399	0.715

Table 6.1: Evaluation of the synthetic data. Comparison between CTGAN and our enhanced model.

In general, we observe that our enhanced CTGAN better identifies and recovers the various modes of numerical columns than CTGAN. This is particularly clear in (b), where the data is distributed globally in a Gaussian-like fashion, but actually consists in many local modes. We believe that this increased ability to recognize modes is due in part to our modification of the VGM model used to perform *mode specific normalization*. Moreover, we can see in (a) that the enhanced CTGAN is able to generate data from minor secondary modes, also when such modes are separated from the bulk of the distribution, as in (c) and (d). This can be explained by two factors. First, the extended conditional vector, together with the *training by sampling* method, allows for more training opportunities on data that belongs to minor modes. Secondly, the log transform reduces the distance to separated modes in skewed and/or long tailed distributions.

Note that in (c), CTGAN generates unrealistic negative values when trying to recreate the mode near zero, whereas in (d), it completely drops the local mode at zero. Yet, columns with a local mode at zero are very frequent in tabular data where zero is usually the default value of a numerical attribute, or the "non-value". For instance, in (d), the attribute represents the final score out of 20 obtained by students on a given test. The zero mode probably contains one or two students that took the test and effectively obtained a zero, but it mostly contains students that did not take the test at all. It is thus important for a tabular data synthesizer to be able to capture such local modes.

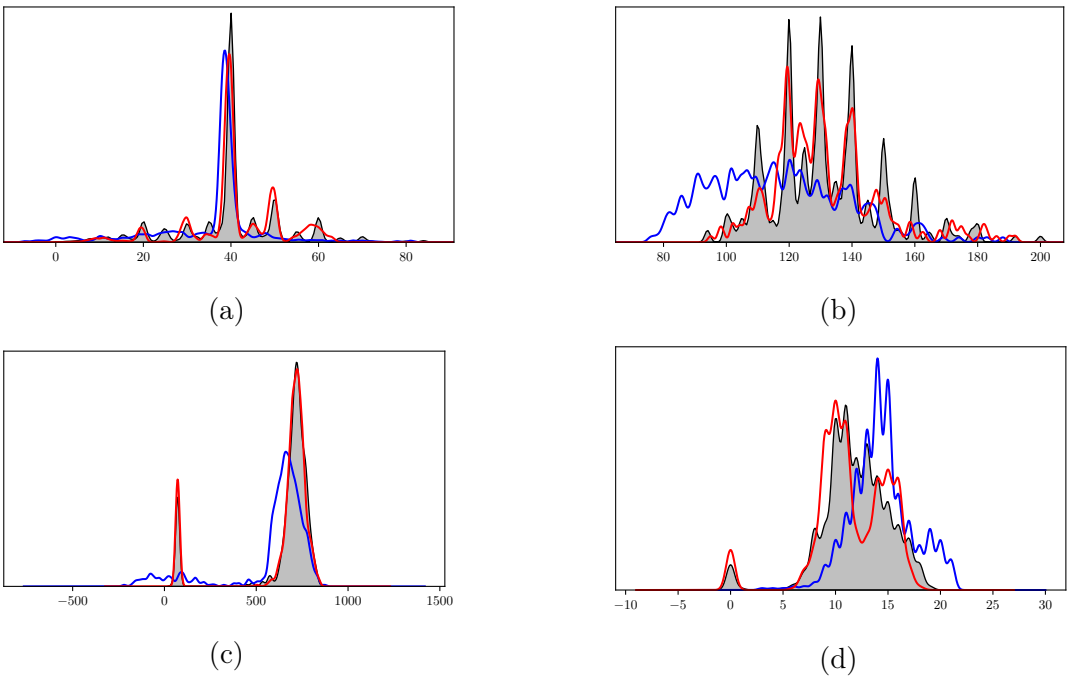


Figure 6.1: Distributions of numerical synthetic columns versus distributions of real columns. (a) column `hours-per-week` from the `Adult` dataset. (b) column `trestbps` from the `Heart` dataset. (c) column `PROT` from the `Hcv` dataset. (d) column `G3` from the `Student` dataset. The probability distributions of the columns are estimated through kernel density estimation. Black shaded curve: distribution of the original data. Blue curve: distribution of the synthetic data produced by CTGAN. Red curve: distribution of the synthetic data produced by our enhanced CTGAN.

In addition to synthesizing numerical columns of better quality, we find that our enhanced CTGAN can also capture the different relationships between the columns more efficiently than CTGAN. This is indicated by higher scores for correlation/association metrics (`C`,`TU`,`CR`) and likelihood metrics (and also `DKL` and `CKL`). We illustrate this for the `Adult` dataset on Figure 6.2 by representing the association matrices[1] of the original data, of the synthetic data produced by the enhanced CTGAN, and of the synthetic data produced by the original CTGAN.

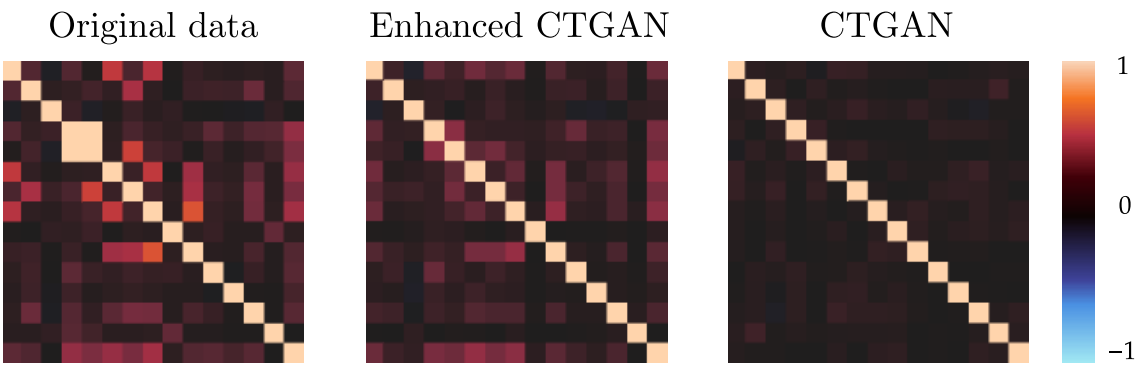


Figure 6.2: Comparison of the association matrices for the Adult dataset.

We see that CTGAN is in fact barely able to capture any association between the columns. This is true for this dataset, and we also obtained similar results on the other three datasets, but we do not claim that it is necessarily true for all possible datasets and in all experimental conditions. Anyway, in our case, we believe that the poor results of CTGAN are due to the use of one-hot vectors $\boldsymbol{\delta}$ and $\boldsymbol{\beta}$ to represent respectively categories and modes. Indeed, using one-hot encoding, CTGAN preprocesses the categorical columns independently of each other and independently of the numerical columns, leaving the task of capturing the associations between the columns entirely to the generator. However, in the `Adult` dataset, most associations are actually between categorical columns or between a categorical column and a numerical column, so CTGAN has difficulty capturing them effectively.

In contrast, in the enhanced CTGAN, we construct encoded representations $\boldsymbol{\delta}_e$ and $\boldsymbol{\beta}_e$ for categories and modes that take into account the relationships between the different columns and thus make the generator's task a little easier[2]. As a result, while certainly not identical, the association matrix of the synthetic data from the enhanced CTGAN is already much closer to the one of the original data.

[1]The value in the association matrix for two given columns is given by the correlation if both are numerical, Theils'U if both are categorical, and the correlation ratio if one is numerical and the other categorical.

[2]As often in machine learning, this is mostly an assumption made on the basis of empirical results. We cannot prove mathematically that our encoding scheme actually makes the optimization of the objective criterion an easier task.

6.1.1 Average scores

To get a more comprehensive view of the performance of the enhanced CTGAN, we average the results over the four datasets and express the percentage change compared to the original CTGAN. These global results are presented in table 6.2.

	Num	Cat	CTGAN	Ours	+ [%]
BNLL		×	-12.655	**-10.776**	14.85
GMLL	×		-5.4E+09	**-7.E+08**	87.14
LRD	×	×	0.379	**0.662**	74.57
SVD	×	×	0.258	**0.692**	168.41
CST		×	**0.963**	0.960	-0.24
KST	×		0.685	**0.882**	28.91
CKL	×		0.453	**0.702**	55.12
DKL		×	0.848	**0.915**	7.89
C	×		0.900	**0.926**	2.94
TU		×	0.944	**0.956**	1.24
CR	×	×	0.865	**0.901**	4.12
ME	×	×	0.538	**0.832**	54.85

Table 6.2: Average results of CTGAN and enhanced CTGAN over all datasets.

As we have already pointed out, the only negative change concerns the `CST` metric and is not very significant. For all other metrics, the change is positive. Furthermore, in line with our previous observations on numerical columns, we find that the changes for `KST`, `CKL` and `GMLL` are rather substantial (between 28% and 87%). We observe less substantial changes for their categorical counterparts `DKL` and `BNLL`.

Note, however, that we must be very careful when comparing these percentages as a change of x% for one metric may not be (and is generally not) qualitatively equivalent to a change of x% in another metric. In our interpretation, we must take into account the relative difficulty of increasing the score of a given metric and the impact of such an increase on the synthesized data. For instance, we see that we only have a small increase in the scores of `C`, `TU` and `CR`[3] but the effective change in the synthetic data is drastic, as demonstrated by the associations matrices represented on Figure 6.2. On the other hand, we have observed in our experiments that larger changes in the detection metrics do not necessarily lead to such drastic changes in the synthetic data.

Finally, we observe a 55% increase of the machine learning efficacy. This is a very significant improvement that has important practical implications. In particular, this means that the synthetic data generated by the enhanced CTGAN can be used as a more reliable substitute to real data for machine learning tasks than the synthetic data produced by the original CTGAN.

[3]We see that these metrics could actually be designed in a better way. Recall that they measure the difference between the association matrices of real and synthetic data; yet, in most tables, numerous columns have no or very little association with other columns. This means that a random synthesizer producing an association matrix with zeros everywhere off the diagonal will still likely obtain high scores for the three metrics.

6.1.2 Ablation study

To assess the impact of each component that we have added to CTGAN, we perform an ablation study. We divide these components in four groups : (1) extended conditional vector, log transform on skewed numerical attributes and recursive scheme for VGM model; (2) encoder and decoder; (3) *feature matching loss* in $\mathcal{L}_G$; (4) auxiliary predictor network and *prediction loss* in $\mathcal{L}_G$. We evaluate the enhanced CTGAN following the same procedure as before but we cut off the different groups one by one. We then express the percentage of change with respect to the case where the enhanced CTGAN model is trained with all its components. The results averaged over all datasets are reported in table 6.3. The individual results for each dataset are available in Appendix A.

	Num	Cat	w/o c+, Log, RVGM	w/o Encoder Decoder	w/o Feature matching	w/o Auxiliary Predictor
BNLL		×	-0.9	-14.08	-0.87	-1.29
GMLL	×		-8.57	-76.69	-53.45	-45.13
LRD	×	×	-6.32	23.99	-12.01	-2.86
SVD	×	×	-3.39	-5.55	-14.09	-9.04
CST		×	1.51	2.16	-0.48	-0.21
KST	×		-5.01	2.77	-3.29	-0.4
CKL	×		-2.61	-4.46	-4	2.04
DKL		×	-3.2	-2.54	-1.14	-0.19
C	×		-0.13	-2.81	-0.81	-0.62
TU		×	0.33	-1.19	-0.72	-0.66
CR	×	×	-0.16	-3.95	-1.07	-0.84
ME	×	×	5.6	-34.96	-15.22	-8.84

Table 6.3: Ablation study on the enhanced CTGAN.

w/o c+, Log, RVGM : we observe a negative impact on `GMLL` and `KST`. This seems intuitive as the components in this group mostly have the goal to improve the quality of synthetic numerical columns. We believe that the impact of these components is more visible when they are combined with others. This is suggested notably by the results for `KST` : cutting off each group one by one only reduces the scores of the metric by a maximum of 5%, whereas cutting off all groups together would result in a 29% decrease.
w/o Encoder-Decoder : removing the encoder improves slightly the scores of metrics that evaluate single column quality (`CST` and `KST`) but has a dramatic effect on all metrics that measure to some extent the interactions between columns. It is thus no surprise that this also has a significantly negative impact on `ME`.
w/o Feature matching : We observe negative impact for every metric. The decrease in `CST` and `KST` can be easily linked to the absence of the terms $\mathcal{L}_{mean-G}$ and $\mathcal{L}_{std-G}$ as they explicitly encourage individual synthetic columns to match their corresponding real column. On the other hand, the decrease in `LRD` and `SVCD` can be linked to the absence of the terms $\mathcal{L}_{mean-D}$ and $\mathcal{L}_{std-D}$ as they encourage features of synthetic data in D to resemble the ones of real data.
w/o Auxiliary predictor : the goal of the predictor being to encourage coherent synthetic rows with respect to their target attribute, we find negative impact on `ME`.

These results support our design choices for the enhanced CTGAN model.

6.2 Data augmentation

Recall that for data augmentation, we train the enhanced CTGAN on $\mathbf{T}_{train}$, sample $\tilde{\mathbf{T}}_{train}$ and then construct $\mathbf{T}_{aug} = \mathbf{T}_{train} \oplus \tilde{\mathbf{T}}_{train}$. Next, we train several classifiers (for `Adult`, `Heart`, `Hcv`) or regressors (for `Student`) on $\mathbf{T}_{aug}$ and evaluate their performance with the adequate metrics on $\mathbf{T}_{test}$.

We construct different tables $\mathbf{T}_{aug}$ according to the various data augmentation scenarios described in chapter 5. The average scores over the different classifiers/regressors are reported for each dataset in table 6.4. Detailed results for each classifier/regressor are available in Appendix B.

Adult	Acc	F1	ROC AUC	PR AUC
R	**0.84**	0.619	0.741	**0.734**
B	0.802	**0.637**	**0.78**	0.686
S	0.808	0.515	0.682	0.615
R+S/2	0.837	0.614	0.738	0.726
R+S	0.836	0.61	0.736	0.716
R+S*2	0.832	0.599	0.729	0.702
Heart	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	**0.842**	**0.855**	**0.84**	0.913
B	0.834	0.845	0.833	**0.914**
S	0.796	0.809	0.795	0.883
R+S/2	0.829	0.843	0.827	0.909
R+S	0.828	0.842	0.827	0.906
R+S*2	0.832	0.845	0.83	0.906
Hcv	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	**0.929**	**0.445**	**0.968**	**0.66**
B	0.491	0.306	0.876	0.53
S	0.898	0.244	0.771	0.395
R+S/2	0.923	0.417	0.957	0.628
R+S	0.918	0.398	0.951	0.621
R+S*2	0.915	0.37	0.944	0.611
Student	**MSE**	**MAE**		
R	**2.046**	**7.696**		
S	2.481	11.062		
R+S/2	2.079	7.997		
R+S	2.103	8.176		
R+S*2	2.16	8.628		

Table 6.4: Data augmentation results for different $\mathbf{T}_{aug}$. **R** : real data $\mathbf{T}_{train}$ (Reference). **S** : synthetic data $\tilde{\mathbf{T}}_{train}$ (Reference). **B**: real data augmented to have balanced class attributes. **R+S/2**: real data augmented with half as much synthetic data. **R+S**: real data augmented with the same quantity of synthetic data. **R+S*2**: real data augmented with twice as much synthetic data.

We find that the best results are almost always obtained by training the models on real data only. This is a bit disappointing but also not so surprising. Indeed, we assume that the different models reach their near maximum capacity when trained on real data and therefore, since the synthetic data is not a perfect substitute (ME is not 1), adding it to the real training data amounts to adding noise. This noise can be useful for regularization, but in general, the more it is added, the more it reduces the performance of the models. This is clearly observable for the Adult, Hcv and Student datasets. This decrease in performance is less significant in the case of the Heart dataset for which the enhanced CTGAN can synthesized data of remarkable machine learning efficacy (ME=0.95).

However, we see that the F1 score and the ROC AUC score are slightly increased for the Adult dataset when training the models on a balanced $\mathbf{T}_{aug}$. This suggests that the enhanced CTGAN may be efficient at oversampling minority classes in the context of imbalanced classification. We investigate this further on the two imbalanced datasets Adult and Hcv. The Adult dataset has a binary target attribute and is lightly imbalanced with a majority/minority class ratio of 3:1. The Hcv dataset has a multi-class target attribute and is strongly imbalanced with about 85% of the samples in one majority class and the remaining 15% distributed among the four other minority classes. Our goal is thus to construct $\mathbf{T}_{aug}$ such that the minority classes are oversampled (not necessarily in a ratio 1:1 with respect to the majority class).

To generate more realistic samples of each class, we divide $\mathbf{T}_{train}$ into one $\mathbf{T}_y$ per class y where $\mathbf{T}_y$ contains only the rows of $\mathbf{T}_{train}$ that belong to class y. We then train one enhanced CTGAN model on each $\mathbf{T}_y$. This allows us to construct $\mathbf{T}_{aug}$ with any class ratio, and more importantly, with minority rows of much higher quality than in the previous experimental setting. In these new conditions, we manage to obtain better results for models trained on the augmented data compared to models trained on the original data. We report such results in table 6.4. Note that we do not improve the original results by a large amount, but this is nevertheless encouraging and an interesting direction to explore in some future research.

Adult	Acc	F1	ROC AUC	PR AUC
R	0.841	0.627	0.739	0.732
B1	0.838	**0.638**	**0.761**	0.741
B2	**0.843**	0.635	0.755	**0.744**
Hcv	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.927	0.458	0.964	0.661
B1	**0.931**	0.527	0.971	0.652
B2	0.928	**0.559**	**0.976**	**0.668**

Table 6.5: Data augmentation on the imbalanced datasets Adult and Hcv. For the Adult dataset, **R** contains 177 rows in the minority class and 573 rows in the majority class. For $\mathbf{T}_{aug}$ =**B1**, we add 200 rows to the minority class and 400 to the majority class. For $\mathbf{T}_{aug}$ =**B2**, we add 400 rows to the minority class and 800 to the majority class. For the Hcv dataset, **R** contains a total of 47 rows in the four minority classes and 394 rows in the majority class. For $\mathbf{T}_{aug}$ =**B1**, we add 100 rows to each minority class and 0 to the majority class. For $\mathbf{T}_{aug}$ =**B2**, we add 100 rows to the each minority class and 250 to the majority class.

6.3 Discussion and perspectives

We chose to work with four datasets that have a good mix of categorical and numerical columns. Yet, because of the wide variety of data types that a given column can contain and of the infinitely many ways column can interact together, those datasets are probably not very representative of the entire space of tables. The results presented in this chapter are therefore not generalizable to every single tabular dataset, even if there are good reasons to believe that the enhanced CTGAN model will outperform the original CTGAN on the vast majority of them. To properly validate the better performance of the enhanced CTGAN, a broader study should be conducted on much more datasets. It would also be valuable to identify datasets that could actually benefit from data augmentation.

Besides, in most discussed papers, the proposed tabular data synthesizer is trained on datasets with a much bigger number of rows. Due to computational limitations, we were not able to train our enhanced CTGAN on such big datasets and we are thus eager to know how our model scales up with the size of data. In particular, we implemented three auxiliary networks that need to be trained in addition to the WGAN's generator and discriminator and which may cause scaling issues.

On the other hand, these three additional networks also imply that there is a lot of room for improvement, as their respective architectures are very basic. Indeed, the decoder and the predictor are simple MLPs with straight forward loss functions that can be tweaked in many ways. Moreover, the encoder is based on one of the original versions of Word2Vec from 2013 which has been much improved since then. So it would naturally be interesting to apply the latest embedding technology to our encoder network in order to achieve even better and more meaningful representations for categories and modes.

Note that encoded representations always make some kind of compromise between how much information they contain and the precision with which they convey it. We could see this in the ablation study : when we removed the encoder-decoder structure, the scores of all metrics related to the global quality of the table decreased as expected, but the metrics related to the quality of individual columns actually increased. This suggests that our encoder gives up some precision in the representation of columns taken individually but conveys significant information on how columns relate to each other.

In addition to using elaborate encodings for categories and modes as a way to better capture relationships between columns, we could also use convolutional architectures for the generator and the discriminator instead of standard fully connected MLPs. This is notably the design choice made by the authors of the CTAB-GAN paper. Recall that CTAB-GAN actually outperforms CTGAN and all other state-of-the-art tabular data synthesizers but still uses one-hot vectors exactly as CTGAN to represent the categories and modes. Because the CTAB-GAN paper is not yet published and the code was not available, we could not determine exactly what architecture was used and so we referred instead to the architecture of CTGAN as a basis for our enhanced model. However, if the authors publish their code, it would be very interesting to use their exact architecture, but with our additional encoder-decoder structure.

Conclusion

The main goal of this work was to build a model capable of generating high quality synthetic tabular data. The motivation for synthetic data comes from the fact that qualitative tabular data is not always easy to obtain or access. Hence, if sufficiently realistic, synthetic data can be used as a substitute in any situation where real data is needed but not available. Furthermore, synthetic data can be added to existing data to hopefully increase the performance of machine learning models. This is known as data augmentation and we built our tabular data synthesizer with this specific application in mind.

We first explained that in a given table, rows can be seen as samples of the joint distribution of the columns and that this distribution is typically highly multi-dimensional and multi-modal. The task of our tabular data synthesizer is thus to model the joint distribution of the columns, then sample from it to produce synthetic rows. Good candidates for modelling such a complex distribution are GANs. However, despite being theoretically sound, GANs are hard to train in practice and given the particular properties of tabular data, learning a GAN-based tabular data synthesizers presents its own set of challenges. These include dealing simultaneously with both categorical and numerical columns, learning imbalanced categorical columns, and capturing all modes of numerical columns, thus avoiding mode collapse.

We then briefly discussed existing approaches that address these challenges with varying degrees of success and chose CTGAN as the basis for our model. CTGAN uses a conditional Wassertein GAN framework : the use of the wassertein metric as an optimization criterion stabilizes the training process while the conditional nature of the framework, together with *training-by-sampling*, allows to alleviate the issue of imbalanced categorical columns. Moreover, CTGAN introduces *mode-specific normalization* to deal with complex multi-modal numerical columns. It identifies the different modes with a VGM model then represents a given numerical value as a combination of a normalized scalar and a one-hot vector indicating the mode with respect to which the value has been normalized. CTGAN also uses one-hot encoding to deal with categorical columns.

We noted that one-hot vectors do not give any information on how categories and/or modes relate to one another and we showed that, by treating each column independently in this way, CTGAN struggles to capture associations between columns. We have thus proposed a novel encoding scheme that allows to capture relationships between categories and/or modes directly in their representations. Besides, our model makes *mode-specific normalization* more flexible, extends the conditional vector and uses a log transform on skewed numerical columns. These additions allow to better model numerical columns, notably through a better ability to capture local and secondary modes. Finally, our enhanced version of CTGAN also uses *feature matching* and an auxiliary *predictor* to enforce the generator to produce qualitative and coherent rows.

Our enhanced version of CTGAN clearly outperforms the original CTGAN on the quality of the synthesized data. We observe significant improvements for the metrics related to numerical columns KST (+29%) and CKL (+55%), but also for the metrics related to categorical columns BNLL (+15%) and DKL (+8%). The few percentage improvements for the three metrics C, TU, and CR do not seem as important in comparison, but we showed that they actually have a huge impact on the synthesized data. Indeed, these metrics measure to which extent a tabular data synthesizer can recover the associations between the different columns of a table, and having this ability seems absolutely crucial to produce qualitative synthetic data. The improvements for C, TU, and CR are therefore directly correlated to the 55% improvement in machine learning efficacy. This can be observed when the encoder is disabled: the scores of C, TU, and CR pretty much drop to the level they are at for the original CTGAN and the machine learning efficacy drops by a striking 35%. This also demonstrates the importance and positive impact of our encoder.

However, despite all these significant improvements, the produced synthetic data does not lead to better performance when used for data augmentation. Indeed, the tested classifiers and regressors actually perform worse when trained on augmented data than when trained on the original data. In our case, we assume that this is because these models almost reach their maximum capacity when trained on real data, and since the synthetic data is not yet of good enough quality, adding it to real data is like adding noise. To get better results, we propose to learn a synthesizer on each class separately. This leads to encouraging results that open up some perspectives for data augmentation in the context of imbalanced classification. Nonetheless, the improvements remain marginal.

If fact, even under the assumption that a given model has not reached maximum capacity when trained on the real data (which is not easy to predict beforehand nor to verify), it seems that synthetic data must have a ME score very close to 1 to efficiently augment the real data and have a significant impact on the performance of the model. It is likely impossible to reach ME = 1, but we believe that we can get very close. As we have discussed, ME is strongly related to the metrics C, TU, and CR and thus one way to improve ME is to better capture the associations between the columns. This, in turn, can be achieved with better encodings of categories and modes. Finally, we find that the representation of data itself may well be the key to a successful data augmentation.

In further work, we would thus like to explore more sophisticated embeddings and analyze their impact on machine learning efficacy and on the ability of the tabular data synthesizer to effectively capture the relationships between the columns of a given table. Moreover, as convolutional networks seem to be more appropriate than standard MLPs for capturing the correlation between features, we would like to combine a convolutional architecture with our encoder-decoder structure.

Appendix A

Ablation study - detailed results

WITHOUT (EXTENDED + LOG + RVGM)					
Student	CTGAN	Ours	Adult	CTGAN	Ours
BNLL	-24.894	-22.636	BNLL	-15.343	-12.439
GMLL	-124.985	-140.670	GMLL	-8.948E+09	-3.02E+09
LD	0.362	0.471	LD	0.479	0.666
SVCD	0.257	0.484	SVCD	0.467	0.892
CST	0.960	0.962	CST	0.983	0.992
KST	0.707	0.834	KST	0.632	0.756
CKL	0.450	0.744	CKL	0.610	0.770
DKL	0.938	0.943	DKL	0.757	0.817
C	0.932	0.949	C	0.936	0.942
TU	0.974	0.977	TU	0.879	0.903
CR	0.902	0.926	CR	0.917	0.925
ME	0.383	0.840	ME	0.541	0.884
Hcv	CTGAN	Ours	Heart	CTGAN	Ours
BNLL	-1.891	-1.828	BNLL	-8.503	-6.590
GMLL	-95.991	-75.737	GMLL	-4523.900	-43.681
LD	0.304	0.535	LD	0.321	0.820
SVCD	0.164	0.487	SVCD	0.160	0.813
CST	0.936	0.973	CST	0.960	0.973
KST	0.711	0.859	KST	0.701	0.912
CKL	0.470	0.651	CKL	0.338	0.573
DKL	0.788	0.838	DKL	0.906	0.949
C	0.880	0.907	C	0.852	0.901
TU	0.988	0.994	TU	0.932	0.963
CR	0.851	0.885	CR	0.795	0.862
ME	0.610	0.842	ME	0.611	0.961

Table A.1: Scores of CTGAN and enhanced CTGAN for each dataset, without the extended conditional vector, the log transform on skewed numerical attributes and the recursive Gaussian mixture model.

WITHOUT ENCODING-DECODING					
Student	CTGAN	Ours	Adult	CTGAN	Ours
BNLL	-24.955	-25.027	BNLL	-15.262	-15.246
GMLL	-11.459	-3.692	GMLL	-1.037E+10	-1.19E+10
LD	0.366	0.899	LD	0.504	0.897
SVCD	0.261	0.585	SVCD	0.426	0.945
CST	0.961	0.983	CST	0.986	0.989
KST	0.698	0.884	KST	0.630	0.909
CKL	0.430	0.666	CKL	0.614	0.767
DKL	0.938	0.957	DKL	0.758	0.775
C	0.927	0.928	C	0.939	0.938
TU	0.974	0.974	TU	0.879	0.879
CR	0.894	0.896	CR	0.920	0.919
ME	0.387	0.679	ME	0.530	0.528
Hcv	CTGAN	Ours	Heart	CTGAN	Ours
BNLL	-1.885	-1.451	BNLL	-8.501	-8.443
GMLL	-100.920	-88.963	GMLL	-22.094	-14.824
LD	0.260	0.737	LD	0.319	0.954
SVCD	0.162	0.609	SVCD	0.167	0.483
CST	0.934	0.980	CST	0.959	0.973
KST	0.701	0.921	KST	0.695	0.916
CKL	0.456	0.702	CKL	0.347	0.554
DKL	0.793	0.925	DKL	0.903	0.914
C	0.882	0.883	C	0.853	0.853
TU	0.989	0.994	TU	0.932	0.932
CR	0.853	0.854	CR	0.795	0.796
ME	0.604	0.618	ME	0.649	0.643

Table A.2: Scores of CTGAN and enhanced CTGAN for each dataset, without the encoder-decoder structure.

WITHOUT FEATURE MATCHING					
Student	CTGAN	Ours	Adult	CTGAN	Ours
BNLL	-24.813	-22.568	BNLL	-15.374	-12.679
GMLL	-13.453	-144.493	GMLL	-2.611E+10	-5.935E+09
LD	0.368	0.459	LD	0.440	0.589
SVCD	0.264	0.421	SVCD	0.382	0.774
CST	0.960	0.952	CST	0.985	0.960
KST	0.733	0.791	KST	0.614	0.879
CKL	0.499	0.663	CKL	0.591	0.764
DKL	0.935	0.942	DKL	0.759	0.800
C	0.929	0.949	C	0.937	0.943
TU	0.974	0.976	TU	0.879	0.891
CR	0.897	0.927	CR	0.918	0.926
ME	0.402	0.705	ME	0.535	0.597
Hcv	CTGAN	Ours	Heart	CTGAN	Ours
BNLL	-1.899	-0.893	BNLL	-8.538	-7.342
GMLL	-100.586	-360.466	GMLL	-1020.360	-527.370
LD	0.272	0.577	LD	0.378	0.741
SVCD	0.157	0.628	SVCD	0.180	0.602
CST	0.943	0.953	CST	0.963	0.958
KST	0.703	0.872	KST	0.708	0.875
CKL	0.446	0.735	CKL	0.333	0.539
DKL	0.787	0.939	DKL	0.903	0.939
C	0.880	0.891	C	0.847	0.891
TU	0.992	0.990	TU	0.931	0.941
CR	0.851	0.865	CR	0.788	0.848
ME	0.591	0.636	ME	0.642	0.951

Table A.3: Scores of CTGAN and enhanced CTGAN for each dataset, without adding the feature matching loss term in the loss of G.

WITHOUT PREDICTOR					
Student	CTGAN	Ours	Adult	CTGAN	Ours
BNLL	-25.107	-22.714	BNLL	-15.332	-12.542
GMLL	-1179.890	-222.475	GMLL	-1.453E+10	-5.035E+09
LD	0.370	0.496	LD	0.552	0.629
SVCD	0.268	0.467	SVCD	0.465	0.799
CST	0.959	0.953	CST	0.983	0.971
KST	0.736	0.815	KST	0.640	0.890
CKL	0.481	0.702	CKL	0.608	0.804
DKL	0.935	0.944	DKL	0.758	0.811
C	0.931	0.949	C	0.938	0.948
TU	0.974	0.976	TU	0.879	0.892
CR	0.899	0.926	CR	0.919	0.932
ME	0.429	0.751	ME	0.547	0.656
Hcv	CTGAN	Ours	Heart	CTGAN	Ours
BNLL	-1.886	-0.991	BNLL	-8.558	-7.418
GMLL	-116.033	-221.309	GMLL	-348.565	-14.365
LD	0.259	0.619	LD	0.408	0.832
SVCD	0.143	0.638	SVCD	0.199	0.634
CST	0.935	0.949	CST	0.959	0.959
KST	0.702	0.910	KST	0.736	0.901
CKL	0.447	0.763	CKL	0.336	0.598
DKL	0.791	0.960	DKL	0.908	0.939
C	0.882	0.900	C	0.850	0.885
TU	0.991	0.994	TU	0.932	0.938
CR	0.853	0.875	CR	0.792	0.840
ME	0.611	0.731	ME	0.619	0.921

Table A.4: Scores of CTGAN and enhanced CTGAN for each dataset, without the auxiliary predictor.

Appendix B

Data augmentation - detailed results

Adult				
R	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.844	0.628	0.746	0.759
RF	0.836	0.609	0.736	0.714
MLP	0.836	0.627	0.750	0.726
SVR	0.845	0.613	0.732	0.738
B	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.799	0.653	0.800	0.712
RF	0.803	0.628	0.770	0.669
MLP	0.798	0.616	0.760	0.677
SVR	0.807	0.650	0.791	0.686
S	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.813	0.543	0.697	0.635
RF	0.815	0.504	0.674	0.624
MLP	0.793	0.537	0.696	0.580
SVR	0.812	0.475	0.659	0.620
R+S/2	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.841	0.624	0.744	0.744
RF	0.833	0.603	0.732	0.713
MLP	0.828	0.611	0.740	0.707
SVR	0.847	0.618	0.736	0.740
R+S	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.839	0.618	0.740	0.730
RF	0.834	0.606	0.733	0.709
MLP	0.824	0.600	0.734	0.687
SVR	0.847	0.617	0.735	0.739
R+S*2	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.837	0.610	0.735	0.719
RF	0.832	0.598	0.728	0.703
MLP	0.815	0.580	0.722	0.664
SVR	0.844	0.607	0.729	0.724

Table B.1: Scores for each classifier on the `Adult` dataset in various augmentation scenarios. **R** is a logistic regression model; **RF** is a random forest classifier; **MLP** is a multi-layer perceptron classifier; **SVR** is a support vector machine classifier.

Heart				
R	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.838	0.852	0.835	0.923
RF	0.831	0.844	0.829	0.913
MLP	0.847	0.860	0.845	0.915
SVR	0.851	0.865	0.849	0.902
B	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.840	0.852	0.838	0.923
RF	0.821	0.831	0.821	0.912
MLP	0.832	0.842	0.832	0.917
SVR	0.843	0.853	0.844	0.905
S	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.809	0.823	0.808	0.898
RF	0.789	0.802	0.788	0.876
MLP	0.791	0.804	0.790	0.875
SVR	0.795	0.808	0.794	0.882
R+S/2	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.838	0.852	0.834	0.922
RF	0.823	0.836	0.822	0.905
MLP	0.821	0.834	0.819	0.904
SVR	0.835	0.850	0.833	0.903
R+S	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.838	0.851	0.836	0.918
RF	0.822	0.836	0.820	0.905
MLP	0.818	0.832	0.817	0.899
SVR	0.835	0.848	0.834	0.901
R+S*2	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.843	0.856	0.840	0.921
RF	0.829	0.843	0.828	0.905
MLP	0.818	0.831	0.818	0.896
SVR	0.837	0.850	0.835	0.903

Table B.2: Scores for each classifier on the `Heart` dataset in various augmentation scenarios. **R** is a logistic regression model; **RF** is a random forest classifier; **MLP** is a multi-layer perceptron classifier; **SVR** is a support vector machine classifier.

Hcv				
R	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.932	0.482	0.975	0.659
RF	0.935	0.464	0.985	0.725
MLP	0.939	0.493	0.968	0.665
SVR	0.910	0.341	0.943	0.590
B	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.414	0.267	0.892	0.576
RF	0.539	0.371	0.937	0.599
MLP	0.579	0.405	0.840	0.487
SVR	0.431	0.182	0.833	0.457
S	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.900	0.281	0.801	0.428
RF	0.897	0.223	0.852	0.435
MLP	0.903	0.279	0.700	0.374
SVR	0.893	0.191	0.732	0.343
R+S/2	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.922	0.439	0.968	0.634
RF	0.933	0.454	0.979	0.705
MLP	0.932	0.470	0.944	0.610
SVR	0.907	0.305	0.936	0.565
R+S	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.914	0.420	0.963	0.634
RF	0.929	0.445	0.979	0.695
MLP	0.927	0.460	0.941	0.613
SVR	0.902	0.270	0.922	0.544
R+S*2	**Acc**	**F1**	**ROC AUC**	**PR AUC**
R	0.910	0.358	0.962	0.618
RF	0.926	0.433	0.975	0.686
MLP	0.926	0.454	0.922	0.608
SVR	0.898	0.235	0.917	0.534

Table B.3: Scores for each classifier on the `Hcv` dataset in various augmentation scenarios. **R** is a logistic regression model; **RF** is a random forest classifier; **MLP** is a multi-layer perceptron classifier; **SVR** is a support vector machine classifier.

Student		
R	**MSE**	**MAE**
R	2.149	8.069
RF	1.966	7.215
MLP	2.118	7.883
SVR	1.950	7.617
S	**MSE**	**MAE**
R	2.515	11.226
RF	2.437	10.886
MLP	2.521	11.231
SVR	2.452	10.908
R+S/2	**MSE**	**MAE**
R	2.139	8.131
RF	2.015	7.621
MLP	2.160	8.292
SVR	2.002	7.942
R+S	**MSE**	**MAE**
R	2.114	8.075
RF	2.055	7.878
MLP	2.186	8.536
SVR	2.055	8.218
R+S*2	**MSE**	**MAE**
R	2.117	8.250
RF	2.112	8.285
MLP	2.287	9.334
SVR	2.125	8.643

Table B.4: Scores for each regressor on the `Student` dataset in various augmentation scenarios. **R** is a Ridge regression model; **RF** is a random forest regressor; **MLP** is a multi-layer perceptron regressor; **SVR** is a support vector machine regressor.

Bibliography

[1] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling Tabular data using Conditional GAN. *arXiv:1907.00503 [cs, stat]*, October 2019. arXiv: 1907.00503.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661.

[3] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[4] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. arXiv: 1312.6114.

[5] Geoffrey E. Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, May 2009.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. (Journal Article), 2017.

[7] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]*, September 2017. arXiv: 1708.07747.

[8] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784 [cs, stat]*, November 2014. arXiv: 1411.1784 version: 1.

[9] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. *arXiv:1606.03498 [cs]*, June 2016. arXiv: 1606.03498 version: 1.

[10] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. (Journal Article), 2017.

[11] Cédric Villani. Cyclical monotonicity and Kantorovich duality. In Cédric Villani, editor, *Optimal Transport: Old and New*, Grundlehren der mathematischen Wissenschaften, pages 51–92. Springer, Berlin, Heidelberg, 2009.

[12] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. page 11.

[13] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. *arXiv:1802.05957 [cs, stat]*, February 2018. arXiv: 1802.05957.

[14] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. (Generic), 2017.

[15] Shreyas Patel, Ashutosh Kakadiya, Maitrey Mehta, Raj Derasari, Rahul Patel, and Ratnik Gandhi. Correlated discrete data generation using adversarial training. (Generic), 2018.

[16] Amirsina Torfi and Edward A. Fox. CorGAN: Correlation-Capturing Convolutional Generative Adversarial Networks for Generating Synthetic Healthcare Records. *arXiv:2001.09346 [cs, stat]*, March 2020. arXiv: 2001.09346.

[17] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating Multi-Categorical Samples with Generative Adversarial Networks. *arXiv:1807.01202 [cs, stat]*, July 2018. arXiv: 1807.01202 version: 1.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. *arXiv:1611.01144 [cs, stat]*, August 2017. arXiv: 1611.01144.

[19] Alejandro Mottini, Alix Lheritier, and Rodrigo Acuna-Agost. Airline Passenger Name Record Generation using Generative Adversarial Networks. *arXiv:1807.06657 [cs, stat]*, July 2018. arXiv: 1807.06657.

[20] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17*, pages 1–7, Halifax NS Canada, August 2017. ACM.

[21] Marc G. Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The Cramer Distance as a Solution to Biased Wasserstein Gradients. *arXiv:1705.10743 [cs, stat]*, May 2017. arXiv: 1705.10743.

[22] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data Synthesis based on Generative Adversarial Networks. *Proceedings of the VLDB Endowment*, 11(10):1071–1083, June 2018. arXiv: 1806.03384.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*, January 2016. arXiv: 1511.06434.

[24] Lei Xu and Kalyan Veeramachaneni. Synthesizing Tabular Data using Generative Adversarial Networks. *arXiv:1811.11264 [cs, stat]*, November 2018. arXiv: 1811.11264.

[25] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. PacGAN: The Power of Two Samples in Generative Adversarial Networks. *IEEE Journal on Selected Areas in Information Theory*, 1(1):324–335, May 2020.

[26] Justin Engelmann and Stefan Lessmann. Conditional Wasserstein GAN-based Oversampling of Tabular Data for Imbalanced Learning. *arXiv:2008.09202 [cs]*, August 2020. arXiv: 2008.09202.

[27] Zilong Zhao, Aditya Kunar, Hiek Van der Scheer, Robert Birke, and Lydia Y. Chen. CTAB-GAN: Effective Table Data Synthesizing. *arXiv:2102.08369 [cs]*, February 2021. arXiv: 2102.08369.

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. page 30.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, September 2013. arXiv: 1301.3781.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**