Improving the Robustness of Trading Strategy Backtesting with Boltzmann Machines and Generative Adversarial Networks

Edmond Lezmi
Quantitative Research
Amundi Asset Management, Paris
edmond.lezmi@amundi.com

Jules Roche
Department of Mathematics
Ecole des Ponts ParisTech, Paris
jules.roche@eleves.enpc.fr

Thierry Roncalli
Quantitative Research
Amundi Asset Management, Paris
thierry.roncalli@amundi.com

Jiali Xu
Quantitative Research
Amundi Asset Management, Paris
jiali.xu@amundi.com

June 2020

Abstract

In this article, we explore generative models in order to build a market generator. The underlying idea is to simulate artificial multi-dimensional financial time series, whose statistical properties are the same as those observed in the financial markets. In particular, these synthetic data must preserve the first four statistical moments (mean, standard deviation, skewness and kurtosis), the stochastic dependence between the different dimensions (copula structure) and across time (autocorrelation function). The first part of the article reviews the more relevant generative models, which are restricted Boltzmann machines, generative adversarial networks, and convolutional Wasserstein models. The second part of the article is dedicated to financial applications by considering the simulation of multi-dimensional times series and estimating the probability distribution of backtest statistics. The final objective is to develop a framework for improving the risk management of quantitative investment strategies.

Keywords: Machine learning, generative approach, discriminative approach, restricted Boltzmann machine, generative adversarial network, Wasserstein distance, market generator, quantitative asset management, backtesting, trading strategy.

JEL classification: C53, G11.

1 Introduction

In machine learning, we generally distinguish two types of statistical modeling (Jebara, 2004):

- the generative approach models the unconditional probability distribution $\mathbb{P}(X)$ given a set of observable variables X;
- the discriminative approach models the conditional probability distribution $\mathbb{P}(Y \mid X)$ given a set of observable variables X and a target variable Y.

For instance, examples of generative models are the principal component analysis (PCA) or the method of maximum likelihood (ML) applied to a parametric probability distribution. Examples of discriminative models are the linear regression, the linear discriminant analysis or support vector machines. In the first case, generative models can be used to simulate samples that capture and reproduce the statistical properties of a training dataset. In the second case, discriminative models can be used to predict the target variable Y for new examples of X. Said differently, the distinction between generative and discriminative models can be seen as a reformulation of the distinction between unsupervised and supervised machine learning. More specifically, generative models can be used to learn the underlying probability distributions over data manifolds. The objective of these models is to estimate the statistical properties and correlation structure of real data and simulate synthetic data with a probability distribution, which is close to the real one.

In finance, we generally observe only one sample path of market prices. For instance, if we would like to build a trading strategy on the S&P 500 index, we can backtest the strategy using the historical values of the S&P 500 index. We can then measure the performance and the risk of this investment strategy by computing annualized return, volatility, Sharpe ratio, maximum drawdown, etc. In this case, it is extremely difficult to assess the robustness of the strategy, since we can consider that the historical sample of the S&P 500 index is one realization of the unknown stochastic process. Therefore, portfolio managers generally split the study period into two subperiods: the '*in-sample*' period and the '*out-of-sample*' period. The objective is to calibrate the parameters of the trading strategy with one subperiod and measure the financial performance with the other period in order to reduce the overfitting bias. However, if the out-of-sample approach is appealing, it is limited for two main reasons. First, by splitting the study period into two subperiods, the calibration procedure is performed with fewer observations, and does not generally take into account the most recent period. Second, the validation step is done using only one sample path. Again, we observe only one realization of the risk/return statistics. Of course, we could use different splitting methods, but we know that these bootstrap techniques are not well-adapted to times series and must be reserved for modeling random variables. For stochastic processes, statisticians prefer to consider Monte Carlo methods. Nevertheless, financial times series are difficult to model, because they exhibit non-linear autocorrelations, fat tails, heteroscedasticity, regime switching and non-stationary properties (Cont, 2001).

In this article, we are interested in generative models in order to obtain several training/validation sets. The underlying idea is then to generate artificial but realistic financial market prices. The choice of generative model and market data leads naturally to the concept of market generator introduced by Kondratyev and Schwarz (2019). If the market generator is able to replicate the probability distribution of the original market data, we can then backtest quantitative investment strategies on several financial markets. The backtesting procedure is then largely improved, since we obtain a probability distribution of performance and risk statistics, and not only one value. Therefore, we can reduce the in-sample property of the backtest procedure and the overfitting bias of the parameters that define the trading strategy. However, the challenge is not simple, since the market generator must be sufficiently robust and flexible in order to preserve the uni-dimensional statistical properties of the original financial time series, but also the multi-dimensional dependence structure.

This paper is organized as follows. Section Two reviews the more promising generative models that may be useful in finance. In particular, we focus on restricted Boltzmann machines, generative adversarial networks and Wasserstein distance models. In Section Three, we apply these models in the context of trading strategies. We first consider the joint simulation of S&P 500 and VIX indices. We then build an equity/bond risk parity strategy with an investment universe of six futures contracts. Finally, Section Four offers some concluding remarks.

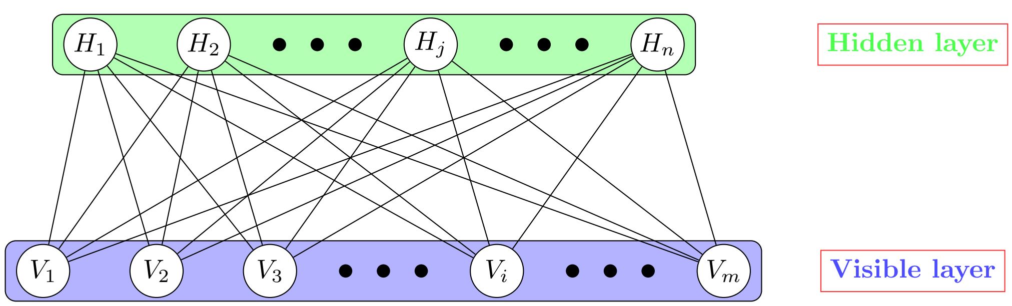
2 Generative models

In this section, we consider two main approaches. The first one is based on a restricted Boltzmann machine (RBM), while the second one uses a conditional generative adversarial network (GAN). Both are stochastic artificial neural networks that learn the probability distribution of a real data sample. However, the objective function strongly differs. Indeed, RBMs consider a log-likelihood maximization problem, while the framework of GANs corresponds to a minimax two-player game. In the first case, the difficulty lies in the gradient approximation of the log-likelihood function. In the second case, the hard task is to find a learning process algorithm that solves the two-player game. This section also presents an extension of generative adversarial networks, which is called a convolutional Wasserstein model.

2.1 Restricted Boltzmann machines

Restricted Boltzmann machines were initially invented under the name '*Harmonium*' by Smolensky (1986). Under the framework of undirected graph models[1], an RBM is a Markov random field (MRF) associated with a bipartite undirected graph[2]. RBMs are made of only two layers as shown in Figure 1. We distinguish visible units belonging to the visible layer from hidden units belonging to the hidden layer. Each unit of visible and hidden layers is respectively associated with visible and hidden random variables. The term '*restricted*' comes from the facts that the connections are only between the visible units and the hidden units and that there is no connection between two different units in the same layer.

Figure 1: The schema of a RBM with m visible units and n hidden units



We would like to model the distribution of m visible variables $V = (V_1, V_2, \ldots, V_m)$ representing the observable data whose elements V_i are highly dependent. A first way to directly model these dependencies is to introduce a Markov chain or a Bayesian network. In this case, networks are no longer restricted and those methods are computationally expensive particularly when V is a high-dimensional vector. The RBM approach consists in introducing hidden variables $H = (H_1, H_2, \ldots, H_n)$ as latent variables which will indirectly capture dependencies. Therefore, the hidden layer can be considered as an alternative representation of the visible layer.

[1] Fundamental concepts of undirected graph model are explained in Appendix A.1 on page 48.

[2] A bipartite graph is a graph whose nodes can be divided into two disjoint and independent sets $\mathcal{U}$ and $\mathcal{V}$ such that every edge connects a node in $\mathcal{U}$ to one in $\mathcal{V}$.

2.1.1 Bernoulli RBMs

Definition A Bernoulli RBM is the standard type of RBMs and has binary-valued visible and hidden variables. Let us denote by $v = (v_1, v_2, \ldots, v_m)$ and $h = (h_1, h_2, \ldots, h_n)$ the configurations of visible variables $V = (V_1, V_2, \ldots, V_m)$ and hidden variables $H = (H_1, H_2, \ldots, H_n)$, where v_i and h_j are the binary states of the i^{th} visible variable V_i and the j^{th} hidden variable H_j such that $(v, h) \in \{0,1\}^{m+n}$. The joint probability distribution of a Bernoulli RBM is given by the Boltzmann distribution:

$$\mathbb{P}(v, h) = \frac{1}{Z} e^{-E(v,h)}$$

where the energy function $E(v, h)$ is defined as:

$$\begin{aligned} E(v,h) &= -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m} \sum_{j=1}^{n} w_{i,j} v_i h_j \\ &= -a^\top v - b^\top h - v^\top W h \end{aligned} \tag{1}$$

where $a = (a_i)$ and $b = (b_j)$ are the two vectors of bias terms associated with V and H and $W = (w_{i,j})$ is the matrix of weights associated with the edges between V and H. The normalizing constant Z is the partition function and ensures the overall distribution sums to one[3]. It follows that the marginal probability distributions of the visible and hidden unit states are:

$$\mathbb{P}(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

and:

$$\mathbb{P}(h) = \frac{1}{Z} \sum_v e^{-E(v,h)}$$

The underlying idea of a Bernoulli RBM is to learn the unconditional probability distributions $\mathbb{P}(v)$ of the observable data.

Conditional Distributions According to Long and Servedio (2010), the partition function Z is intractable in the case of Bernoulli RBMs, since its calculus requires summing up 2^{m+n} elements. Therefore, the probability distribution $\mathbb{P}(v)$ is also intractable when $m+n$ increases. However, we can take advantage of the property of the bipartite graph structure of the RBM. Indeed, there are no connections between two different units in the same layer. The probabilities $\mathbb{P}(v_i \mid h)$ and $\mathbb{P}(h_j \mid v)$ are then independent for all i and j. It follows that:

$$\mathbb{P}(v \mid h) = \mathbb{P}(v_1, v_2, \ldots, v_m \mid h) = \prod_{i=1}^{m} \mathbb{P}(v_i \mid h)$$

and:

$$\mathbb{P}(h \mid v) = \mathbb{P}(h_1, h_2, \ldots, h_n \mid v) = \prod_{j=1}^{n} \mathbb{P}(h_j \mid v)$$

With these properties, we can find some useful results that help when computing the gradient of the log-likelihood function on page 5. For instance, we can show that[4]:

$$\sum_h \mathbb{P}(h \mid v) h_j = \mathbb{P}(h_j = 1 \mid v)$$

[3] We have $Z = \sum_{v,h} e^{-E(v,h)}$.

[4] See Appendix A.2.1 on page 51.

A neural network perspective of RBMs In Appendix A.2.2 on page 51, we show that:

$$\mathbb{P}\left(v_i = 1 \mid h\right) = \sigma\left(a_i + \sum_{j=1}^{n} w_{i,j} h_j\right) \tag{2}$$

and:

$$\mathbb{P}\left(h_j = 1 \mid v\right) = \sigma\left(b_j + \sum_{i=1}^{m} w_{i,j} v_i\right) \tag{3}$$

where $\sigma(x)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Thus, a Bernoulli RBM can be considered as a stochastic artificial neural network, meaning that the nodes and edges correspond to neurons and synaptic connections. For a given vector v, h is obtained as follows:

$$h_j = f\left(\sum_{i=1}^{m} w_{i,j} v_i + b_j\right)$$

where $f = \varphi \circ \sigma$, $\varphi : x \in [0:1] \mapsto X \sim \mathcal{B}(x)$ is a binarizer function and σ is the sigmoid activation function. In addition, we may also go backward in the neural network as follows:

$$v_i = f\left(\sum_{j=1}^{n} w_{i,j} h_j + a_i\right)$$

According to Fischer and Igel (2014), "*an RBM can [then] be reinterpreted as a standard feed-forward neural network with one layer of nonlinear processing units*".

Training process Let $\theta = (a, b, W)$ be the set of parameters to estimate. The objective is to find a value of θ such that $\mathbb{P}_\theta(v) \approx \mathbb{P}_{\text{data}}(v)$. Since the log-likelihood function $\ell(\theta \mid v)$ of the input vector v is defined as $\ell(\theta \mid v) = \log \mathbb{P}_\theta(v)$, the Bernoulli RBM model is trained in order to maximize the log-likelihood function of a training set of N samples $\left\{v_{(1)}, \ldots, v_{(N)}\right\}$:

$$\theta^\star = \arg\max_\theta \sum_{s=1}^{N} \ell\left(\theta \mid v_{(s)}\right) \tag{4}$$

where:

$$\begin{aligned}
\ell(\theta \mid v) &= \log\left(\sum_h \frac{e^{-E(v,h)}}{Z}\right) \\
&= \log\left(\sum_h e^{-E(v,h)}\right) - \log\left(\sum_{v',h} e^{-E\left(v',h\right)}\right)
\end{aligned} \tag{5}$$

Hinton (2002) proposed to use gradient ascent method with the following update rule between iteration steps t and $t+1$:

$$\begin{aligned}
\theta^{(t+1)} &= \theta^{(t)} + \eta^{(t)} \frac{\partial}{\partial\, \theta}\left(\sum_{s=1}^{N} \ell\left(\theta^{(t)} \mid v_{(s)}\right)\right) \\
&= \theta^{(t)} + \eta^{(t)} \Delta\theta^{(t)}
\end{aligned}$$

where $\eta^{(t)}$ is the learning rate parameter, $\Delta\theta^{(t)} = \sum_{s=1}^{N} \nabla_{\theta^{(t)}}\left(v_{(s)}\right)$ and $\nabla_\theta(v)$ is the gradient vector given in Appendix A.2.3 on page 52.

Gibbs sampling Ackley *et al.* (1985) and Hinton and Sejnowski (1986) showed that the expectation over $\mathbb{P}(v)$ can be approximated by Gibbs sampling, which belongs to the family of MCMC algorithms. The goal of Gibbs sampling is to simulate correlated random variables by using a Markov chain. Usually, we initialize the Gibbs sampling with a random vector and the algorithm updates one variable iteratively, based on its conditional distribution given the state of the remaining variables. After a sufficiently large number of sampling steps, we get the unbiased samples from the joint probability distribution. Formally, Gibbs sampling of the joint probability distribution of n random variables $X = (X_1, X_2, \ldots, X_n)$ consists in sampling $x_i \sim \mathbb{P}(X_i \mid X_{-i} = x_{-i})$ iteratively.

Algorithm 1 Gibbs sampling

initialization: $x^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$
for step $k = 1, 2, 3, \ldots$ **do**
Sample $x^{(k)}$ as follows:

$$\begin{aligned}
x_1^{(k)} &\sim \mathbb{P}\left(X_1 \mid x_2^{(k-1)}, x_3^{(k-1)}, \ldots, x_n^{(k-1)}\right) \\
x_2^{(k)} &\sim \mathbb{P}\left(X_2 \mid x_1^{(k)}, x_3^{(k-1)}, \ldots, x_n^{(k-1)}\right) \\
&\vdots \\
x_i^{(k)} &\sim \mathbb{P}\left(X_i \mid x_1^{(k)}, \ldots, x_{i-1}^{(k)}, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)}\right) \\
&\vdots \\
x_n^{(k)} &\sim \mathbb{P}\left(X_n \mid x_1^{(k)}, x_2^{(k)}, \ldots, x_{n-1}^{(k)}\right)
\end{aligned}$$

end for

Let us consider a Bernoulli RBM as a Markov random field defined by the random variables $V = (V_1, V_2, \ldots, V_m)$ and $H = (H_1, H_2, \ldots, H_n)$. Since an RBM is a bipartite undirected graph, we can take advantage of conditional independence properties between the variables in the same layer. At each step, we jointly sample the states of all variables in one layer, as shown in Figure 2. Thus, Gibbs sampling for an RBM consists in alternating[5] between sampling a new state h for all hidden units based on $\mathbb{P}(h \mid v)$ and sampling a new state v for all visible units based on $\mathbb{P}(v \mid h)$. Let $v^{(k)} = \left(v_1^{(k)}, v_2^{(k)}, \ldots, v_m^{(k)}\right)$ and $h^{(k)} = \left(h_1^{(k)}, h_2^{(k)}, \ldots, h_n^{(k)}\right)$ denote the states of the visible layer and the hidden layer at time step k. For each unit, we rely on the fact that the conditional probabilities $\mathbb{P}\left(v_i^{(k)} \mid h^{(k)}\right)$ and $\mathbb{P}\left(h_j^{(k)} \mid v^{(k-1)}\right)$ are easily tractable according to Equations (2) and (3). We start by initializing the state of the visible units and we can choose a binary random vector for the first time step. At time step k, here are the steps of the algorithm:

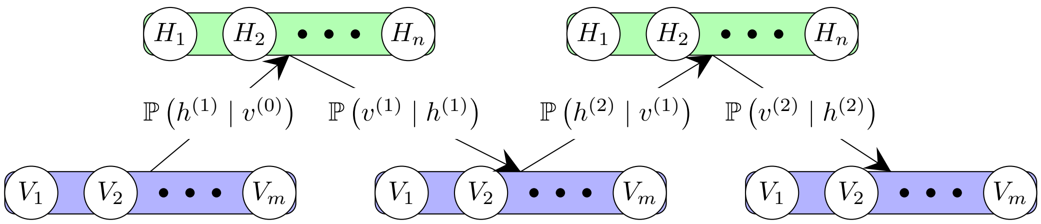
1. We go forward in the network by computing simultaneously for each hidden unit j the following probability $p\left(h_j^{(k)}\right) = \mathbb{P}\left(h_j^{(k)} \mid v^{(k-1)}\right)$. The state of the j^{th} is then simulated according to the Bernoulli distribution $\mathcal{B}\left(p\left(h_j^{(k)}\right)\right)$.

2. We go backward in the network by computing simultaneously for each visible unit i

[5] This method is also known as block Gibbs sampling.

the following probability $p\left(v_i^{(k)}\right) = \mathbb{P}\left(v_i^{(k)} \mid h^{(k)}\right)$. The state of the i^{th} unit is then simulated according to the Bernoulli distribution $\mathcal{B}\left(p\left(v_i^{(k)}\right)\right)$.

Figure 2: The schema of blocks Gibbs sampling for an RBM



Contrastive divergence algorithm As shown in the previous paragraph, it is possible to use Gibbs sampling to get samples from the joint distribution $\mathbb{P}(v,h)$. However, the computational effort is still too large since the sampling chain needs many sampling steps to obtain unbiased samples. To address this issue, Hinton (2002) initialized the Gibbs sampling with a sample from the real data, instead of using a random vector and suggested the use of only a few sampling steps to get a sample that could produce a sufficiently good approximation of log-likelihood gradient. This faster method is called contrastive divergence algorithm.

Given a training set of N samples $\left\{v_{(1)}, \ldots, v_{(N)}\right\}$, we obtain:

$$\begin{aligned} \frac{1}{N}\sum_{s=1}^{N} \ell\left(\theta \mid v_{(s)}\right) &= \frac{1}{N}\sum_{s=1}^{N} \log \mathbb{P}_\theta\left(v_{(s)}\right) \\ &= \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log \mathbb{P}_\theta(v)\right] \end{aligned} \tag{6}$$

We note $\mathbb{P}_{\text{model}}(v) = \mathbb{P}_\theta(v)$. Since $\mathbb{P}_{\text{data}}$ is independent of θ, maximizing the log-likelihood function (5) is equivalent to minimizing the Kullback-Leibler divergence between $\mathbb{P}_{\text{data}}$ and $\mathbb{P}_{\text{model}}$:

$$\begin{aligned} \theta^\star &= \arg\max_\theta \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log \mathbb{P}_\theta(v)\right] - \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log \mathbb{P}_{\text{data}}(v)\right] \\ &= \arg\max_\theta \sum_v \mathbb{P}_{\text{data}}(v)\log \mathbb{P}_{\text{model}}(v) - \sum_v \mathbb{P}_{\text{data}}(v)\log \mathbb{P}_{\text{data}}(v) \\ &= \arg\max_\theta \sum_v \mathbb{P}_{\text{data}}(v)\log\left(\frac{\mathbb{P}_{\text{model}}(v)}{\mathbb{P}_{\text{data}}(v)}\right) \\ &= \arg\max_\theta -\text{KL}\left(\mathbb{P}_{\text{data}} \parallel \mathbb{P}_{\text{model}}\right) \\ &= \arg\min_\theta \text{KL}\left(\mathbb{P}_{\text{data}} \parallel \mathbb{P}_{\text{model}}\right) \end{aligned} \tag{7}$$

In contrastive divergence algorithms, we note the distribution of starting values as $\mathbb{P}^{(0)}$, which is also the distribution of real data in training set: $\mathbb{P}^{(0)} = \mathbb{P}_{\text{data}}$. Let $\mathbb{P}^{(k)}$ and $\mathbb{P}^{(\infty)}$ be the distribution after running k steps of Gibbs sampling and the equilibrium distribution. Compared to $\mathbb{P}^{(0)}$, $\mathbb{P}^{(k)}$ is k steps closer to the equilibrium distribution $\mathbb{P}^{(\infty)}$, so the divergence measure $\text{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right)$ should be greater than or equal to $\text{KL}\left(\mathbb{P}^{(k)} \parallel \mathbb{P}^{(\infty)}\right)$. In particular, when the model is well-trained, $\mathbb{P}^{(0)}$, $\mathbb{P}^{(k)}$, and $\mathbb{P}^{(\infty)}$

should have the same distribution as $\mathbb{P}_{\text{data}}$. In this case, $\text{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right)$ is equal to 0 and the difference $\text{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right) - \text{KL}\left(\mathbb{P}^{(k)} \parallel \mathbb{P}^{(\infty)}\right)$ is also equal to 0. Thus, according to Hinton (2002), we can find the optimal parameter $\theta^\star$ by minimizing the difference $\text{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right) - \text{KL}\left(\mathbb{P}^{(k)} \parallel \mathbb{P}^{(\infty)}\right)$ instead of minimizing directly $\text{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right)$ in Equation (7). This quantity is called a contrastive divergence since we compare two KL divergence measures. Therefore, we can rewrite the objective function in Equation (7) as follows:

$$\begin{aligned}\theta^\star &= \arg\min_\theta \text{CD}^{(k)} \\ &= \arg\min_\theta \text{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right) - \text{KL}\left(\mathbb{P}^{(k)} \parallel \mathbb{P}^{(\infty)}\right)\end{aligned} \tag{8}$$

Again, we can use the gradient descent method to find the minimum of the objective function. In this case, the update rule between iteration steps t and $t+1$ is:

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} \frac{\partial \, \text{CD}^{(k)}\left(\theta^{(t)}\right)}{\partial \, \theta}$$

where the gradient vector $\partial_\theta \, \text{CD}^{(k)}\left(\theta^{(t)}\right)$ is given in Appendix A.2.4 on page 54. Thus, Algorithm (2) summarizes the k-step contrastive divergence algorithm for calibrating $\theta^\star$.

2.1.2 Gaussian-Bernoulli RBMs

A Bernoulli RBM is limited to modelling the probability distribution $\mathbb{P}(v)$ where v is a binary vector. To address this issue, given an RBM with m visible units and n hidden units, we can associate a normally distributed variable to each visible unit and a binary variable to each hidden unit. This type of RBMs is called Gaussian-Bernoulli RBM. We are free to choose the expression of the energy function as long as it satisfies the Hammersley-Clifford theorem[6] and its partition function is well defined. For instance, Cho *et al.* (2011) defined the energy function of this RBM as follows:

$$E_g(v, h) = \sum_{i=1}^m \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^n b_j h_j - \sum_{i=1}^m \sum_{j=1}^n w_{i,j} \frac{v_i h_j}{\sigma_i^2}$$

where a_i and b_j are bias terms associated with visible variables V_i and hidden variables H_j, $w_{i,j}$ is the weight associated with the edge between V_i and H_j, and σ_i is a new parameter associated with V_i. Following the same calculus in Equation 3, we can show that the conditional probability is equal to:

$$\mathbb{P}(h_j = 1 \mid v) = \text{sigmoid}\left(b_j + \sum_{i=1}^m w_{i,j} \frac{v_i}{\sigma_i^2}\right) \tag{9}$$

Moreover, according to Krizhevsky (2009), the conditional distribution of V_i given h is Gaussian and we obtain:

$$V_i \mid h \sim \mathcal{N}\left(a_i + \sum_{j=1}^n w_{i,j} h_j, \sigma_i^2\right) \tag{10}$$

As we have previously seen, we can maximize the log-likelihood function in order to train a Gaussian-Bernoulli RBM with m visible units and n hidden units. As all visible units are

[6] See Appendix A.1.3 on page 48.

Algorithm 2 k-step contrastive divergence algorithm

input: $\{v_{(1)}, \ldots, v_{(N)}\}$
initialization: We note $\Delta a_i = \partial_{a_i} \mathrm{CD}^{(k)}$, $\Delta b_j = \partial_{b_i} \mathrm{CD}^{(k)}$ and $\Delta w_{i,j} = \partial_{w_{i,j}} \mathrm{CD}^{(k)}$, and we set $\Delta a_i = \Delta b_j = \Delta w_{i,j} = 0$
for v from $v_{(1)}$ to $v_{(N)}$ **do**
  $v^{(0)} \leftarrow v$
  {Gibbs sampling}
  for $t = 1$ to k **do**
    for $j = 1$ to n **do**
      compute $p\left(h_j^{(t)}\right) = \mathbb{P}\left(h_j^{(t)} \mid v^{(t-1)}\right)$
      sample the j^{th} hidden unit state: $h_j^{(t)} \sim \mathcal{B}\left(p\left(h_j^{(t)}\right)\right)$
    end for
    for $i = 1$ to m **do**
      compute $p\left(v_i^{(t)}\right) = \mathbb{P}\left(v_i^{(t)} \mid h^{(t)}\right)$
      sample the i^{th} visible unit state: $v_i^{(t)} \sim \mathcal{B}\left(p\left(v_i^{(t)}\right)\right)$
    end for
  end for
  {Gradient approximation}
  for $i = 1$ to m **do**
    $\Delta a_i \leftarrow \Delta a_i + \frac{1}{N}\left(v_i^{(k)} - v_i^{(0)}\right)$
    for $j = 1$ to n **do**
      $\Delta b_j \leftarrow \Delta b_j + \frac{1}{N}\left(\mathbb{P}\left(h_j = 1 \mid v^{(k)}\right) - \mathbb{P}\left(h_j = 1 \mid v^{(0)}\right)\right)$
      $\Delta w_{i,j} \leftarrow \Delta w_{i,j} + \frac{1}{N}\left(\mathbb{P}\left(h_j = 1 \mid v^{(k)}\right) \cdot v_i^{(k)} - \mathbb{P}\left(h_j = 1 \mid v^{(0)}\right) \cdot v_i^{(0)}\right)$
    end for
  end for
end for
Output: gradient estimation Δa_i, Δb_j and $\Delta w_{i,j}$

associated with continuous probability distribution, the log-likelihood function $\ell(\theta \mid v)$ of an input vector $v = (v_1, v_2, \ldots, v_m)$ is equal to $\log p_\theta(v)$ where $p_\theta(v)$ is the probability density function of $V = (V_1, V_2, \ldots, V_m)$. In Appendix A.2.5 on page 55, we give the expression of the gradient vector. Therefore, there is no difficulty to use the gradient ascent method or the k-step contrastive divergence algorithm to train a Gaussian-Bernoulli RBM.

Remark 1. *If we normalize the data of the training set using a z-score function, we can set the standard deviation σ_i to 1 during the training process. This reduces the number of parameter and accelerates the convergence of the training process.*

2.1.3 Conditional RBM structure

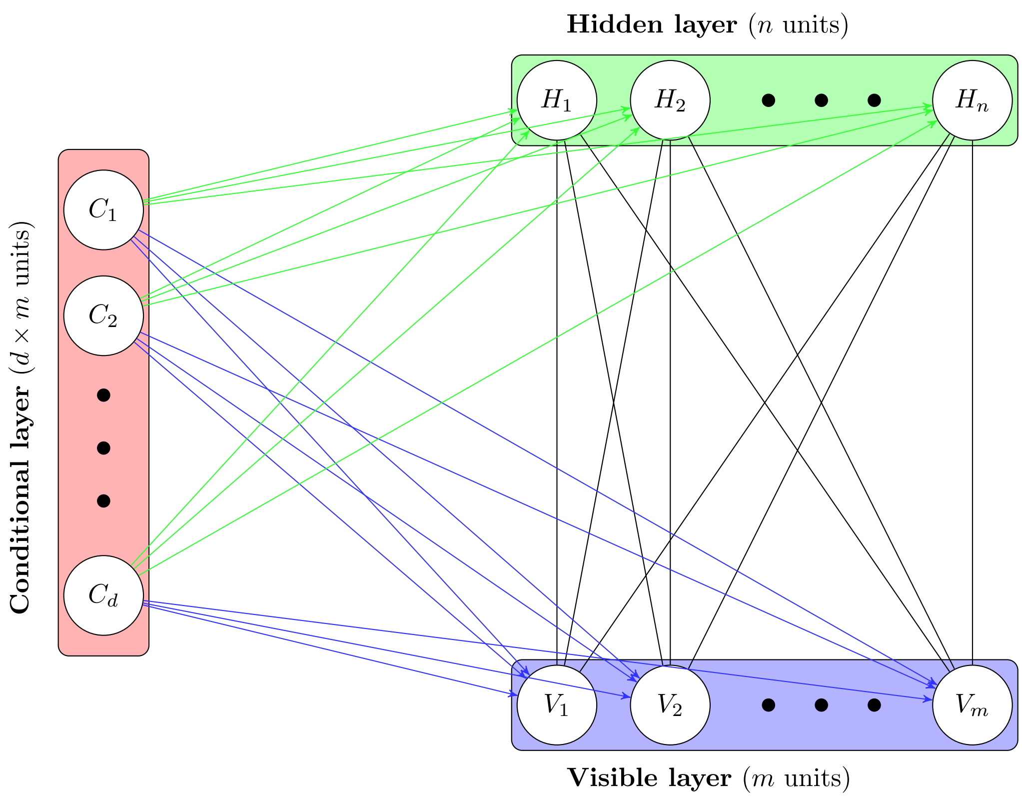
Traditional Bernoulli and Gaussian-Bernoulli RBMs can only model the static dependence of variables. However, in the practice of financial data modeling, we also want to capture the temporal dependencies between variables. To address this issue, Taylor *et al.* (2011) introduced a conditional RBM structure by adding a new layer to the Gaussian-Bernoulli RBM. Thus, this conditional RBM is made of three parts as shown in Figure 3:

1. a hidden layer with n binary units;

2. a visible layer with m units;

3. a conditional layer with d conditional meta-units $\{C_1, \ldots, C_d\}$. Since we model the temporal structure, we note the observation at time t as $v_t = (v_{t,1}, v_{t,2}, \ldots, v_{t,m})$. The conditional layer is fed with d past values $(v_{t-1}, \ldots, v_{t-d})$ that are concatenated into a $d \times m$-dimensional vector[7] and we note it as c_t. The conditional layer is fully linked to visible and hidden layers with directed connections. Let us denote by $Q = (q_{d\times m,m})$ the weight matrix connecting the conditional layer to the visible layer and $P = (p_{d\times m,n})$ the weight matrix connecting the conditional layer to the hidden layer.

Figure 3: The schema of a conditional RBM with m visible layer units, n hidden layer units and $d \times m$ conditional layer units



A conditional RBM contains both undirected and directed connections in the graph. Thus, it can't be defined as an MRF or Bayesian network. However, conditionally on c_t, we can consider both visible and hidden layers as an undirected graph and we can compute $\mathbb{P}(v_t, h_t \mid c_t)$ instead of computing $\mathbb{P}(v_t, h_t, c_t)$ in order to take still advantage of undirected graph properties. Thus, according to the Hammersley-Clifford theorem, the conditional probability distribution has the following form:

$$\begin{aligned}\mathbb{P}(v_t, h_t \mid c_t) &= \frac{1}{Z} e^{-\tilde{E}_g(v_t,h_t,c_t)} \\ &= \frac{1}{Z} e^{-\sum_{i=1}^m \tilde{E}_i(v_{t,i},c_t) - \sum_{j=1}^n \tilde{E}_j(h_{t,j},c_t) - \sum_{i=1}^m \sum_{j=1}^n \tilde{E}_{i,j}(v_{t,i},h_{t,j},c_t)}\end{aligned}$$

[7] A conditional meta-unit is then composed of m values: $C_k = (V_{t-k,1} \ldots, V_{t-k,m})$.

where Z is the partition function. Taylor *et al.* (2011) proposed a form of energy function that is an extension of the Gaussian-Bernoulli RBM energy:

$$\begin{aligned} \tilde{E}_i\left(v_{t,i}, c_t\right) &= \frac{\left(v_{t,i} - \tilde{a}_{t,i}\right)^2}{2\sigma_i^2} \\ \tilde{E}_j\left(h_{t,j}, c_t\right) &= -\tilde{b}_{t,j} h_{t,j} \\ \tilde{E}_{i,j}\left(v_{t,i}, h_{t,j}, c_t\right) &= -w_{i,j} \frac{v_{t,i} h_{t,j}}{\sigma_i^2} \end{aligned}$$

where $\tilde{a}_t = a + c_t Q^\top$ and $\tilde{b}_t = b + c_t P^\top$ are dynamic bias terms with respect to a and b. Thus, the energy function $\tilde{E}_g$ corresponds to a Gaussian-Bernoulli RBM energy function E_g by replacing constant biases a and b by dynamic bias $\tilde{a}_t$ and $\tilde{b}_t$. By updating these terms in Equations (9) and (10), we obtain:

$$\mathbb{P}\left(h_j = 1 \mid v_t, c_t\right) = \operatorname{sigmoid}\left(\tilde{b}_{t,j} + \sum_{i=1}^{m} w_{i,j} \frac{v_{t,i}}{\sigma_i^2}\right) \tag{11}$$

and:

$$V_{t,i} \mid h_t, c_t \sim \mathcal{N}\left(\tilde{a}_{t,i} + \sum_{j=1}^{n} w_{i,j} h_{t,j}, \sigma_i^2\right) \tag{12}$$

Again, the partition function Z is still intractable:

$$Z = \sum_{h_t} \int_{v_t} e^{-\tilde{E}_g(v_t, h_t, c_t)}\, \mathrm{d}v_t$$

Remark 2. *As for Gaussian-Bernoulli RBMs, we can use the k-step contrastive divergence algorithm for the training process with some modifications of the gradient vector*[8].

2.2 Generative adversarial networks

Generative models are an important part of machine learning algorithms that learn the underlying probability distributions of the real data sample. In other words, given a finite sample data with a distribution $\mathbb{P}_{\text{data}}(x)$, can we build a model such that $\mathbb{P}_{\text{model}}(x; \theta) \approx \mathbb{P}_{\text{data}}(x)$? The goal is to learn to sample a complex distribution given a real sample. Generative adversarial networks (GANs) belong to the class of generative models and move away from the classical likelihood maximization approach, whose objective is to estimate the parameter θ. GAN models enable the estimation of the high dimensional underlying statistical structure of real data and simulate synthetic data, whose probability distribution is close to the real one. To assess the difference between real and simulated data, GANs are trained using a discrepancy measure. Two widely classes of discrepancy measures are information-theoretic divergences and integral probability metrics. The choice of the GAN objective function associated with the selected discrepancy measure explains the multitude of GAN models appearing in the machine learning literature. However, the different GAN models share a common framework. Indeed, Appendix A.3 page 58 shows how the original formulation of GAN models is a particular case of the theory of ϕ-divergence and probability functional descent. Moreover, the link between influence function used in robust statistics and the formulation of the discriminator and generator is derived.

[8]The calculations are given in Appendix A.2.6 on page 56.

According to Goodfellow *et al.* (2014), "*a generative adversarial process trains two models: a generative model $\boldsymbol{\mathcal{G}}$ that captures the data distribution, and a discriminative model $\boldsymbol{\mathcal{D}}$ that estimates the probability that a sample came from the training data rather than $\boldsymbol{\mathcal{G}}$. The training procedure for $\boldsymbol{\mathcal{G}}$ is to maximize the probability of $\boldsymbol{\mathcal{D}}$ making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{D}}$, a unique solution exists, with $\boldsymbol{\mathcal{G}}$ recovering the training data distribution and $\boldsymbol{\mathcal{D}}$ equal to $1/2$ everywhere*". Therefore, a generative adversarial model consists of two neural networks. The first neural network simulates a new data sample, and is interpreted as a data generation process. The second neural network is a classifier. The input data are both the real and simulated samples. If the generative model has done a good job, the discriminative model is unable to know whether an observation comes from the real dataset or the simulated dataset.

2.2.1 Adversarial training problem

We use the framework of Goodfellow *et al.* (2014) and Wiese *et al.* (2020). Let $\mathcal{Z}$ be a random noise space. $z \in \mathcal{Z}$ is sampled from the prior distribution $\mathbb{P}_{\mathrm{noize}}(z)$. The generative model $\boldsymbol{\mathcal{G}}$ is then specified as follows:

$$\boldsymbol{\mathcal{G}} : \left\{ \begin{array}{l} (\mathcal{Z}, \Theta_{\boldsymbol{\mathcal{G}}}) \longrightarrow \mathcal{X} \\ (z, \theta_g) \longmapsto x_0 = \boldsymbol{\mathcal{G}}(z; \theta_g) \end{array} \right. \tag{13}$$

where $\mathcal{X}$ denotes the data space and $\Theta_{\boldsymbol{\mathcal{G}}}$ defines the parameter space including generator weights and bias that will be optimized. The generator $\boldsymbol{\mathcal{G}}$ helps to simulate data x_0. The discriminative model is defined as follows:

$$\boldsymbol{\mathcal{D}} : \left\{ \begin{array}{l} (\mathcal{X}, \Theta_{\boldsymbol{\mathcal{D}}}) \longrightarrow [0,1] \\ (x, \theta_d) \longmapsto p = \boldsymbol{\mathcal{D}}(x; \theta_d) \end{array} \right. \tag{14}$$

where x corresponds to the set of simulated data x_0 and training (or real) data x_1. In this approach, the statistical model of X corresponds to:

$$\mathbb{P}_{\mathrm{model}}(x; \theta) \sim \boldsymbol{\mathcal{G}}(\mathbb{P}_{\mathrm{noise}}(z); \theta_g) \tag{15}$$

The probability that the observation comes from the real (or true) data is equal to $p_1 = \boldsymbol{\mathcal{D}}(x_1; \theta_d)$, whereas the probability that the observation is simulated is given by $p_0 = \boldsymbol{\mathcal{D}}(x_0; \theta_d)$. If the model (15) is wrong and does not reproduce the statistical properties of the real data, the classifier has no difficulty in separating the simulated data from the real data, and we obtain $p_0 \approx 0$ and $p_1 \approx 1$. Otherwise, if the model is valid, we must verify that:

$$p_0 \approx p_1 \approx \frac{1}{2}$$

The main issue of GANs is the specification of the two functions $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{D}}$ and the estimation of the parameters θ_g and θ_d associated to $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{D}}$. For the first step, Goodfellow *et al.* (2014) proposed to use two multi-layer neural networks $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{D}}$, whereas they consider the following cost function for the second step:

$$\begin{aligned} \mathcal{C}(\theta_g, \theta_d) \quad = \quad & \mathbb{E}\left[\log \boldsymbol{\mathcal{D}}(x_1; \theta_d) \mid x_1 \sim \mathbb{P}_{\mathrm{data}}(x)\right] + \\ & \mathbb{E}\left[\log\left(1 - \boldsymbol{\mathcal{D}}(x_0; \theta_d)\right) \mid x_0 \sim \boldsymbol{\mathcal{G}}(z; \theta_g), z \sim \mathbb{P}_{\mathrm{noise}}(z)\right] \end{aligned}$$

The optimization problem becomes:

$$\left\{\hat{\theta}_g, \hat{\theta}_d\right\} = \arg \min_{\theta_g \in \Theta_{\boldsymbol{\mathcal{G}}}} \max_{\theta_d \in \Theta_{\boldsymbol{\mathcal{D}}}} \mathcal{C}(\theta_g, \theta_d)$$

In other words, the discriminator is trained in order to maximize the probability to correctly classify historical samples from simulated samples. The objective is to obtain a good classification model since the maximum value $\mathcal{C}(\theta_g, \theta_d)$ with respect to θ_d is reached when:

$$\begin{cases} \boldsymbol{\mathcal{D}}(x_1; \theta_d) = 1 \\ \boldsymbol{\mathcal{D}}(x_0; \theta_d) = 0 \end{cases}$$

In the meantime, the generator is trained in order to minimize the probability that the discriminator is able to perform a correct classification or equivalently to maximize the probability to fool the discriminator, since the minimum value $\mathcal{C}(\theta_g, \theta_d)$ with respect to θ_g is reached when:

$$\begin{cases} \boldsymbol{\mathcal{D}}(x_0; \theta_d) = 1 \\ x_0 \sim \boldsymbol{\mathcal{G}}(z; \theta_g) \end{cases}$$

Remark 3. *In Appendix A.5 on page 67, we show that the cost function is related to the binary cross-entropy measure or the opposite of the log-likelihood function of the logit model. Moreover, the cost function can be interpreted as a ϕ-divergence measure as explained in Appendix A.3.1 on page 58.*

2.2.2 Solving the optimization problem

The minimax optimization problem is difficult to solve directly, because the gradient vector $\nabla_\theta \mathcal{C}(\theta_g, \theta_d)$ is not well informative if the discriminative model is poor. Therefore, the traditional way to solve this problem is to use a two-stage approach:

1. In a first stage, the vector of parameters θ_g is considered to be constant whereas the vector of parameters θ_d is unknown. This implies that the minimax problem reduces to a maximization step:

$$\hat{\theta}_d^{(\max)} = \arg \max_{\theta_d \in \Theta_{\boldsymbol{\mathcal{D}}}} \mathcal{C}^{(\max)}(\theta_d \mid \theta_g)$$

   where $\mathcal{C}^{(\max)}(\theta_d \mid \theta_g)$ corresponds to the cost function by assuming that θ_g is given.

2. In a second stage, the vector of parameters θ_d is considered to be constant whereas the vector of parameters θ_g is unknown. This implies that the minimax problem reduces to a minimization step:

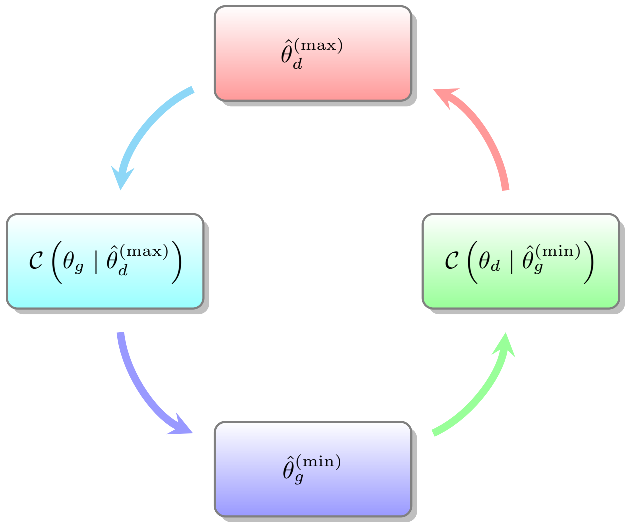
$$\hat{\theta}_g^{(\min)} = \arg \min_{\theta_g \in \Theta_{\boldsymbol{\mathcal{G}}}} \mathcal{C}^{(\min)}(\theta_g \mid \theta_d)$$

   where $\mathcal{C}^{(\min)}(\theta_g \mid \theta_d)$ corresponds to the cost function by assuming that θ_d is given.

The two-stage approach is repeated until convergence by setting θ_g to the value $\hat{\theta}_g^{(\min)}$ calculated at the minimization step and θ_d to the value $\hat{\theta}_d^{(\max)}$ calculated at the maximization step. The cycle sequence is given in Figure 4. A drawback of this approach is the computational time. Indeed, this implies to solve two optimization problems at each iteration.

Therefore, Goodfellow *et al.* (2014) proposed another two-stage approach, which converges more rapidly. The underlying idea is not to estimate the optimal generator at each iteration. The objective is, rather, to improve the generative model at each iteration, such that the new estimated model is better than the previous. The convergence is only needed for the discriminator step (Goodfellow *et al.*, 2014, Proposition 2). Moreover, the authors

Figure 4: Two-stage minimax algorithm



applied a mini-batch sampling in order to reduce the computational time. It follows that the cost functions become:

$$\mathcal{C}^{(\max)}\left(\theta_d \mid \theta_g\right) \approx \frac{1}{m}\sum_{i=1}^{m} \log \boldsymbol{\mathcal{D}}\left(x_1^{(i)};\theta_d\right) + \log\left(1 - \boldsymbol{\mathcal{D}}\left(x_0^{(i)};\theta_d\right)\right)$$

and:

$$\mathcal{C}^{(\min)}\left(\theta_g \mid \theta_d\right) \approx c + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - \boldsymbol{\mathcal{D}}\left(\boldsymbol{\mathcal{G}}\left(z^{(i)};\theta_g\right);\theta_d\right)\right)$$

where m is the size of the mini-batch sample and c is a constant that does not depend on the generator parameters θ_g:

$$c = \frac{1}{m}\sum_{i=1}^{m} \log \boldsymbol{\mathcal{D}}\left(x_1^{(i)};\theta_d\right)$$

In Algorithm (3), we describe the stochastic gradient optimization method proposed by Goodfellow *et al.* (2014) when the learning rule corresponds to the steepest descent method. But other learning rules can be used such as the momentum method or the adaptive learning method. In practice, these algorithms may not converge[9]. Another big issue known as '*mode collapse*' concerns the diversity of generated samples. In this case, the simulated data drawn by the generator exhibit small and limited differences, meaning all generated samples are almost identical. This is why researchers have proposed alternatives to Algorithm (3) in order to solve these two problems (Mao *et al.*, 2017; Karras *et al.*, 2018; Metz *et al.*, 2017).

2.2.3 Time series modeling with GANs

It is possible to generate fake time series from a single random noise vector z without any labels. However, this implies the use of complex structures for both generator and

[9]For instance, we may observe a cycle because the parameters oscillate

Algorithm 3 Stochastic gradient optimization for GAN training

The goal is to compute the optimal parameters $\hat{\theta}_g$ and $\hat{\theta}_d$

We note $n_{\max}$ the number of iterations to apply to the maximization problem (discriminator step) and $n_{\min}$ the number of iterations to apply to the minimization problem (generator step)

m is the size of the mini-batch sample

The starting values are denoted by $\theta_g^{(0)}$ and $\theta_d^{(0)}$

for $j = 1$ to $n_{\min}$ **do**

$\theta_d^{(j,0)} \leftarrow \theta_d^{(j-1)}$

for $k = 1$ to $n_{\max}$ **do**

Sample m random noise vector $\left(z^{(1)}, \ldots, z^{(m)}\right)$ from the prior distribution $\mathbb{P}_{\text{noise}}(z)$

Compute the simulated data $\left(x_0^{(1)}, \ldots, x_0^{(m)}\right)$:

$$x_0^{(i)} \sim \boldsymbol{\mathcal{G}}\left(z^{(i)}; \theta_g^{(j-1)}\right)$$

Sample m examples $\left(x^{(1)}, \ldots, x^{(m)}\right)$ from the data distribution $\mathbb{P}_{\text{data}}(x)$

Compute the gradient vector of the maximization problem with respect to the parameter θ_d:

$$\Delta_{\theta_d}^{(j,k)} \leftarrow \nabla_{\theta_d}\left(\frac{1}{m}\sum_{i=1}^{m} \log \boldsymbol{\mathcal{D}}\left(x_1^{(i)}; \theta_d\right) + \log\left(1 - \boldsymbol{\mathcal{D}}\left(x_0^{(i)}\right); \theta_d\right)\right)\Bigg|_{\theta_d = \theta_d^{(j,k-1)}}$$

Update the discriminator parameters θ_d using a backpropagation learning rule. For instance, in the case of the steepest descent method, we obtain:

$$\theta_d^{(j,k)} \leftarrow \theta_d^{(j,k-1)} + \eta_d \cdot \Delta_{\theta_d}^{(j,k)}$$

end for

$\theta_d^{(j)} \leftarrow \theta_d^{(j,n_{\max})}$

Sample m random noise vector $\left(z^{(1)}, \ldots, z^{(m)}\right)$ from the prior distribution $\mathbb{P}_{\text{noise}}(z)$

Compute the gradient vector of the minimization problem with respect to the parameter θ_g:

$$\Delta_{\theta_g}^{(j)} \leftarrow \nabla_{\theta_g}\left(\frac{1}{m}\sum_{i=1}^{m} \log\left(1 - \boldsymbol{\mathcal{D}}\left(\boldsymbol{\mathcal{G}}\left(z^{(i)}; \theta_g\right); \theta_d^{(j)}\right)\right)\right)\Bigg|_{\theta_g = \theta_g^{(j-1)}}$$

Update the generator parameters θ_g using a backpropagation learning rule. For instance, in the case of the steepest descent method, we obtain:

$$\theta_g^{(j)} \leftarrow \theta_g^{(j-1)} - \eta_g \cdot \Delta_{\theta_g}^{(j)}$$

end for

return $\hat{\theta}_g \leftarrow \theta_g^{(n_{\min})}$ and $\hat{\theta}_d \leftarrow \theta_d^{(n_{\min})}$

Source: Goodfellow *et al.* (2014, Algorithm 1).

discriminator. These structures can cause the models to be computationally expensive, particularly when using convolutional networks. Moreover, such structures don't address the issue concerning the lack of data. Let us remember that we only have one single historical scenario. A less expensive alternative is to use labels combined with simple multi-layer perceptrons for the generator and the discriminator. This approach has been introduced by Mirza and Osindero (2014) and is called conditional generative adversarial networks (cGANs).

Before training a GAN, the data need to be labelled. An additional vector that encodes structural information about real historical data must then be defined. It will help the GAN to generate specific scenarios depending on labels defined before. Therefore, the learning process will be supervised. The previous framework remains valid, but the generative and discriminative models are written as $\boldsymbol{\mathcal{G}}\left(z \mid v; \theta_g\right)$ and $\boldsymbol{\mathcal{D}}\left(x \mid v; \theta_d\right)$, where $v \in \mathcal{V}$ is the label vector. Therefore, the cost function becomes:

$$\begin{aligned}\mathcal{C}\left(\theta_g, \theta_d\right) &= \mathbb{E}\left[\log \boldsymbol{\mathcal{D}}\left(x_1 \mid v; \theta_d\right) \mid x_1 \sim \mathbb{P}_{\mathrm{data}}\left(x \mid v\right)\right] + \\ & \quad \mathbb{E}\left[\log\left(1 - \boldsymbol{\mathcal{D}}\left(x_0 \mid v; \theta_d\right)\right) \mid x_0 \sim \mathbb{P}_{\mathrm{mod\,el}}\left(x \mid v\right)\right]\end{aligned}$$

where $\mathbb{P}_{\mathrm{mod\,el}}\left(x \mid v\right) = \boldsymbol{\mathcal{G}}\left(\mathbb{P}_{\mathrm{noise}}\left(z\right) \mid v; \theta_g\right)$.

The label vector may encode various types of information, and can be categorical or continuous. For example, if we consider the S&P 500 index, we can specify $\mathcal{V} = \{-1, 0, +1\}$ depending on its short trend. Let y_t be the value of the S&P 500 index and m_t the corresponding 10-day moving average. v_t is equal to -1 if the S&P 500 index exhibits a negative trend, 0 if it has no trend, and $+1$ otherwise[10]. An example of continuous labels is a vector composed of the last p values (Koshiyama *et al.*, 2019). This type of labels acts as a time memory and helps to reproduce auto-correlation patterns of the stochastic process.

2.3 Wasserstein GAN models

In Section 3, we will see that the basic GAN model using the cross-entropy as the loss function for the discriminator suffers from three main problems. First, the visualization of the training process is not obvious. Traditionally, we focus on the loss error curve to decide whether or not the network is trained correctly. In the case of financial applications, looking at binary cross entropy is not relevant. Computer vision algorithms can trust generated images to evaluate the GAN, financial time series are extremely noisy. Therefore, a visual evaluation is much more questionable. Second, the basic GAN model is not suitable for generating multi-dimensional time series. A possible alternative is to modify the GAN's structure. For instance, we can replace the binary cross entropy function by the Wasserstein distance for the training error (Arjovsky *et al.*, 2017a,b), or temporal information of multi-dimensional time series can be encoded using a more complex structure such as convolutional neural networks (Radford *et al.*, 2016) or recurrent neural networks (Hyland *et al.*, 2017). Third, the mode collapse phenomenon must be addressed in order to manage the poor diversity of generated samples, because we would like to have several simulated time series, as we have when performing Monte Carlo methods.

[10]For instance, we can define the labels in the following way: $v_t = \mathbb{1}\{\varepsilon_t > \epsilon\} - \mathbb{1}\{\varepsilon_t < -\epsilon\}$ where $\varepsilon_t = y_t - m_t$ and $\epsilon > 0$ is a threshold.

2.3.1 Optimization problem

Let $\mathbb{P}$ and $\mathbb{Q}$ be two univariate probability distributions. The Wasserstein (or Kantorovich) distance between $\mathbb{P}$ and $\mathbb{Q}$ is defined as:

$$W_p(\mathbb{P},\mathbb{Q}) = \left(\inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \int \|x-y\|^p \,\mathrm{d}\mathbb{F}(x,y)\right)^{1/p} \tag{16}$$

where $\mathcal{F}(\mathbb{P},\mathbb{Q})$ denotes the Fréchet class of all joint distributions $\mathbb{F}(x,y)$ whose marginals are equal to $\mathbb{P}$ and $\mathbb{Q}$. In the case $p=1$, the Wasserstein distance represents the cost of the optimal transport problem[11] (Villani, 2008):

$$W(\mathbb{P},\mathbb{Q}) = \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \mathbb{E}\left[\|X-Y\| \mid (X,Y)\sim\mathbb{F}\right] \tag{17}$$

because $\mathbb{F}(x,y)$ describes how many masses are needed in order to transport one distribution to another (Arjovsky *et al.*, 2017a,b). The Kantorovich-Rubinstein duality introduced by Villani (2008) allows us to rewrite Equation (17) as follows:

$$W(\mathbb{P},\mathbb{Q}) = \sup_{\varphi} \mathbb{E}\left[\varphi(X) \mid X\sim\mathbb{P}\right] - \mathbb{E}\left[\varphi(Y) \mid Y\sim\mathbb{Q}\right] \tag{18}$$

where φ is a 1-Lipschitz function, such that $|\varphi(x)-\varphi(y)| \leq \|x-y\|$ for all (x,y).

In the case of a generative model, we would like to check whether sample and generated data follows the same distribution. Therefore, we obtain $\mathbb{P}=\mathbb{P}_{\text{data}}$ and $\mathbb{Q}=\mathbb{P}_{\text{model}}$. In the special case of a GAN model, $\mathbb{P}_{\text{model}}(x;\theta_g)$ is given by $\boldsymbol{\mathcal{G}}(\mathbb{P}_{\text{noise}}(z);\theta_g)$. Arjovsky *et al.* (2017a,b) demonstrated that if $\boldsymbol{\mathcal{G}}(z;\theta_g)$ is continuous with respect to θ, then $W(\mathbb{P}_{\text{data}},\mathbb{P}_{\text{model}})$ is continuous everywhere and differentiable almost everywhere. This result implies that GAN can be trained until it has converged to its optimal solution contrary to basic GANs, where the training loss were bounded and not continuous[12]. Moreover, Gulrajani *et al.* (2017) adapted Equation (18) in order to recover the two player min-max game:

$$\mathcal{C}(\theta_g,\theta_d) = \mathbb{E}\left[\boldsymbol{\mathcal{D}}(x_1;\theta_d) \mid x_1\sim\mathbb{P}_{\text{data}}(x)\right] - \mathbb{E}\left[\boldsymbol{\mathcal{D}}(x_0;\theta_d) \mid x_0\sim\boldsymbol{\mathcal{G}}(z;\theta_g), z\sim\mathbb{P}_{\text{noise}}(z)\right]$$

The Wasserstein GAN (WGAN) plays the same min-max optimization problem as previously:

$$\hat{\theta}_g^{(\min)} = \arg\min_{\theta_g\in\Theta_{\boldsymbol{\mathcal{G}}}} \mathcal{C}^{(\min)}(\theta_g \mid \theta_d)$$

This implies that the discriminator needs to be a 1-Lipschitz function.

Remark 4. *The discriminator function is not necessarily a classifier, since the output of the function $\boldsymbol{\mathcal{D}}$ can take a value in $[0,1]$. It can then be a general scoring method.*

Remark 5. *The Kantorovich-Rubinstein duality makes the link between Wasserstein distance and integral probability metrics presented in Appendix A.3.2 on page 63.*

2.3.2 Properties of the optimal solution

In Wasserstein GAN models, the cost function becomes:

$$\mathcal{C}(\theta_g,\theta_d) = \mathbb{E}\left[\varphi(x_1;\theta_d) \mid x_1\sim\mathbb{P}_{\text{data}}\right] - \mathbb{E}\left[\varphi(x_0;\theta_d) \mid x_0\sim\boldsymbol{\mathcal{G}}(z;\theta_g), z\sim\mathbb{P}_{\text{noise}}(z)\right]$$

[11] See Appendix A.6 on page 68 for an introduction of Monge-Kantorovich problems, and the relationship between optimal transport and the Wasserstein distance.

[12] There is also no problem in computing the gradient.

and we have:

$$\nabla_{\theta_g} \mathcal{C}(\theta_g, \theta_d) = -\mathbb{E}\left[\nabla_{\theta_g} \varphi\left(\boldsymbol{\mathcal{G}}(z; \theta_g); \theta_d\right)\right]$$

We recall that the discriminator φ needs to satisfy the Lipschitz property otherwise the loss gradient will explode[13]. A first alternative proposed by Arjovsky *et al.* (2017a,b) is to clip the weights into a closed space $[-c, c]$. However, this method is limited because the gradient can vanish and weights can saturate. A second alternative proposed by Gulrajani *et al.* (2017) is to focus on the properties of the optimal discriminator gradient. They showed that the optimal solution $\varphi^\star$ has a gradient norm 1 almost everywhere under $\mathbb{P}$ and $\mathbb{Q}$. Therefore, they proposed to add a regularization term to the cost function in order to constraint the gradient norm to converge to 1. This gradient penalty leads to define a new cost function:

$$\mathcal{C}_{\text{regularized}}(\theta_g, \theta_d; \lambda) = \mathcal{C}(\theta_g, \theta_d) + \lambda \mathbb{E}\left[\left(\left\|\nabla_x \varphi(x_2; \theta_d)\right\|_2 - 1\right)^2 \mid x_2 \sim \mathbb{P}_{\text{mixture}}\right]$$

where $\mathbb{P}_{\text{mixture}}$ is a mixture distribution of $\mathbb{P}$ and $\mathbb{Q}$. More precisely, Gulrajani *et al.* (2017) proposed to sample x_2 as follows: $x_2 = \alpha x_1 + (1 - \alpha) x_0$ where $\alpha \sim \mathcal{U}_{[0,1]}$, $x_0 \sim \mathbb{Q}$ and $x_1 \sim \mathbb{P}$.

Remark 6. *Gulrajani et al. (2017) found that a good value of the coefficient λ is 10.*

2.4 Convolutional neural networks

A convolutional neural network (CNN) is a class of deep neural networks, where the inputs are transformed using convolution and filtering operators. For instance, this type of neural networks is extensively used in computer vision. Recently, Radford *et al.* (2016) implemented a new version of GANs using convolutional networks as generator and discriminator in order to improved image generation[14]. In this approach, the underlying idea is to extract pertinent features. In finance, inputs are different and correspond to n_x-dimensional time series representing asset prices over n_t days. Therefore, inputs are represented by a matrix belonging to $\mathcal{M}_{n_t,n_x}(\mathbb{R})$, and are characterized by the time axis, where order matters and the asset axis, where order doesn't matter. Therefore, the input transformation is referring to 1-dimensional convolution (1D-CNN).

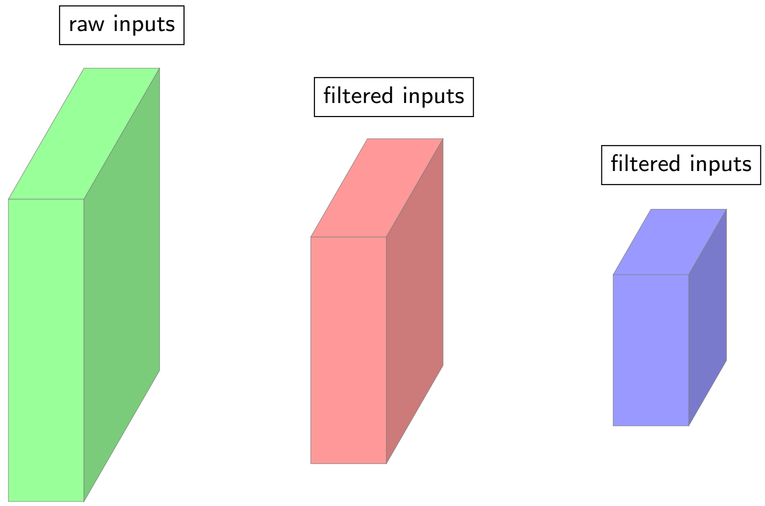
2.4.1 Extracting features using convolution

Convolution is no more than a linear transformation of a given input. But, contrary to simple multilayer perceptrons, convolutional operations preserve the notion of ordering according to a specific axis. To achieve this, a weight matrix of a given length n_k called kernel will slide across the input data. At each location, a continuous part of the input data is selected, and the kernel receives this vector as input in order to produce a single output by matrix product. This process is repeated with n_f different filters of similar dimension. Consequently, we obtained n_f down-sampled versions of the original input. The way the kernel can slide across the input data is controlled by the stride n_s. It is defined as the distance between two consecutive positions of the kernel (Dumoulin and Visin, 2016). The higher the stride value, the more important the sub-sampling. If s is equal to one, all the data are considered and there is no sub-sampling. Finally, the notion of padding allows us to address the situation when the kernel arrive at the end or the beginning of an axis. So, padding is defined as the number of zeros concatenated at the beginning or at the end of a given axis.

[13] As said previously, φ is not necessarily a discriminator function, but we continue to use this term to name φ.

[14] This class of CNNs is called deep convolutional generative adversarial networks (DCGANs).

Figure 5: Input architecture of convolutional neural networks



In finance, we want to down-sample a given multi-dimensional time series belonging to $\mathcal{M}_{n_t,n_x}(\mathbb{R})$ into a subspace $\mathcal{M}_{n_t',n_x}(\mathbb{R})$ with the condition that $n_t' < n_t$. With a 1-dimensional convolution, output time axis is automatically defined with respect to the dimension of the kernel. The first dimension of the kernel is free, whereas the second dimension is set to the number of time series n_x. Thus, the kernel is defined by a weight matrix belonging to $\mathcal{M}_{n_k,n_x}(\mathbb{R})$. In order to sample the input data, the stride is chosen, such that $s \geq 1$. The padding is set, such that n_p rows of zeros is padded at the (bottom and top) limits of the input data. Finally, the output will belong to $\mathcal{M}_{n_t',n_x}(\mathbb{R})$, such that:

$$n_t' = \frac{n_t - n_k + 2n_p}{n_s} + 1 \tag{19}$$

This type of convolution will be used to build the discriminator that should output a single scalar. In this case, down-sampling real or fake time series become essential.

2.4.2 Up-sampling a feature using transpose convolution

This transformation also called '*deconvolution*' is useful to build the generator when we would like to up-sample a random noise in order to produce a fake time series belonging to $\mathcal{M}_{n_t,n_x}(\mathbb{R})$. More generally, it could be used as the decoding layer in all forms of auto-encoders. Transpose convolution is the exact inverse transformation of the convolution previously defined. Considering an input data belonging to $\mathcal{M}_{n_t,n_x}(\mathbb{R})$, we obtain an output data belonging to $\mathcal{M}_{n_t',n_x}(\mathbb{R})$ such that:

$$n_t' = n_s(n_t - 1) + n_k - 2n_p \tag{20}$$

The term transpose comes from the fact that convolution is in fact a matrix operation. When we perform a convolution, input matrix flattens into a $n_t n_x$-dimensional vector. All the parameters (stride, kernel settings or padding) are encoded into the weight convolution matrix belonging to $\mathcal{M}_{n_t n_x, n_t'}(\mathbb{R})$. Therefore, the output data is obtained by computing the product between the input vector and the convolution matrix. Performing a transpose convolution is equivalent of taking the transpose of this convolution matrix.

3 Financial applications

3.1 Application of RBMs to generate synthetic financial time series

A typical financial time series of length T may be described by a real-valued vector $y = (y_1, ..., y_T)$. As we have introduced in Sections 2.1.1 and 2.1.2 on page 4, Bernoulli RBMs take binary vectors as input for training and Gaussian RBMs take the data with unit variance as input in order to ignore the parameter σ. Therefore, data preprocessing is necessary before any training process of RBMs. For a Bernoulli RBM, each value of y_t needs to be converted to an N-digit binary vector using the algorithm proposed by Kondratyev and Schwarz (2019). The underlying idea consists in discretizing the distribution of the training dataset and representing them with binary numbers. For instance, N may be set to 16 and a Bernoulli RBM, which takes a one-dimensional time series as training dataset, will have 16 visible layer units. In the same way, the visible layer will have $16 \times n$ neurons in the case of an n-dimensional time series. Moreover, samples generated by a Bernoulli RBM after Gibbs sampling are also binary vectors and we need another algorithm to transform binary vectors into real values. These transformation algorithms between real values and binary vectors are detailed in Appendix A.7 on page 72. For a Gaussian RBM, we need only to normalize data to unit variance and scale generated samples after Gibbs sampling.

For the training process of RBMs, we use the contrastive divergence algorithm $\mathrm{CD}^{(1)}$ to estimate the log-likelihood gradient and the learning rate $\eta^{(t)}$ is set to 0.01. All models are trained using mini-batch gradient descent with batch size 500 and we apply 100 000 epochs to ensure that models are well-trained. Using a larger mini-batch size will get a more stable gradient estimate at each step, but it will also use more memory and take a longer time to train the model.

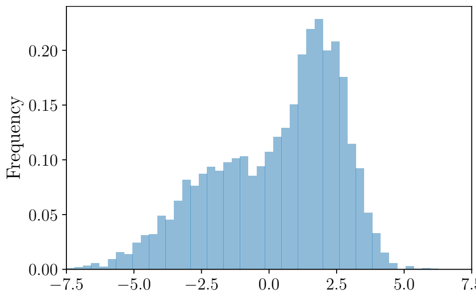
After having trained the RBMs, we may use these models to generate synthetic (or simulated) samples to match training samples. Theoretically, a well-trained RBM should be able to transform random noise into samples that have the same distribution as the training dataset after doing Gibbs sampling for a sufficiently long time. Therefore, the trained RBM is fed by samples drawn from the Gaussian distribution $\mathcal{N}(0, 1)$ as input data. We then perform a large number of forward and backward passes through the network to ensure convergence between the probability distribution of generated samples and the probability distribution of the training sample. In the particular case of Bernoulli RBMs, after the last backward pass from the hidden layer to the visible layer, we need to transform generated binary vectors into real-valued vectors.

3.1.1 Simulating multi-dimensional datasets

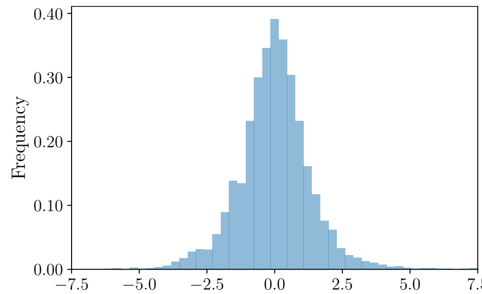
In this first study, we test Bernoulli and Gaussian RBMs on a simulated multi-dimensional dataset to check whether RBMs can learn the joint distribution of training samples that involves marginal distributions and a correlation structure. We simulate 10 000 observations of 4-dimensional data with different marginal distributions, as shown in Figure 6. The first dimension consists of samples drawn from a Gaussian mixture model[15]. The samples in the second dimension are drawn from a Student's t distribution with 4 degrees of freedom. For the third and fourth dimensions, we draw respectively samples from the Gaussian distribution $\mathcal{N}(0, 2)$. The empirical probability distributions of these 4-dimensional data are reported in Figure 6. In addition, we use a Gaussian copula to construct a simple correlation structure of the simulated samples with the correlation matrix of the Gaussian copula

[15]The mixture probability is equal to 50%, whereas the two underlying probability distributions correspond to two Gaussian random variables $\mathcal{N}(-1.5, 2)$ and $\mathcal{N}(2, 1)$.

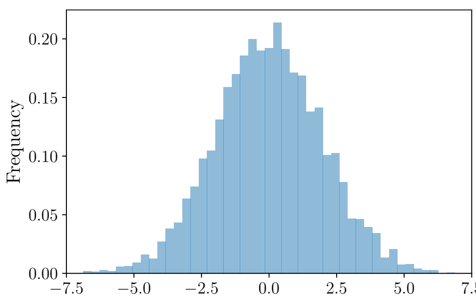
Figure 6: Histograms of simulated training samples



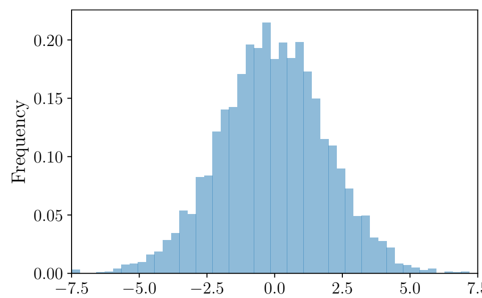
(a) Gaussian mixture model



(b) Student's t-distribution with $\nu = 4$

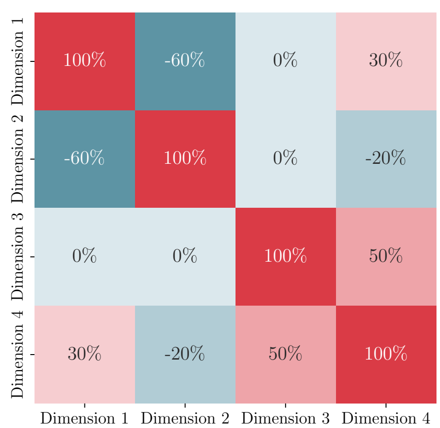


(c) Gaussian distribution $\mathcal{N}(0, 2)$



(d) Gaussian distribution $\mathcal{N}(0, 2)$

Figure 7: The theoretical correlation matrix of the Gaussian copula



given in Figure 7. We set a strong negative correlation -60% between the Gaussian mixture model and the Student's t distribution and a strong positive correlation 50% between the two Gaussian distributions. Moreover, the Gaussian distribution in the fourth dimension has a more complicated correlation structure[16] than the Gaussian distribution in the third dimension, which is independent from the Gaussian mixture model and the Student's t distribution or the first and second dimensions. Here, the challenge is to learn both the marginal distribution and the copula function.

[16]The correlations are equal to 30% and -20% with respect to the first and second dimensions.

Before implementing the training process, we need to update the parameters for the RBMs to adjust multi-dimensional input. In the case of the Bernoulli RBM, each value should be transformed into a 16-digit binary vector and we need to concatenate them to form a 64-digit binary vector. So, the visible layer of the Bernoulli RBM has 64 neurons. For the Gaussian RBM, we have simply 4 visible layer units. According to our experience, we need to set a large number of hidden layer units in order to learn marginal distributions and the correlation structure at the same time. Therefore, we choose 256 hidden layer units for the Bernoulli RBM and 128 hidden layer units for the Gaussian RBM. We also recall that the number of epochs is set to 100 000 in order to ensure that models are well-trained. After the training process, we perform 1 000 steps of the Gibbs sampling on a 4-dimensional random noise to generate 10 000 samples with the same size as the training dataset. These simulated samples are expected to have not only the same marginal distributions but also the same correlation structure as the training dataset.

Bernoulli RBM Figures 8 and 9 compare the histograms and QQ-plots between training samples and generated samples after 1 000 Gibbs sampling steps in the case of the Bernoulli RBM. According to these figures, we observe that the Bernoulli RBM can learn pretty well each marginal distribution of training samples. However, we also find that the Bernoulli RBM focuses on extreme values in the training dataset instead of the whole tail of the distribution and this phenomenon is more evident for heavy-tailed distributions. For example, in the case of the Student's t distribution (Panel (b) in Figure 9), the Bernoulli RBM tries to learn the behavior of extreme values, but ignores the other part of the distribution tails. Comparing the results for the two Gaussian distributions (Panels (c) and (d) in Figure 9), we notice that the learning of the Gaussian distribution in the fourth dimension, which has a more complicated correlation structure with the other dimensions, is not as good as the result for the Gaussian distribution in the third dimension.

We have replicated the previous simulation 50 times[17] and we compare the average of the mean, the standard deviation, the 1^{st} percentile and the 99^{th} percentile between training and simulated samples in Table 1. Moreover, in the case of simulated samples, we indicate the confidence interval by $\pm$ one standard deviation. These statistics demonstrate that the Bernoulli RBM overestimates the value of standard deviation, the 1^{st} percentile and the 99^{th} percentile, especially in the case of Student's t distribution. This means that the Bernoulli RBM is sensitive to extreme values and the learned probability distribution is more heavy-tailed than the empirical probability distribution of the training dataset.

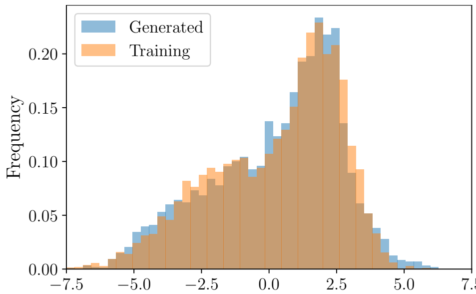
In Figure 10, we show the comparison between the empirical correlation matrix[18] of the training dataset and the average of the correlation matrices computed by the 50 Monte Carlo replications. We notice that the Bernoulli RBM does not capture perfectly the correlation structure since the correlation coefficient values are less significant comparing with the empirical correlation matrix. For instance, the correlation between the first and second dimensions is equal to -57% for the training data, but only -31% for the simulated data on average[19].

[17] Each replication corresponds to 10 000 generated samples of the four random variables, and differs because they use different random noise series as input data.

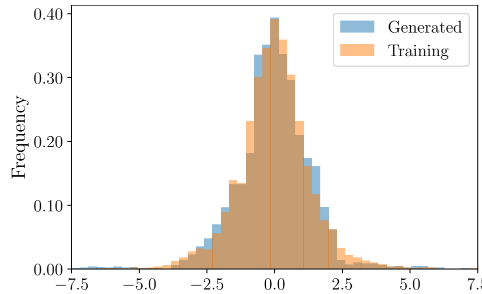
[18] This empirical correlation matrix is not exactly equal to the correlation matrix of the copula function, because the marginals are not all Gaussian.

[19] Similarly, the correlation between third and fourth dimensions is equal to 50% for the training data, but only 31% for the simulated data on average.

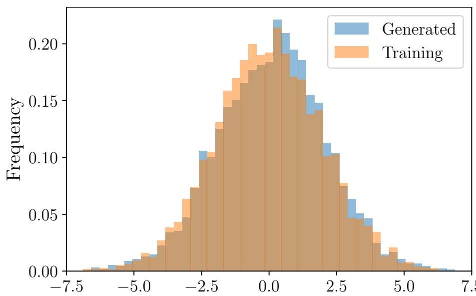
Figure 8: Histogram of training and Bernoulli RBM simulated samples



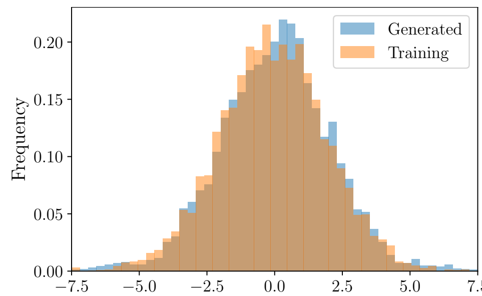
(a) Gaussian mixture model



(b) Student's t-distribution with $\nu = 4$

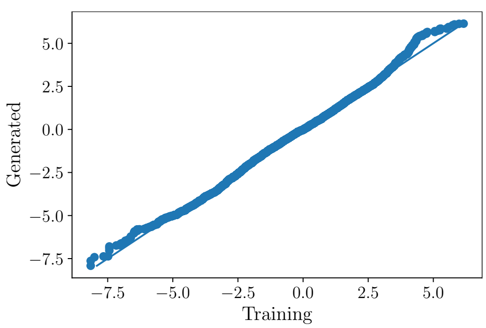


(c) Gaussian distribution $\mathcal{N}(0, 2)$

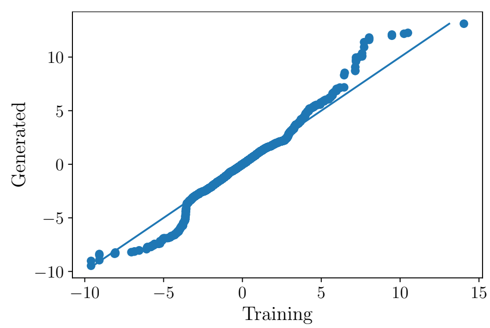


(d) Gaussian distribution $\mathcal{N}(0, 2)$

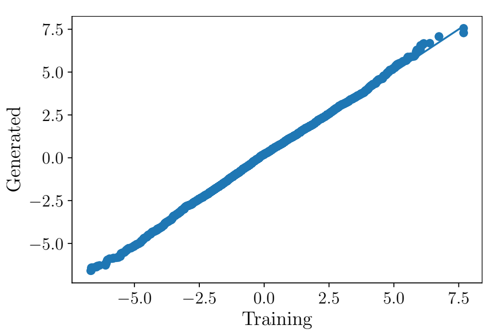
Figure 9: QQ-plot of training and Bernoulli RBM simulated samples



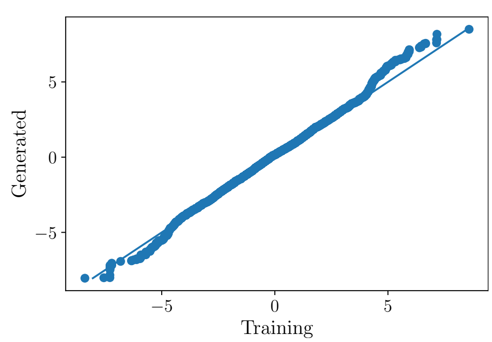
(a) Gaussian mixture model



(b) Student's t-distribution with $\nu = 4$



(c) Gaussian distribution $\mathcal{N}(0, 2)$

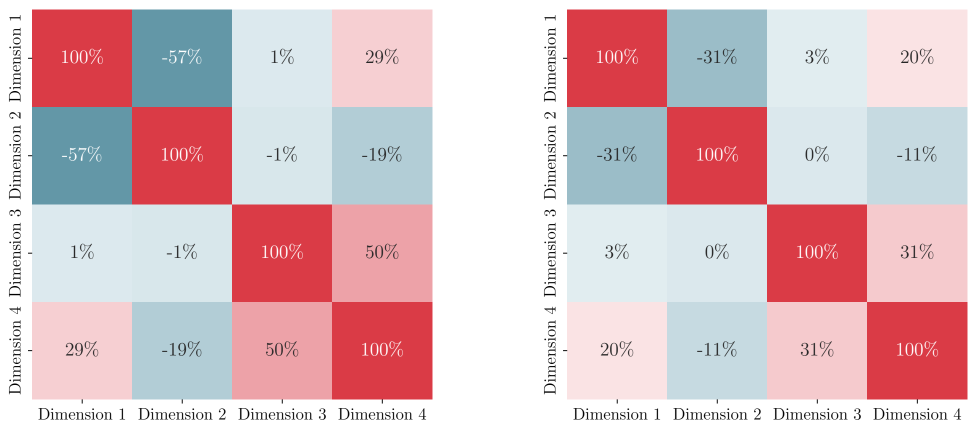


(d) Gaussian distribution $\mathcal{N}(0, 2)$

Table 1: Comparison between training and Bernoulli RBM simulated samples

Statistic	Dimension 1		Dimension 2	
	Training	Simulated	Training	Simulated
Mean	0.303	0.319 (± 0.039)	−0.006	−0.054 (± 0.029)
Standard deviation	2.336	2.385 (± 0.124)	1.409	1.540 (± 0.044)
1^{st} percentile	−5.512	−5.380 (± 0.121)	−3.590	−5.008 (± 0.905)
99^{th} percentile	4.071	4.651 (± 0.143)	3.862	4.255 (± 0.374)
Statistic	Dimension 3		Dimension 4	
	Training	Simulated	Training	Simulated
Mean	−0.002	0.071 (± 0.039)	−0.063	0.033 (± 0.043)
Standard deviation	1.988	2.044 (± 0.028)	1.982	2.037 (± 0.030)
1^{st} percentile	−4.691	−4.987 (± 0.167)	−4.895	−5.227 (± 0.244)
99^{th} percentile	4.677	4.918 (± 0.190)	4.431	4.938 (± 0.322)

Figure 10: Comparison between the empirical correlation matrix and the average correlation matrix of Bernoulli RBM simulated data



Training samples

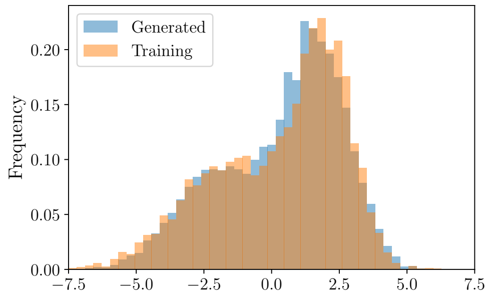
Generated samples

Gaussian RBM Let us now consider a Gaussian RBM with 4 visible layer units and 128 hidden layer units using the same simulated training dataset. After the training process, synthetic samples are always generated using the Gibbs sampling with 1 000 steps between visible and hidden layers. Histograms and QQ-plots of training and generated samples are reported in Figures 11 and 12. We notice that the Gaussian RBM works well for the Gaussian mixture model and the two Gaussian distributions (Panels (a), (c) and (d) in Figure 12). Again, we observe that the Gaussian distribution with the simplest correlation structure is easier to learn than the Gaussian distribution with a more complicated correlation structure. However, the Gaussian RBM fails to learn heavy-tailed distributions such as the Student's t distribution since the model tends to ignore all values in the distribution tails.

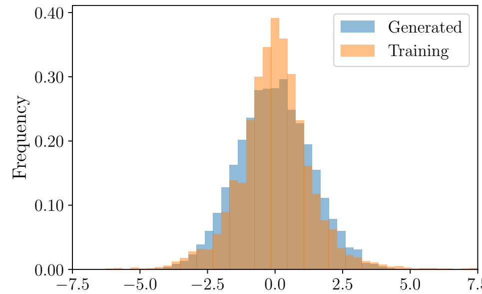
Comparing Tables 1 on page 24 for the Bernoulli RBM and Tables 2 on page 26 for the Gaussian RBM, we notice that the Gaussian RBM generally underestimates the value of the standard deviation, the 1^{st} percentile and the 99^{th} percentile. This means that this is challenging to generate leptokurtic probability distributions with Gaussian RBMs.

In Figure 13, we also compare the empirical correlation matrix of the training dataset

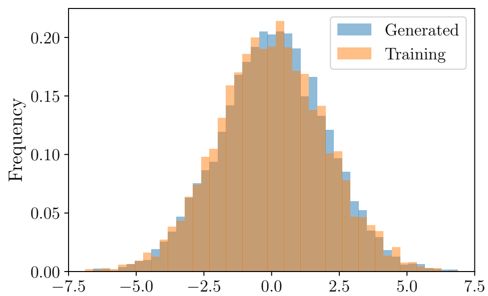
Figure 11: Histogram of training and Gaussian RBM simulated samples



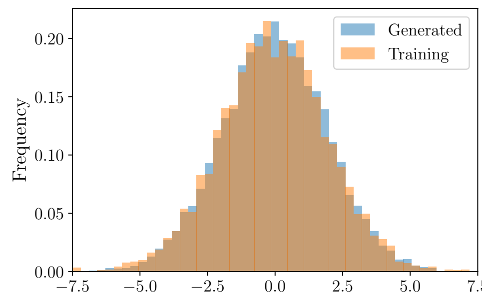
(a) Gaussian mixture model



(b) Student's t distribution with $\nu = 4$

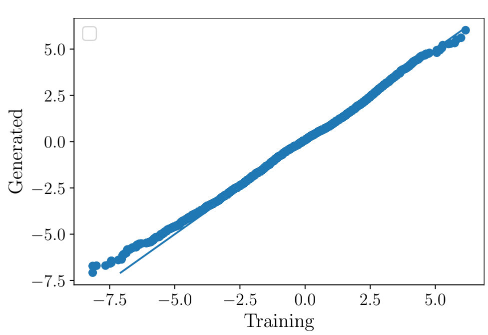


(c) Gaussian distribution $\mathcal{N}(0, 2)$

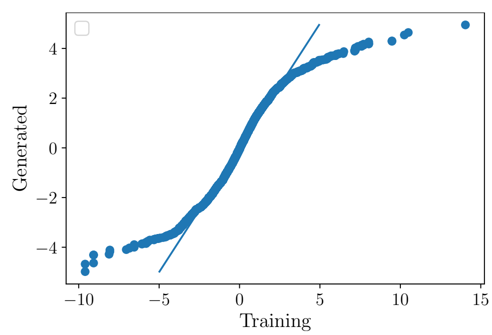


(d) Gaussian distribution $\mathcal{N}(0, 2)$

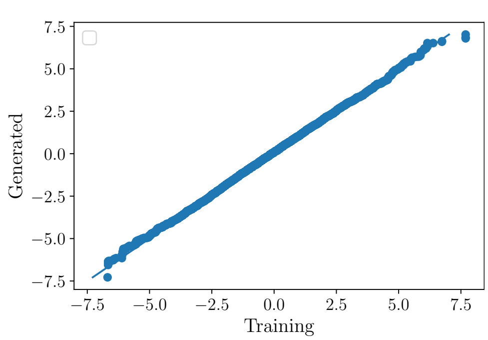
Figure 12: QQ-plot of training and Gaussian RBM simulated samples



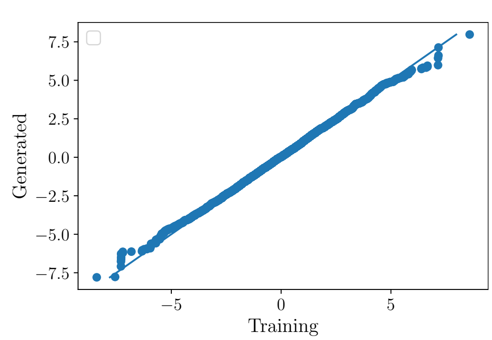
(a) Gaussian mixture model



(b) Student's t-distribution with $\nu = 4$



(c) Gaussian distribution $\mathcal{N}(0, 2)$



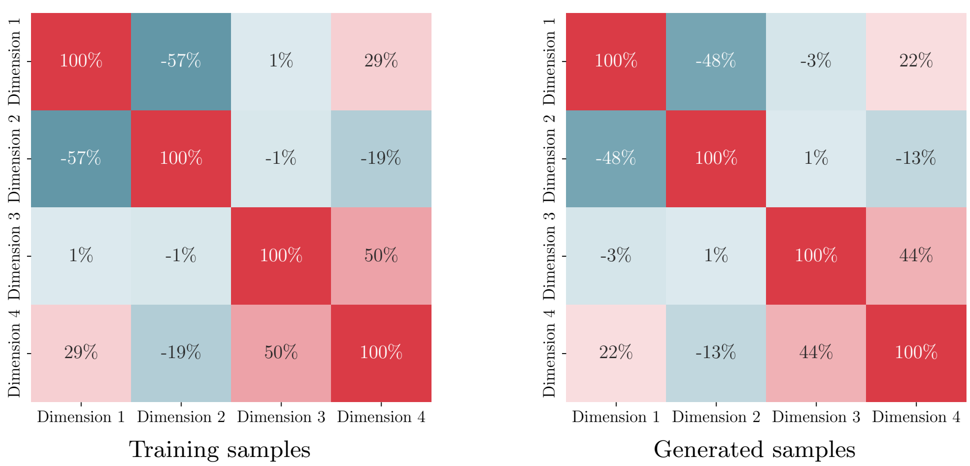
(d) Gaussian distribution $\mathcal{N}(0, 2)$

Table 2: Comparison between training and Gaussian RBM simulated samples

Statistic	Dimension 1		Dimension 2	
	Training	Simulated	Training	Simulated
Mean	0.303	0.325 (± 0.037)	−0.006	−0.006 (± 0.023)
Standard deviation	2.336	2.232 (± 0.021)	1.409	1.355 (± 0.020)
1^{st} percentile	−5.512	−4.992 (± 0.121)	−3.590	−3.054 (± 0.078)
99^{th} percentile	4.071	4.167 (± 0.078)	3.862	3.240 (± 0.105)
Statistic	**Dimension 3**		**Dimension 4**	
	Training	Simulated	Training	Simulated
Mean	−0.002	0.064 (± 0.029)	−0.063	0.046 (± 0.031)
Standard deviation	1.988	1.942 (± 0.023)	1.982	1.910 (± 0.026)
1^{st} percentile	−4.691	−4.382 (± 0.109)	−4.895	−4.512 (± 0.122)
99^{th} percentile	4.677	4.572 (± 0.109)	4.431	4.303 (± 0.139)

and the average of the correlation matrices computed with 50 Monte Carlo replications. Compared to the Bernoulli RBM, the Gaussian RBM captures much better the correlation structure of the training dataset. This is particular true for the largest correlation values. For instance, the correlation between first and second dimensions is equal to −57% for the training data, −31% for the Bernoulli RBM simulated data and −48% for the Gaussian RBM simulated data.

Figure 13: Comparison between the empirical correlation matrix and the average correlation matrix of Gaussian RBM simulated data



Summary of the results According to our tests, we consider that both Bernoulli and Gaussian well-trained RBMs can transform random noise series to the joint distribution of a training dataset. But they have their own characteristics:

- A Bernoulli RBM is very sensitive to extreme values of the training dataset and has a tendency to overestimate the tail of distribution. On the contrary, a Gaussian RBM has a tendency to underestimate the tails of probability distribution and faces difficulties in learning leptokurtic probability distributions.

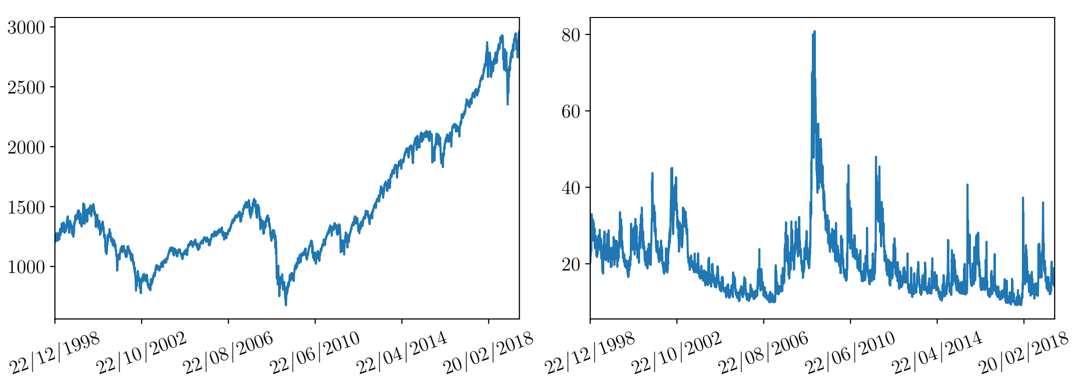
- Gaussian RBMs may capture more accurately the correlation structure of the training dataset than Bernoulli RBMs.

In practice, we may apply the normal score transformation in order to transform the training dataset, such that each marginal has a standard normal distribution. For that, we rank the values of each dimension from the lowest to the highest, map these ranks to a uniform distribution and apply the inverse Gaussian cumulative distribution function. We then train the RBM with these transformed values. After the training process, the marginals of samples generated by well-trained RBMs will follow a standard normal distribution. By using the inverse transformation, we can generate synthetic samples with the same marginal distributions than those of the training dataset, and a correlation structure that is closed to the empirical correlation matrix of the training data. In this case, we may consider Gaussian RBMs as an alternate method of the bootstrap sampling approach.

3.1.2 Application to financial time series

According to the results in the previous section, Gaussian RBMs capture more accurately the correlation structure of the training dataset and we can avoid its drawbacks by applying the normal score transformation. In this paragraph, we test and compare a Gaussian RBM and a conditional RBM on a multi-dimensional autocorrelated time series in order to check if conditional RBMs may capture at the same time the correlation structure and the time dependence of the training dataset. The RBM is trained on a real financial dataset consisting of 5 354 historical daily returns of the S&P 500 index and the VIX index from December 1998 to May 2018 (see Figure 14). During this period, the S&P 500 index and the VIX index have a remarkable negative correlation that is equal to -71%. In order to reinforce the autocorrelation level in the training samples, we apply a 3-day exponential weighted moving average approach to historical prices before calculating daily returns. Figure 15 shows the autocorrelation of training data. As both daily returns of the S&P 500 and VIX indices are leptokurtic and have some extreme values, we need to apply the normal score transformation on input data before the training process.

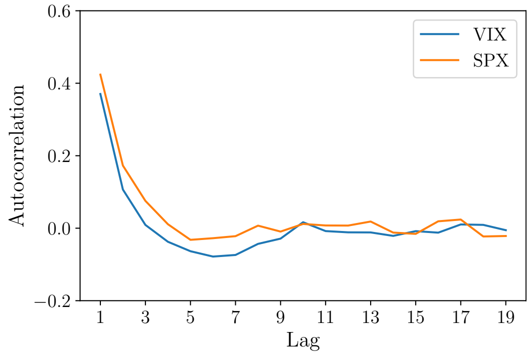
Figure 14: Historical prices of the S&P 500 and VIX indices



S&P 500 index VIX index

Normal score transformation In order to verify the learning quality of Gaussian RBMs using the normal score transformation, we run 50 Monte Carlo simulations after the training process and for each simulation, we generate 5 354 simulated observations from different

Figure 15: Autocorrelation function of the training dataset



random noise series. We then calculate the statistics of these samples as we have done in the previous section. According to Table 3, we find that although the Gaussian RBM still underestimates a little the standard deviation, the 1st percentile and the 99th percentile, the learning quality is much more improved. In addition, the average correlation coefficient between daily returns of S&P 500 and VIX indices over 50 Monte Carlo simulations is equal to −76%, which is very closed to the figure 71% of the empirical correlation. Therefore, we consider that training Gaussian RBMs with the normal score transformation may be used as an alternate method of bootstrap sampling.

Table 3: Comparison between the training sample and Gaussian RBM simulated samples using the normal score transformation

Statistic	S&P 500 index		VIX index	
	Training	Simulated	Training	Simulated
Mean	0.02%	−0.01% (± 0.01%)	0.06%	0.09% (± 0.06%)
Standard deviation	0.64%	0.62% (± 0.01%)	3.84%	3.65% (± 0.12%)
1st percentile	−1.93%	−1.88% (± 0.12%)	−7.31%	−6.97% (± 0.30%)
99th percentile	1.61%	1.49% (± 0.06%)	11.51%	10.96% (± 0.63%)

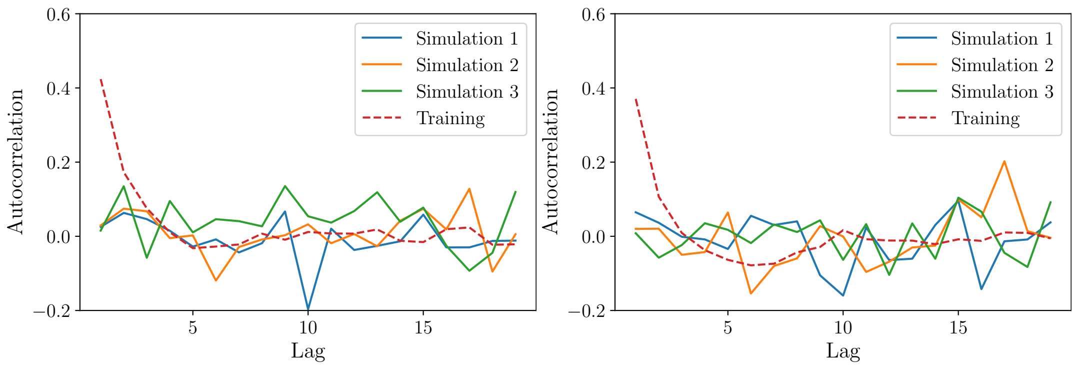
Learning time dependence As conditional RBM is an extension of Gaussian RBM, applying the normal score transformation should still work. Moreover, conditional RBM should be able to learn the time-dependent relationship by its design. So, we train a conditional RBM on the same training dataset in order to check whether this model can capture the autocorrelation patterns previously given in Figure 15.

In practice, we choose to use a long memory for the conditional layer in the conditional RBM and the values of the last 20 days will be fed to the model. Therefore, the conditional RBM has 2 visible layer units, 128 hidden layer units and 40 conditional layer units[20]. We then use the normal score transformation in order to transform the training dataset as mentioned before. After having done a training process of 100 000 epochs, we are interested in generating samples of consecutive time steps to verify if the conditional RBM can learn the joint distribution but also capture the autocorrelation of the training dataset. We run 3 Monte Carlo simulations and for each simulation, we generate consecutively samples of 250 observations. In Figures 16 and 17, we compare the autocorrelation function of generated

[20] Corresponding to two time series of 20 lags.

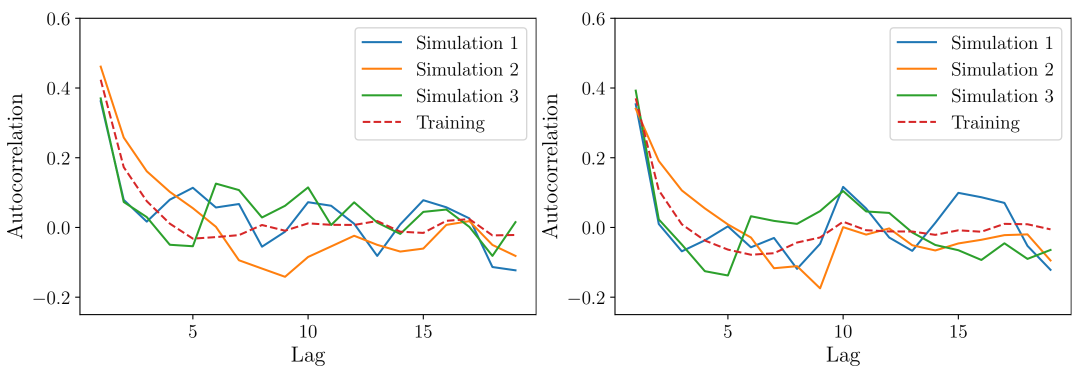
samples by Gaussian and conditional RBMs. These results show clearly that a conditional RBM can capture well the autocorrelation of the training dataset, which is not the case of a traditional Gaussian RBM.

Figure 16: Autocorrelation function of synthetic samples generated by a Gaussian RBM



S&P 500 index VIX index

Figure 17: Autocorrelation function of synthetic samples generated by a conditional RBM

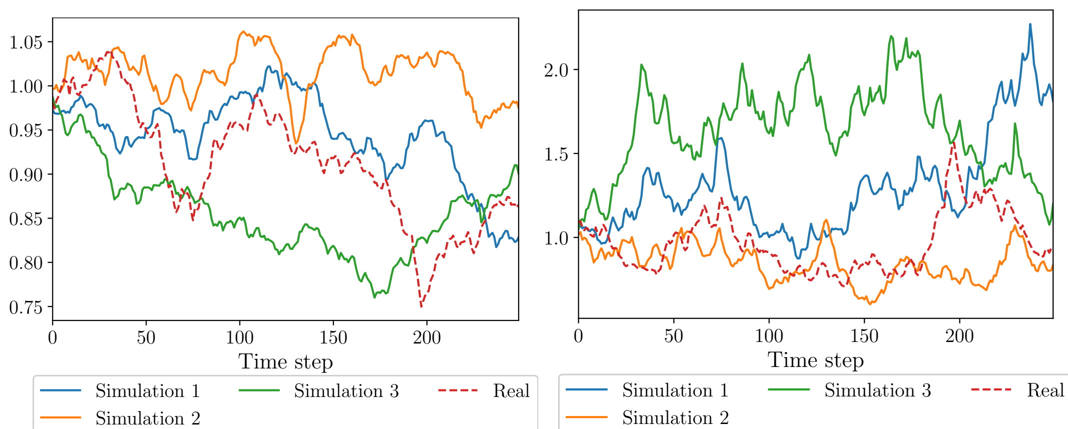


S&P 500 index VIX index

Market generator Based on above results, we consider that conditional RBMs can be used as a market generator (Kondratyev and Schwarz, 2019). In the example of S&P 500 and VIX indices, we have trained the models with all historical observations and for each date, we have used the values of the last 20 days as a memory vector for the conditional layer. In this approach, the normal score transformation ensures the learning of marginal distributions, whereas the conditional RBM ensures the learning of the correlation structure and the time dependence. After having calibrated the training process, we may choose a date as the starting point. The values of this date and its last 20 days are then passed to the trained RBM model and, after sufficient steps of Gibbs sampling, we get a sample for the next day. We then feed this new sample and the updated memory vector to the model in order to generate a sample for the following day. Iteratively, the conditional RBM can generate a multi-dimensional time series that has the same patterns as those of the training data in terms of marginal distribution, correlation structure and time dependence.

For instance, Figure 18 shows three time series of 250 trading days starting at the date 22/11/2000, which are generated by the trained conditional RBM. We notice that these three simulated time series have the same characteristics as the historical prices represented by the dashed line. The negative correlation between S&P 500 and VIX indices is also learned and we can see clearly that there exists a positive autocorrelation in each simulated one-dimensional time series.

Figure 18: Comparison between scaling historical prices of S&P 500 and VIX indices and synthetic time series generated by the conditional RBM



S&P 500 index VIX index

3.2 Application of GANs to generate fake financial time series

In this section, we perform the same tests for GAN models as those we have done for RBMs. The objective is to compare the results between these two types of generative models on the same training dataset. We recall that the first test consists of learning the joint distribution of a simulated multi-dimensional dataset with different marginal distributions and a simple correlation structure driven by a Gaussian copula. In this part, we test a Wasserstein GAN model with only simple dense layers in the generator and in the discriminator. In the second test, we train a conditional Wasserstein GAN model on the historical daily returns of the S&P 500 index and the VIX index from December 1998 to May 2018. As we have done in the previous tests for RBMs, we also apply a three-day exponential weighted moving average approach to historical prices before calculating daily returns. We know that this transformation will reinforce the autocorrelation level in the training samples. As a result, we need to choose a more complex structure for the generator and the discriminator to capture the complex features in the training dataset. For instance, we construct the generator and the discriminator with convolutional neural network as introduced in Section 2.4 on page 18. According to the results of these two tests, we check whether the family of GAN models may also learn the marginal distribution, the correlation structure and the time dependence of the training dataset as RBMs can do.

3.2.1 Simulating multi-dimensional datasets

As we have introduced in Section 2.3 on page 16, Wasserstein GAN models, which use the Wasserstein distance as the loss function, have several advantages comparing the basic traditional GAN models using the cross-entropy loss measure. Therefore, we test in this first study Wasserstein GAN models on the simulated multi-dimensional dataset as we have done for Bernoulli and Gaussian RBMs. We recall that we simulate 10 000 samples of 4-dimensional data with different marginal distributions (Gaussian mixture model, Student's t distribution and two Gaussian distributions), as shown in Figure 6 on page 21 and the training dataset has a correlation structure simulated by a Gaussian copula with the correlation matrix given in Figure 7 on page 21. Here, the objective of the test is to check whether a well-trained Wasserstein GAN models can learn the marginal distribution and the copula function of training samples.

Data preprocessing In the practice of GAN models, we need to match the dimension and the range of the output of the generator and the input of discriminator. To address this issue, there are two possible ways:

- We can use a complex activation function for the last layer of the generator in order to ensure that the generated samples have the same range as the training samples for the discriminator (Clevert *et al.*, 2015).
- We can modify the range of the training samples by applying a preprocessing function and choose a usual activation function (Wiese *et al.*, 2020).

In our study, we consider the second approach by applying the MinMax scaling function[21] to the training samples for the discriminator. As a result, the input data scaled for the discriminator will take their values in $[0, 1]$ and we may choose the sigmoid activation function for the last layer of the generator to ensure that the outputs of the generator have theirs values in $[0, 1]$. In addition, we need to simulate the random noise as the inputs for the generator. As the random noise will play the role of latent variables, we have the freedom to choose its distribution. For instance, we use the Gaussian distribution $z \sim \mathcal{N}(0, 1)$ in this study.

Structure and training of Wasserstein GAN models As we have mentioned, we need to match the output of the generator and the input of the discriminator. Therefore, the last layer in the generator should have 4 neurons for our simulated training dataset and we may use sigmoid activation function for each neuron as we have applied MinMax scaling function to the training samples. The discriminator corresponds to a traditional classification problem, so its last layer should have only one neuron. We may choose the sigmoid activation function for this neuron if the training samples are labelled by $\{0, 1\}$ or the tangent hyperbolic activation function if the training samples are labelled by $\{-1, 1\}$.

In our study, we want to construct a Wasserstein GAN model, which can convert a 100-dimensional random noise vector to the data with the same joint distribution as the training dataset. According to our tests, a simple structure as multi-layer perceptrons for the generator and the discriminator is sufficient for learning the distribution of these simulated training samples. More precisely, the generator is fed by 100-dimensional vectors and has 5 layers with the structure $\mathcal{S}_{\mathcal{G}} = \{200, 100, 50, 25, 4\}$ where $s_i \in \mathcal{S}_{\mathcal{G}}$ represents the numbers of neurons of the i^{th} layer. The discriminator has 4 layers with $\mathcal{S}_{\mathcal{D}} = \{100, 50, 10, 1\}$ and takes 4-dimensional vectors as input data. As explained in the previous paragraph, we may choose the sigmoid activation function for the last layer of the generator and the tangent hyperbolic activation function for the last layer of the discriminator. For all other layers

[21] We have $f(x) = (x - \min x) / (\max x - \min x)$.

of the discriminator and the generator, the activation function $f(x)$ corresponds to a leaky rectified linear unit (or RELU) function with $\alpha = 0.5$:

$$f_{\alpha}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

According to Arjovsky *et al.* (2017a), the Wasserstein GAN model should be trained with the RMSProp optimizer using a small learning rate. Therefore, we set the learning rate to $\eta = 1.10^{-4}$ and use mini-batch gradient descent with batch size 500. Table 4 summarizes the setting of the Wasserstein GAN model implementation using Python and the TensorFlow library. After the training process, we generate 10 000 samples in order to compare the marginal distributions and the correlation structure with the training dataset.

Table 4: The setting of the Wasserstein GAN model

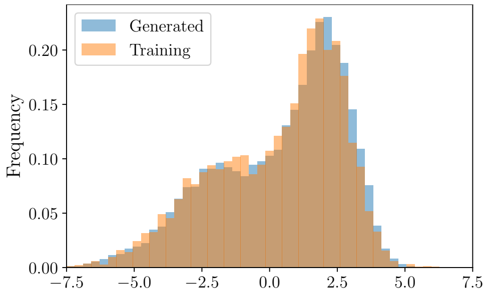
Training data dimension	4
Input feature scaling function	MinMax
Random noise vector dimension	100
Random noise distribution	$\mathcal{N}(0,1)$
Generator structure	$\mathcal{S}_{\mathcal{G}} = \{200, 100, 50, 25, 4\}$
Discriminator structure	$\mathcal{S}_{\mathcal{D}} = \{100, 50, 10, 1\}$
Loss function	Wasserstein distance
Learning optimizer	RMSProp
Learning rate	1.10^{-4}
Batch size	500

Results Figures 19 and 20 compare the histograms and QQ-plots between training samples and generated samples. According to these figures, we observe that the Wasserstein GAN model can learn very well each marginal distribution of training samples, even better than Bernoulli and Gaussian RBMs. In particular, the Wasserstein GAN model fits more accuracy for heavy-tailed distributions than RBMs as in the case of the Student's t distribution (Panel (b) in Figure 20).

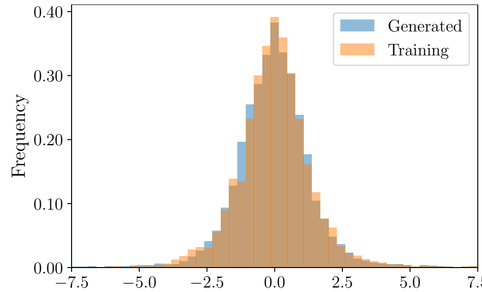
As we have done for RBMs, we replicate the previous simulation 50 times and we compare the average of the mean, the standard deviation, the 1^{st} percentile and the 99^{th} percentile between training and simulated samples in Table 5. Moreover, in the case of simulated samples, we indicate the confidence interval by $\pm$ one standard deviation. These statistics show that the Wasserstein GAN model estimates very well the value of standard deviation, the 1^{st} percentile and the 99^{th} percentile for each marginal distribution, especially for Gaussian mixture model and Student's t distribution in the first and second dimensions. One shortcoming is that the Wasserstein GAN model slightly overestimates the value of the 99^{th} percentile for the two Gaussian distributions. Comparing with the results shown in Table 1 on page 24 for the Bernoulli RBM and Table 2 on page 26 for the Gaussian RBM, we notice that the Wasserstein GAN model learns better the marginal distributions of the training dataset and doesn't have tendency to always underestimate or overestimate the tails of probability distribution as RBMs. However, there exists a small bias between the empirical mean computed with 50 Monte Carlo replications and the mean of the training dataset.

In Figure 21, we also compare the empirical correlation matrix of the training dataset and the average of the correlation matrices computed with 50 Monte Carlo replications. We notice that the Wasserstein GAN model captures very well the correlation structure.

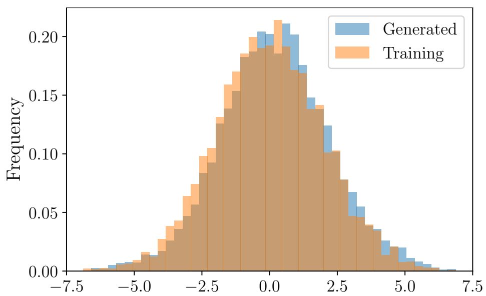
Figure 19: Histogram of training and WGAN simulated samples



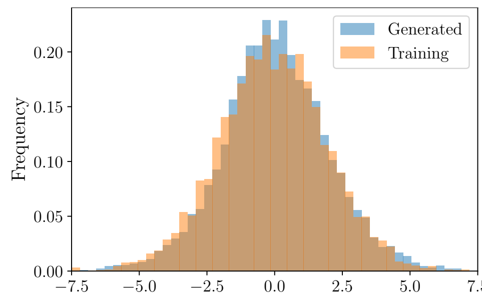
(a) Gaussian mixture model



(b) Student's t-distribution with $\nu = 4$

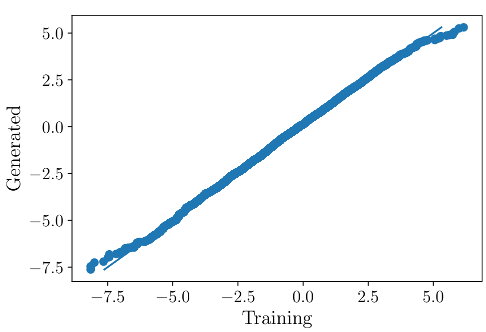


(c) Gaussian distribution $\mathcal{N}(0, 2)$

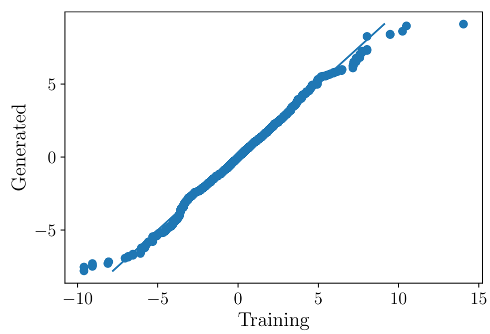


(d) Gaussian distribution $\mathcal{N}(0, 2)$

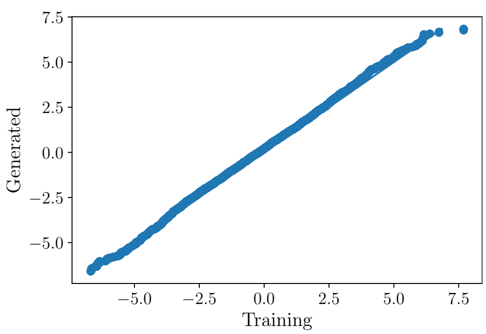
Figure 20: QQ-plot of training and WGAN simulated samples



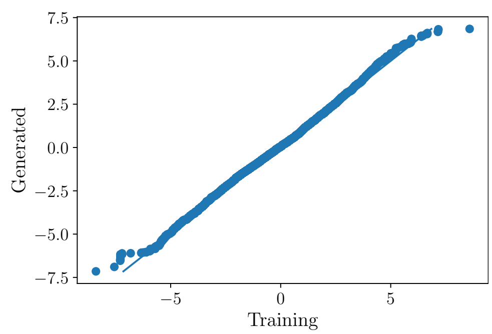
(a) Gaussian mixture model



(b) Student's t-distribution with $\nu = 4$



(c) Gaussian distribution $\mathcal{N}(0, 2)$



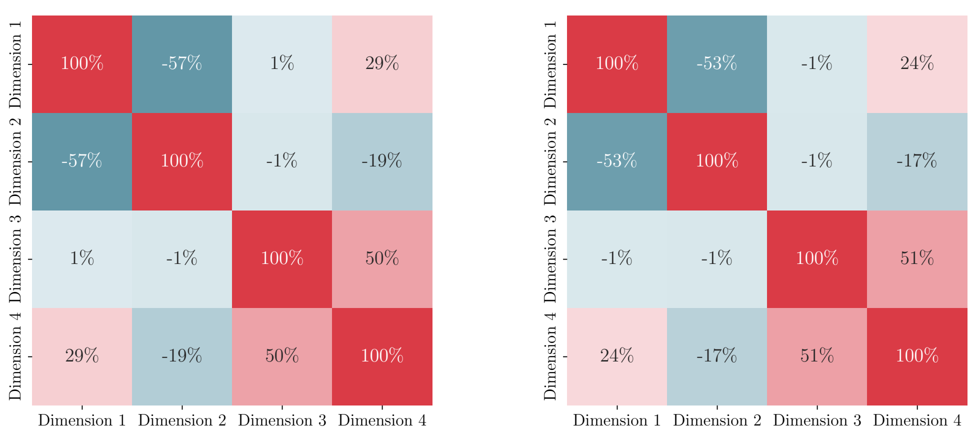
(d) Gaussian distribution $\mathcal{N}(0, 2)$

Table 5: Comparison between training and WGAN simulated samples

Statistic	Dimension 1		Dimension 2	
	Training	Simulated	Training	Simulated
Mean	0.303	0.427 (± 0.027)	−0.006	−0.026 (± 0.013)
Standard deviation	2.336	2.361 (± 0.014)	1.409	1.379 (± 0.019)
1^{st} percentile	−5.512	−5.539 (± 0.099)	−3.590	−3.638 (± 0.116)
99^{th} percentile	4.071	4.103 (± 0.030)	3.862	4.073 (± 0.181)
Statistic	Dimension 3		Dimension 4	
	Training	Simulated	Training	Simulated
Mean	−0.002	0.199 (± 0.021)	−0.063	0.062 (± 0.020)
Standard deviation	1.988	2.002 (± 0.013)	1.982	1.958 (± 0.016)
1^{st} percentile	−4.691	−4.717 (± 0.080)	−4.895	−4.821 (± 0.106)
99^{th} percentile	4.677	4.954 (± 0.066)	4.431	4.865 (± 0.082)

For each value in the correlation matrix, the difference is less than 5%. Compared to the Bernoulli RBM and the Gaussian RBM, the Wasserstein GAN model captures much better the correlation structure of the training dataset. For instance, the correlation between first and second dimensions is equal to −57% for the training data, −53% for the Wasserstein GAN simulated data, −48% for the Gaussian RBM simulated data and −31% for the Bernoulli RBM simulated data. If we consider the first and fourth dimensions, these figures become 29% for the training data, 24% for the Wasserstein GAN simulated data, 22% for the Gaussian RBM simulated data and 20% for the Bernoulli RBM simulated data.

Figure 21: Comparison between the empirical correlation matrix and the average correlation matrix of WGAN simulated data



Training sample

Generated samples

Summary of the results According to our tests, the Wasserstein GAN model performs very well in the task of learning the joint distribution of our simulated training dataset. Comparing with the results of Bernoulli and Gaussian RBMs, the Wasserstein GAN model has several advantages:

1. The Wasserstein GAN model is less sensitive to extreme values of the training dataset

and doesn't have the tendency to underestimate or overestimate the tail of probability distribution.

2. We don't need to apply complex transformation to input data for Wasserstein GAN model in data preprocessing. In our study, we just use a MinMax scaling function and we recall that we need to use the binary transformation for Bernoulli RBM and the normal score transformation for Gaussian RBM.

3. The Wasserstein GAN model may capture more accurately the correlation structure of the training dataset than Bernoulli and Gaussian RBMs.

3.2.2 Application to financial time series

According to the results in the previous section, the Wasserstein GAN model preforms very well in the case of multi-dimensional simulated dataset without the time dependence relationship. In this paragraph, we construct a more complex Wasserstein GAN model with convolutional layers in the generator and the discriminator in order to extract more features in the real financial dataset consisting of 5 354 historical daily returns of the S&P 500 index and the VIX index from December 1998 to May 2018. To capture the time dependence of the multi-dimensional autocorrelated time series, we also need to associate the training samples with conditional labels as we have done in the case of the conditional RBM. Mariani *et al.* (2019) proposed a method to train the model with historical returns over n_t consecutive days conditioned by the last n_h day historical returns. As a result, the generator of this model should generate a matrix belonging to $\mathcal{M}_{n_t,n_x}(\mathbb{R})$, which means the generator simulate directly daily returns of S&P 500 index and the VIX index for n_t days. For reasons of simplicity, this model is named in this paper as the conditional deep convolutional Wasserstein GAN model (or CDCWGAN). In the step of data preprocessing, we need to apply the MinMax scaling function on input data before the training process to ensure that historical returns are scaled into the range $[0, 1]$.

Structure and training of CDCWGAN models In this study on financial time series, the training samples for the discriminator correspond to a n_x-dimensional time series representing historical returns over n_t days. Therefore, inputs are represented by a matrix belonging to $\mathcal{M}_{n_t,n_x}(\mathbb{R})$. In addition, each training sample is conditioned by the past values of returns over n_h days, which are represented by a matrix belonging to $\mathcal{M}_{n_h,n_x}(\mathbb{R})$. In practice, we concatenate these two matrices to form a matrix belonging to the $\mathcal{M}_{n_h+n_t,n_x}(\mathbb{R})$ and this matrix will be fed to the discriminator as inputs. In this study, the inputs of the discriminator will be down sampled using 4 convolutional layers with number of filters $\{16, 32, 64, 128\}$. We set the kernel length to 3 and the stride to 2 and we choose to use the leaky RELU function with $\alpha = 0.5$ for each convolutional layer. For the last layer of the discriminator, we choose always a 1-neuron dense layer and use the tangent hyperbolic activation function for this neuron as in the case of the traditional Wasserstein GAN model.

For the generator, we modify the original method of Mariani *et al.* (2019) and we borrow the idea of conditional layer in the case of conditional RBM. The generator of our CDCWGAN model will take two inputs: a 100-dimensional random noise vector and n_h past values that are concatenated into a $n_h \times n_x$-dimensional vector. We then feed these two vectors to the first layer of the generator, which is a dense layer and we will reshape the output to a two-dimensional $n_t \times 256$ matrix before passing them to the second layer. We construct the rest of the generator using 3 convolutional layers with number of filters $\{256, 64, 2\}$. The kernel length is set to 3 and the stride is set to 2. We also use a leaky RELU function with $\alpha = 0.5$ for each convolutional layer and since the last convolutional layer will give directly the simulated scenarios, we choose the sigmoid activation function in order to get the value in range $[0, 1]$.

For this conditional deep convolutional Wasserstein GAN model, we use always the RMSProp optimizer with a small learning rate. Indeed, the learning rate is set to $\eta = 1.10^{-4}$ and the batch size is set to 500. Table 6 summarizes the setting of the model implementation using Python and the TensorFlow library.

Table 6: The setting of the CDCWGAN model

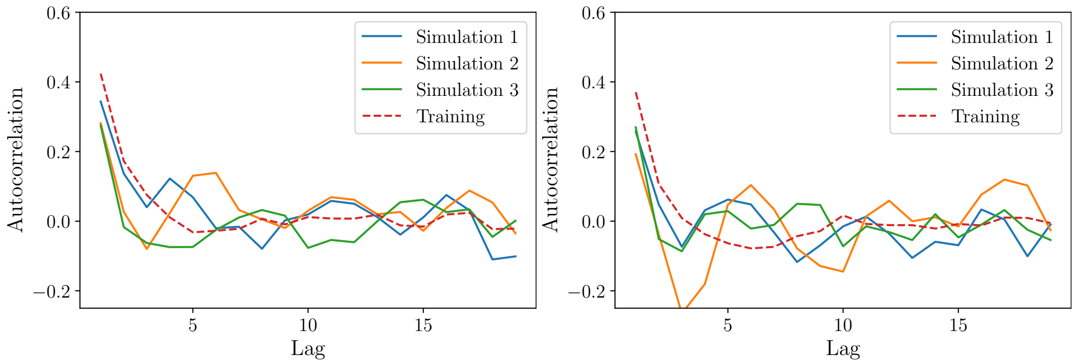
Training data dimension n_x	2
Training data window n_t	5
Training data label length n_h	20
Input feature scaling function	MinMax
Random noise vector dimension	100
Random noise distribution	$\mathcal{N}(0, 1)$
Generator structure	$\{$Dense$(5, 256)$, Conv1D(256), Conv1D(64), Conv1D$(2)\}$
Discriminator structure	$\{$Conv1D(16), Conv1D(32), Conv1D(64), Conv1D(128), Dense$(1)\}$
Loss function	Wasserstein distance
Learning optimizer	RMSProp
Learning rate	1.10^{-4}
Batch size	500

Learning joint distribution and time dependence In this study, we choose to set the length of training data window n_t to 5, which means that the generator of the model will generate at each time step the scenarios for the 5 next days. In addition, we also use the values of the last 20 days as a long memory for input data of the model in order to compare with the results of the conditional RBM. After the training process, we generate samples of consecutive time steps to verify the quality of learning the joint distribution of daily returns of the S&P index and the VIX index. We run 50 Monte Carlo simulations and for each simulation, we generate 5 354 simulated observations from different random noise series. According to Table 7, we find that the CDCWGAN model learns also pretty well the joint distribution of the S&P index and the VIX index. In addition, the average correlation coefficient between daily returns of the two indices over 50 Monte Carlo replications is equal to -71%, which is exactly the figure of the empirical correlation. In Figure 22, we observe clearly that a CDCWGAN model can capture well the autocorrelation of the training dataset as in the case of the conditional RBM shown in Figure 17 on page 29.

Table 7: Comparison between the training sample and simulated samples using the conditional deep convolutional Wasserstein GAN model

Statistic	S&P 500 index		VIX index	
	Training	Simulated	Training	Simulated
Mean	0.02%	0.08% ($\pm$ 0.01%)	0.06%	-0.33% ($\pm$ 0.02%)
Standard deviation	0.64%	0.65% ($\pm$ 0.01%)	3.84%	3.58% ($\pm$ 0.02%)
1$^{\text{st}}$ percentile	-1.93%	-1.57% ($\pm$ 0.02%)	-7.31%	-7.39% ($\pm$ 0.06%)
99$^{\text{th}}$ percentile	1.61%	1.56% ($\pm$ 0.02%)	11.51%	9.74% ($\pm$ 0.13%)

Figure 22: Autocorrelation function of synthetic samples generated by the CDCWGAN model



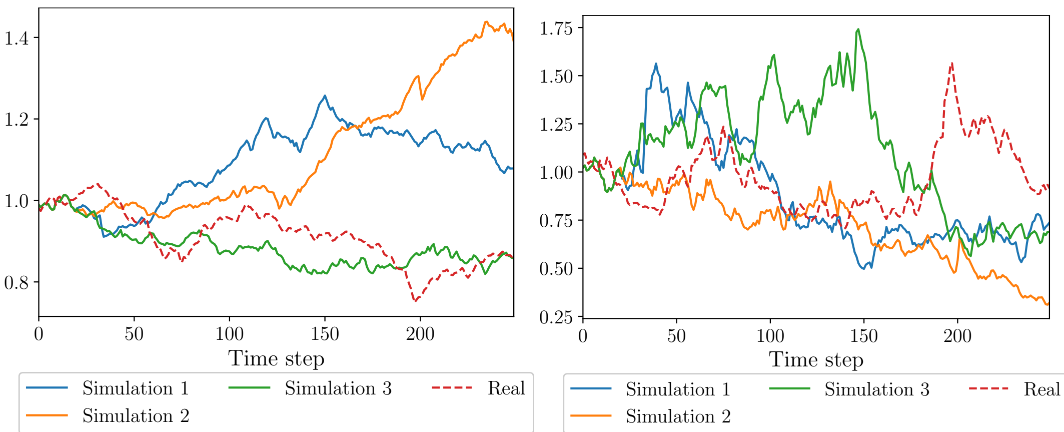
Market generator Based on above results, we consider that the CDCWGAN model can be also used as a market generator. In the example of S&P 500 and VIX indices, we have trained the model with all historical observations using the method proposed by Mariani *et al.* (2019) and the generator of this method will generate scenarios over several consecutive days. After having calibrated the training process, we may choose a date as the starting point and generate new samples for several days. We then update the memory vector and generate samples for the following days. Iteratively, the CDCWGAN model can generate a multi-dimensional time series that has the same patterns as those of the training data in terms of marginal distribution, correlation structure and time dependence. Comparing with the conditional RBM that we have studied, we don't need the normal score transformation to ensures the learning of marginal distributions and, convolutional layers in the generator and the discriminator may help us to extract more features in the training dataset.

As we have shown in Figure 18 on page 30 for the conditional RBM, Figure 23 shows three time series of 250 trading days starting at the date 22/11/2000, which are generated by the trained CDCWGAN model. We notice that these three simulated time series have clearly the positive autocorrelation in each dimension and they capture very well the negative correlation between S&P 500 and VIX indices. Comparing with the historical prices that are represented by the dashed line, these three simulated time series have the same characteristics.

3.3 Managing the out-of-sample robustness of backtesting methods

We now consider an application of generative models in the context of quantitative asset management. Traditionally, we use the historical daily returns of assets to backtest an investment strategy, meaning that only one real time series is available. Therefore, we get only one estimated value for the performance and risk statistics of the strategy, such as the annualized return, the volatility, the Sharpe ratio, the maximum drawdown, etc. Generative models can be used here to manage the out-of-sample robustness of backtesting methods. For instance, we may use a generative model like RBMs and GANs to learn the joint distribution and the time dependence of historical daily returns of assets. We may then use these models to generate new time series. Finally, we may backtest our investment strategy using these new simulated time series in order to construct the probability distribution of the different statistics. In this approach, the estimated value of the statistic obtained with the

Figure 23: Comparison between scaling historical prices of S&P 500 and VIX indices and synthetic time series generated by the CDCWGAN model



S&P 500 index VIX index

true real time series becomes one realization of its probability distribution. Suppose for example that the maximum drawdown of the backtest is equal to 10%, and that the live investment strategy has a maximum drawdown of 20%. Does it mean that the process of the investment strategy has been overfitted? Certainly *yes* if there is a zero probability to experience a maximum drawdown of 20% when the strategy is backtested with generative models. Definitively *not* if some samples of generative models have produced a maximum drawdown larger than 20%.

3.3.1 Backtest of the risk parity strategy

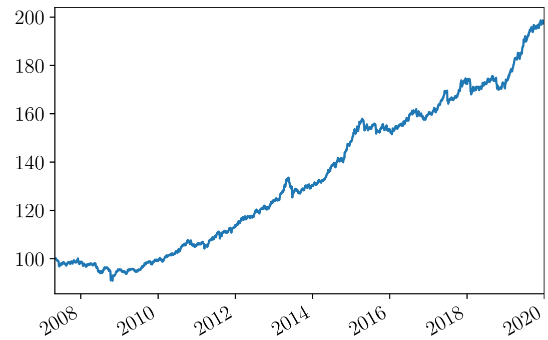
Our dataset consists of daily returns of six futures contracts on world-wide equity indices such as S&P 500, Eurostoxx 50 and Nikkei 225 indices and 10Y sovereign bonds such as US, Germany and Australia from January 2007 to December 2019. In this study, we build a risk parity strategy on this multi-asset investment universe. Moreover, the strategy is unfunded and we calibrate its leverage in order to obtain a volatility around 3%. In Figure 24, we have reported the cumulative performance of the risk parity strategy. Moreover, the descriptive statistics of performance and risk are given in Table 8, and correspond to the annualized performance $\mu(x)$, the volatility $\sigma(x)$, the Sharpe ratio $\mathrm{SR}(x)$, the maximum drawdown[22] $\mathrm{MDD}(x)$ and the skew measure $\xi(x)$, which is the ratio between the maximum drawdown and the volatility[23].

Table 8: Descriptive statistics of the risk parity backtest

Period	$\mu(x)$	$\sigma(x)$	$\mathrm{SR}(x)$	$\mathrm{MDD}(x)$	$\xi(x)$
2007 – 2019	5.30%	3.54%	1.50	9.40%	2.65
2018 – 2019	6.97%	3.48%	2.00	3.69%	1.05

[22] The maximum drawdown corresponds to a loss. For example, if the maximum drawdown is equal to 10%, the investor may face a maximum loss of 10%. This is why the maximum drawdown is expressed by a positive value.
[23] If $\xi(x)$ is greater than 3, this indicates that the strategy has a high skewness risk.

Figure 24: Cumulative performance of the risk parity strategy



3.3.2 Market generator of the risk parity strategy

We first train the conditional RBM introduced in Section 3.1.2 on page 27 on the daily returns of the six futures contracts. In this study, the training of the models is done with all historical observations of futures contracts returns from January 2007 to December 2019. Moreover, for each date, we have used the values of the last 20 days as a memory vector for the conditional layer. We also choose a large number for the number of hidden layers units in order to capture the complex correlation structure of financial time series. Therefore, the conditional RBM has 6 visible layer units, 256 hidden layer units and 120 conditional layer units. In addition, we use the normal score transformation to avoid the problem of outliers in the training dataset and improve the learning quality. After having calibrated the training process, we choose the 1$^{\text{st}}$ January 2018 as the starting point and generate new futures contracts returns iteratively for the next two years. We run 500 Monte Carlo simulations with different random noise series and for each simulation, we backtest our risk parity strategy using synthetic time series and calculate the descriptive statistics of the strategy. Finally, we construct probability distributions for these descriptive statistics and we compare them with the results obtained by the traditional method of bootstrap sampling, which consists of using a random sampling of the historical returns with replacement.

Figure 25: Comparison between cumulative performance of the risk parity strategy using synthetic time series generated by the bootstrap sampling and conditional RBM models

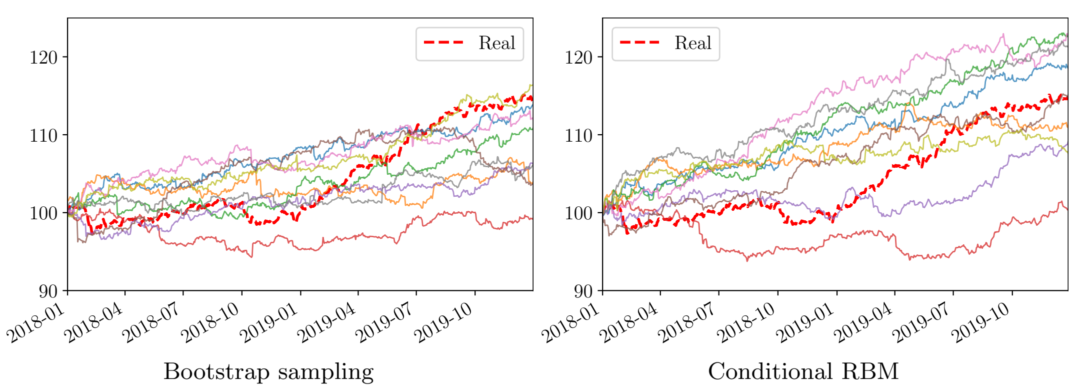
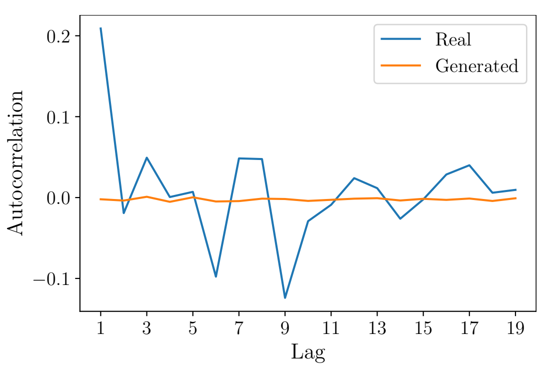
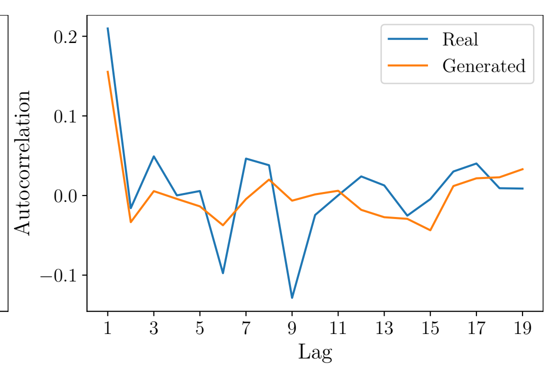


Figure 25 shows the cumulative performance of ten risk parity backtests using synthetic time series generated by the two methods. Among these simulations, we notice that the cumulative performance of the risk parity strategy using synthetic time series generated by the bootstrap sampling method is more centered around the real time series of backtesting. This phenomenon corresponds to the drawback of traditional bootstrap sampling method, since it cannot capture the time dependence in risk parity strategy. As shown in Figure 26, our risk parity strategy has a first-lag positive autocorrelation of 20%, and we compare the average autocorrelation function over 500 Monte Carlo simulations generated by the two approaches. We notice that the bootstrap sampling method cannot replicate this positive autocorrelation as the conditional RBM can do. This property is very important when we backtest meta-strategies, which means a strategy of an existing strategy. For instance, if we want to design a stop-loss strategy for our risk parity strategy, we should tune several parameters for implementing this stop-loss strategy. If we use the real historical time series to calibrate the stop-loss parameters, we might fall into the trap of overfitting because we only have one real historical time series. Therefore, if we use the time series generated by the bootstrap sampling method, we will not find the appropriate values of the stop-loss parameters, since the autocorrelation plays a key role in the stop-loss strategy as explained by Kaminski and Lo (2014). Using the conditional RBM as a market generator to generate time series is then a better way to find the appropriate parameters for the meta-strategy and manage the out-of-sample robustness.

Figure 26: Autocorrelation function of the risk parity strategy using synthetic time series generated by the bootstrap sampling and conditional RBM methods



Bootstrap sampling



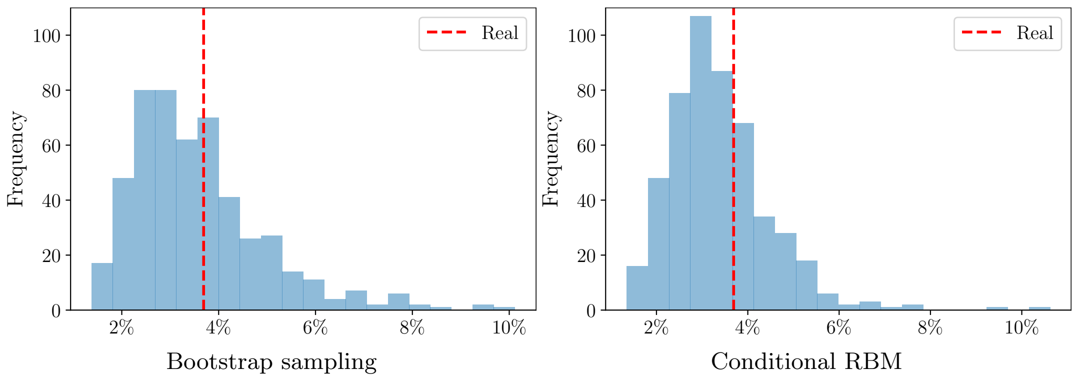
Conditional RBM

3.3.3 Building the probability distribution of backtest statistics

In our study, we are interested in building the probability distribution of the maximum drawdown $\mathrm{MDD}(x)$ and the skew measure $\xi(x)$ for the risk parity strategy. Figure 27 shows the distributions of the maximum drawdown generated by the bootstrap sampling and conditional RBM methods. We notice that the distribution of the maximum drawdown generated by the conditional RBM are more centered and have less severe scenarios. For instance, the value of the maximum drawdown of the real risk parity strategy from January 2018 to December 2019 is 3.69% and this value corresponds respectively to the 61.5% quantile in the probability distribution generated by the bootstrap sampling method and the 69.1% quantile in the probability distribution generated by the conditional RBM. We consider that the difference comes from the quality of learning about the time dependence of the dataset.

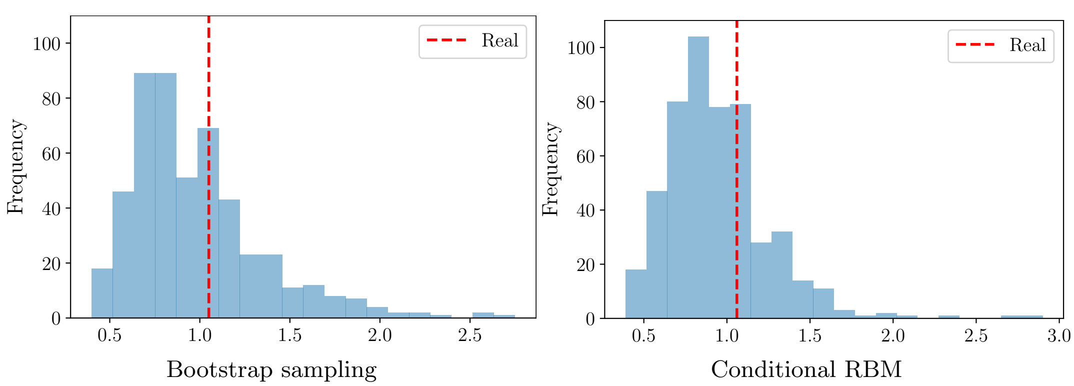
Although we use a volatility targeting method to control the volatility of risk parity strategies, we cannot ensure that the strategy volatility is exactly equal to 3%. To avoid the

Figure 27: Histogram of the maximum drawdown of the risk parity strategy using synthetic time series generated by the bootstrap sampling and conditional RBM methods



influence of the difference between realized volatility, we plot in Figure 28 the probability distribution of the skew measure. In this case, we notice that the tail of the probability distribution generated by the conditional RBM is less fat than that generated by the bootstrap sampling method but has several extreme severe scenarios. The skew measure of the real risk parity strategy from January 2018 to December 2019 is equal to 1.05, and this value corresponds respectively to the 66.3% quantile in the probability distribution generated by the bootstrap sampling method and the 70.1% quantile in the probability distribution generated by the conditional RBM.

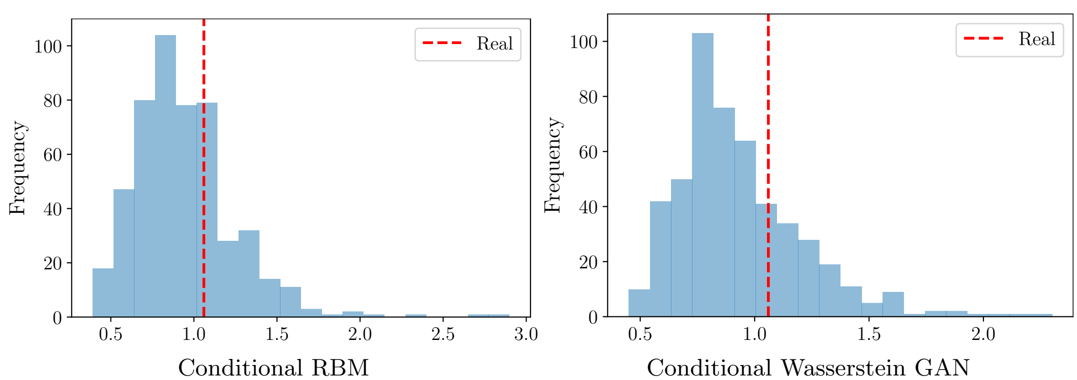
Figure 28: Histogram of the skew measure of the risk parity strategy using synthetic time series generated by the bootstrap sampling and the conditional RBM methods



3.3.4 Comparison with Wasserstein GAN models

For the purpose of illustration, we train also a conditional Wasserstein GAN model with a simple structure as multi-layer perceptrons for the generator and the discriminator. In order to make a comparison with the results of the conditional RBM, we also use the values of the last 20 days as a long memory for input data of the model. More precisely, the generator will take two inputs: a 100-dimensional random noise vector and 20 past values that are concatenated into a 20×6-dimensional vector. We then construct the generator using 4 dense layers with the structure $\mathcal{S}_{\mathcal{G}} = \{100, 50, 10, 6\}$. We use a leaky RELU function with $\alpha = 0.5$ for each dense layer and a sigmoid activation function for the output layer. The discriminator also takes two inputs: a 6-dimensional vector of current daily returns of futures contracts and a 20×6-dimensional vector of past values. We then construct the discriminator using 4 dense layers with the structure $\mathcal{S}_{\mathcal{D}} = \{100, 50, 10, 1\}$. For the first three dense layers, the activation function corresponds to a leaky RELU function with $\alpha = 0.5$, whereas the activation function of the output layer is a tanh activation function. To avoid the problem of outliers, we also use the normal score transformation as in the case of conditional RBMs before applying the MinMax scaling function on input data. In Figure 29, we compare the distributions of the skew measure of the risk parity strategy using synthetic time series generated by the conditional RBM and conditional Wasserstein GAN models. We notice that these two distributions are similar, except that we have several extreme severe scenarios in the case of the conditional RBM. We recall that the value of the skew measure of the real risk parity strategy from January 2018 to December 2019 is equal to 1.05 and this value corresponds respectively to the 70.1% quantile in the probability distribution generated by the conditional RBM and the 72.9% quantile in the probability distribution generated by the conditional Wasserstein GAN.

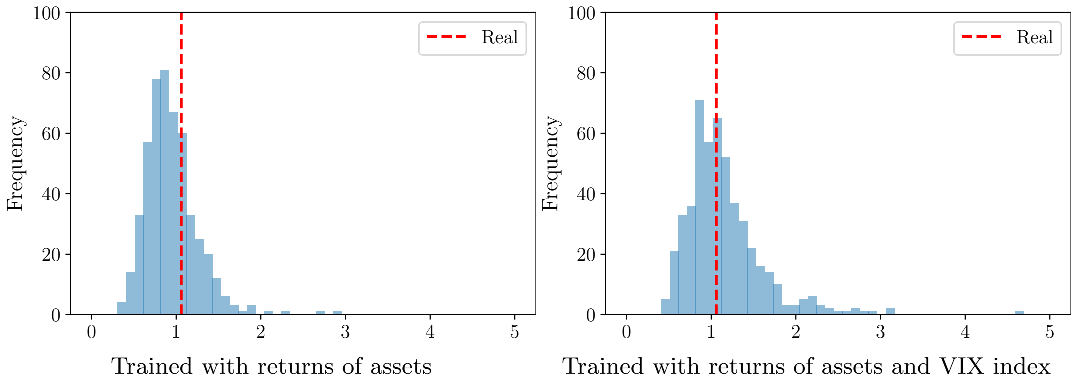
Figure 29: Histogram of the skew measure of the risk parity strategy using synthetic time series generated by the conditional RBM and Wasserstein GAN models



3.3.5 Augmenting the investment universe with market regime indicators

In the real world of finance, the correlation structure is very complex and learning the joint distribution of daily returns of futures contracts using only time series themselves is not sufficient. For instance, if we train the generative models by including the time series of the VIX index, it's sure that we will obtain different results. Indeed, since the daily returns of the VIX index has a very negative correlation with equity futures contracts and a positive correlation with bond futures contracts and its distribution is very leptokurtic, the generative models trained using historical returns of futures contracts and VIX index will generate more distinct and more severe scenarios. Figure 30 shows the histograms of the skew measure of the risk parity strategy using synthetic time series generated by two conditional RBMs. The first one is trained with only historical daily returns of futures contracts and the other is trained with historical daily returns of futures contracts and VIX index. In this figure, we notice that the probability distribution generated by the model trained with the VIX index has a fatter tail and more extreme scenarios. Therefore, we consider that this model may generate more realistic data than the model trained using only the time series of the future contracts. In order to have a high-quality market generator, we believe that we should train a conditional RBM or Wasserstein GAN model with not only the time series of assets that compose the investment portfolio, but also those of the several market regime indicators such as the VIX index.

Figure 30: Histogram of the skew measure of the risk parity strategy using synthetic time series generated by two conditional RBMs



4 Conclusion

In this article, we explore the use of generative models for improving the robustness of trading strategies. We consider two approaches for simulating financial time series. The first one is based on restricted Boltzmann machines, whereas the second approach corresponds to the family of generative adversarial networks, including Wasserstein distance models. Given an historical sample of financial data, we show how to generate new samples of financial data using these techniques. These new samples of financial data are called synthetic or fake financial time series, and the objective is to preserve the statistical properties of the original sample. By statistical properties, we mean the statistical moments of each univariate time series, the stochastic dependence between the different variables that compose the multi-dimensional random vector, and also the time dependence between the observations. If we consider the financial times series as a multi-dimensional data matrix, the challenge is then to model both the row and column stochastic structures.

There are few satisfactory methods for simulating non-gaussian multi-dimensional financial time series. For instance, the bootstrap method does not preserve the cross-correlation between the different variables. The copula method is better, but it must use the techniques of conditional augmented data in order to reproduce the autocorrelation functions. Restricted Boltzmann machines and generative adversarial networks have been successful for generating complex data with non-linear dependence structures. By applying them to financial market data, our first results are encouraging and show that these new alternative techniques may help for simulating non-gaussian multi-dimensional financial time series. This is particularly true when we consider the backtesting of trading strategies. In this case, RBMs and GANs may be used for estimating the probability distribution of performance and risk statistics of the backtest. This opens the door to a new field of research for improving the risk management of quantitative investment strategies.

References

Ackley, D., Hinton, G.E., and Sejnowski, T.J. (1985), A Learning Algorithm for Boltzmann Machines, *Cognitive Science*, 9(1), pp. 147-169.

Arjovsky, M., Chintala, S., and Bottou, L. (2017a), Wasserstein GAN, *arXiv*, 1701.07875.

Arjovsky, M., Chintala, S. and Bottou, L. (2017b), Wasserstein Generative Adversarial Networks, in Precup, D., and Teh, Y.W. (eds), *Proceedings of the 34th International Conference on Machine Learning*, 70, pp. 214-223.

Barbu, V., and Precupanu, T. (2012), *Convexity and Optimization in Banach spaces*, Fourth edition, Springer Monographs in Mathematics, Springer.

Bengio, Y., and Delalleau, O. (2009), Justifying and Generalizing Contrastive Divergence, *Neural Computation*, 21(6), pp. 1601-1621.

Brenier, Y. (1991), Polar Factorization and Monotone Rearrangement of Vector-valued Functions, *Communications on Pure and Applied Mathematics*, 44(4), pp. 375-417.

Broniatowski, M., and Keziou, A. (2006), Minimization of ϕ-divergences on Sets of Signed Measures, *Studia Scientiarum Mathematicarum Hungarica*, Akadémiai Kiadó, 43(4), pp. 403-442.

Cho, K., Ilin, A., and Raiko, T. (2011), Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines, in *Proceedings of the Twentith International Conference on Artificial Neural Networks (ICANN) 2011*, pp. 10-17.

Chu, C., Blanchet, J., and Glynn, P. (2019), Probability Functional Descent: A Unifying Perspective on GANs, Variational Inference, and Reinforcement Learning, *arXiv*, 1901.10691.

Clevert, D., Unterthiner, T., and Hochreiter, S. (2015), Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), *arXiv*, 1511.07289.

Cont, R. (2001), Empirical Properties of Asset Returns: Stylized Facts and Statistical Issues, *Quantitative Finance*, 1(2), pp. 223-236.

Cui, Z., Chen, W., and Chen, Y. (2016), Multi-scale Convolutional Neural Networks for Time Series Classification, *arXiv*, 1603.06995.

Denton, E.L., Chintala, S., Szlam, A., and Fergus, R. (2015), Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks, in Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., and Garnett, R. (eds), *Advances in Neural Information Processing Systems*, 28, pp. 1486-1494.

Dumoulin, V., and Visin, F. (2016), A Guide to Convolution Arithmetic for Deep Learning, *arXiv*, 1603.07285.

Fernholz, L.T. (2012), *Von Mises Calculus for Statistical Functionals*, Lecture Notes in Statistics, 19, Springer.

Fischer, A. and Igel, C. (2014), Training Restricted Boltzmann Machines: An Introduction, *Pattern Recognition*, 47(1), pp. 25-39.

Givens, C.R., and Shortt, R.M. (1984), A Class of Wasserstein Metrics for Probability Distributions, *Michigan Mathematical Journal*, 31(2), pp. 231-240.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014), Generative Adversarial Nets, in Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds), *Advances in Neural Information Processing Systems*, 27, pp. 2672-2680.

Gozlan, N., Roberto, C., Samson, P.M., and Tetali, P. (2017), Kantorovich Duality for General Transport Costs and Applications, *Journal of Functional Analysis*, 273(11), pp. 3327-3405.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A.C. (2017), Improved Training of Wasserstein GANs, in Guyon, I, Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds), *Advances in Neural Information Processing Systems*, 30, pp. 5767-5777.

Hinton, G.E. (2002), Training Products of Experts by Minimizing Contrastive Divergence, *Neural Computation*, 14(8), pp. 1771-1800.

Hinton, G.E. (2012), A Practical Guide to Training Restricted Boltzmann Machines, in Montavon, G., Orr, G.B., Müller, K-R. (eds), *Neural Networks: Tricks of The Trade*, pp. 599-619, Second edition, Springer.

Hinton, G.E., and Sejnowski, T.J. (1986), Learning and Relearning in Boltzmann Machines, Chapter 7 in Rumelhart, D.E., and McClelland, J.L. (eds), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, pp. 282-317, MIT Press.

Hyland, S.L., Esteban, C., and Rätsch, G. (2017), Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs, *arXiv*, 1706.02633.

Isola, P., Zhu, J.Y., Zhou, T., and Efros, A.A. (2017), Image-to-image Translation with Conditional Adversarial Networks, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1125-1134.

Jebara, T. (2004), *Machine Learning: Discriminative and Generative*, Springer International Series in Engineering and Computer Science, 755, Springer.

Kaminski, K.M., and Lo, A.W. (2014), When Do Stop-loss Rules Stop Losses?, *Journal of Financial Markets*, 18, pp. 234-254.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018), Progressive Growing of GANs for Improved Quality, Stability, and Variation, *International Conference on Learning Representations (ICLR 2018)*, 7, available on *arXiv*, 1710.10196.

Keziou, A. (2003), Dual Representation of ϕ-divergences and Applications, *Comptes Rendus de l'Académie des Sciences – Series I – Mathematics*, 336(10), pp. 857-862.

Kindermann, R. and Snell, J.L. (1980), *Markov Random Fields and Their Applications*, Contemporary Mathematics, American Mathematical Society.

Koller, D. and Friedman, N. (2009), *Probabilistic Graphical Models: Principles and Techniques*, MIT Press.

Kondratyev, A., and Schwarz, C. (2020), The Market Generator, *SSRN*, https://www.ssrn.com/abstract=3384948.

Koshiyama, A., Firoozye, N., and Treleaven, P. (2019), Generative Adversarial Networks for Financial Trading Strategies Fine-Tuning and Combination, *arXiv*, 1901.01751.

Krizhevsky, A. (2009), Learning Multiple Layers of Features from Tiny Images, University of Toronto, *Technical Report*.

Laschos, V., Obermayer, K., Shen, Y., and Stannat, W. (2019), A Fenchel-Moreau-Rockafellar Type Theorem on the Kantorovich-Wasserstein Space with Applications in Partially Observable Markov Decision Processes, *Journal of Mathematical Analysis and Applications*, 477(2), pp. 1133-1156.

LeCun, Y., and Bengio, Y. (1995), Convolutional Networks for Images, Speech, and Time Series, in Arbib, M.A. (ed.), *The Handbook of Brain Theory and Neural Networks*, MIT Press.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M.A., and Huang, F.J. (2007), A Tutorial on Energy-based Learning, Chapter 10 in Bakır, G., Hofmann, T., Schölkopf, B., Smola, A.J., Taskar, B., and Vishwanathan, S.V.N. (eds), *Predicting Structured Data*, pp. 191-246, MIT Press.

Liu, R., Lehman, J., Molino, P., Such, F.P., Frank, E., Sergeev, A., and Yosinski, J. (2018), An Intriguing Failing of Convolutional Neural Networks and the Coordconv Solution, in Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds), *Advances in Neural Information Processing Systems*, 31, pp.,9605-9616.

Long, P. M. and Servedio, R. A. (2010), Restricted Boltzmann Machines are Hard to Approximately Evaluate or Simulate, in Fürnkranz, J., and Joachims, T. (eds), *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, pp. 703-710, Omnipress.

Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., and Smolley, P.S. (2017), Least Squares Generative Adversarial Networks, *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2794-2802.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2017), Unrolled Generative Adversarial Networks, *International Conference on Learning Representations (ICLR 2018)*, 7, available on *arXiv*, 1611.02163.

Mariani, G., Zhu, Y., Li, J., Scheidegger, F., Istrate, R., Bekas, C., Cristiano, A., and Malossi, I. (2019), PAGAN: Portfolio Analysis with Generative Adversarial Networks, *arXiv*, 1909.10578.

Mirza, M., and Osindero, S. (2014), Conditional Generative Adversarial Nets, *arXiv*, 1411.1784.

Mroueh, Y., and Sercu, T. (2017), Fisher GAN, in Guyon, I, Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds), *Advances in Neural Information Processing Systems*, 30, pp. 2513-2523.

Mroueh, Y., Li, C.L., Sercu, T., Raj, A., and Cheng, Y. (2017), Sobolev GAN, *International Conference on Learning Representations (ICLR 2018)*, 7, available on *arXiv*, 1711.04894.

Müller, A. (1997), Integral Probability Metrics and Their Generating Classes of Functions, *Advances in Applied Probability*, 29(2), pp. 429-443.

Nguyen, X., Wainwright, M.J., and Jordan, M.I.(2010), Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization, *IEEE Transactions on Information Theory*, 56(11), pp. 5847-5861.

Nowozin, S., Cseke, B., and Tomioka, R. (2016), f-gan: Training Generative Neural Samplers using Variational Divergence Minimization, in Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., and Garnett, R. (eds), *Advances in Neural Information Processing Systems*, 29, pp 271-279.

Pearl, J. (1985), Bayesian Networks: A Model of Self-activated Memory for Evidential Reasoning, in *Proceedings of the 7th Conference of the Cognitive Science Society*, pp. 329-334.

Peyré, G., and Cuturi, M. (2019), Computational Optimal Transport, *Foundations and Trends® in Machine Learning*, 11(5-6), pp. 355-607.

Rachev, S.T. (1985), The Monge-Kantorovich Mass Transference Problem and its Stochastic Applications, *Theory of Probability & Its Applications*, 29(4), pp. 647-676.

Rachev, S.T., and Rüschendorf, L. (1998), *Mass Transportation Problems: Theory (Volume 1)*, Springer.

Radford, A., Metz, L., and Chintala, S. (2016), Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, *International Conference on Learning Representations (ICLR 2016)*, available on *arXiv*, 1511.06434.

Rubner, Y., Tomasi, C., and Guibas, L.J. (2000), The Earth Mover's Distance as a Metric for Image Retrieval, *International Journal of Computer Vision*, 40(2), pp. 99-121.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016), Improved Techniques for Training GANs, in Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., and Garnett, R. (eds), *Advances in Neural Information Processing Systems*, 29, pp. 2234-2242.

Seguy, V., Damodaran, B.B., Flamary, R., Courty, N., Rolet, A., and Blondel, M. (2017), Large-scale Optimal Transport and Mapping Estimation, *arXiv*, 1711.02283.

Smolensky, P. (1986), Information Processing in Dynamical Systems: Foundations of Harmony Theory, Chapter 6 in Rumelhart, D.E., and McClelland, J.L. (eds), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, pp. 194-281, MIT Press.

Taylor, G.W., Hinton, G.E., and Roweis, S.T. (2011), Two Distributed-state Models For Generating High-Dimensional Time Series, *Journal of Machine Learning Research*, 12, pp. 1025-1068.

Villani, C. (2008), *Optimal Transport: Old and New*, Grundlehren der mathematischen Wissenschaften, 338, Springer.

Wiese, M., Knobloch, R., Korn, R., and Kretschmer, P. (2020), Quant GANs: Deep Generation of Financial Time Series, *Quantitative Finance*, forthcoming.

Xia, Q. (2008), Numerical Simulation of Optimal Transport Paths, *arXiv*, 0807.3723.

Xia, Q. (2009), The Geodesic Problem in Quasimetric Spaces, *Journal of Geometric Analysis*, 19(2), pp. 452-479.

Appendix

A Mathematical results

A.1 Fundamental concepts of undirected graph model

Probabilistic graphical models use graphs to describe interactions between random variables. Each random variable is represented by a node (or vertice) and each direct interaction between random variables is represented by an edge (or link). According to the directionality of the edge in the graph, probabilistic graphical models can be divided into two categories: directed graphical model and undirected graphical model. For instance, the Bayesian networks that are introduced by Pearl (1985) are a type of directed graphical models, whereas Markov random fields (or Markov networks) use undirected graphs (Kindermann and Snell, 1980).

A.1.1 Undirected graph

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a defined by its finite set of nodes $\mathcal{V}$ and its set of undirected edges $\mathcal{E}$. Each edge $e_{i,j}$ is defined by a pair of two connected nodes v_i and v_j from $\mathcal{V}$. We define the neighborhood $\mathcal{N}(v_i)$ of a given node v_i as the set of all nodes connected to v_i:

$$\mathcal{N}(v_i) = \{v_j \in \mathcal{V} \mid e_{i,j} \in \mathcal{E}\}$$

A clique of size n named $\mathcal{C}_n$ is a subset of $\mathcal{V}$ containing n nodes $(v_1, v_2, \ldots, v_n)$ defined as:

$$\forall i, j \in \{1, 2, \ldots, n\} \text{ and } i \neq j : e_{i,j} \in \mathcal{E}$$

In other words, each node belonging to $\mathcal{C}_n$ is fully connected with the other nodes of $\mathcal{C}_n$. A clique of a graph $\mathcal{G}$ is called maximal if we can't create a bigger clique by adding another node of $\mathcal{G}$, meaning that no node in $\mathcal{G}$ can be added such that the resulting set is still a clique.

A.1.2 Markov random field

Let $X = (X(v_1), X(v_2), \ldots, X(v_n))$ be a set of random variables associated with the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that each random variable $X(v_i)$ is linked to the i^{th} node $v_i \in \mathcal{V}$. X is said to be a Markov random field (MRF) if, for all $i \in \{1, 2, \ldots, n\}$, $X(v_i)$ is conditionally independent from all other variables $X(v_j)$, whose nodes v_j do not belong to the neighborhood $\mathcal{N}(v_i)$:

$$\mathbb{P}(x(v_i) \mid x(v_j), v_j \in \mathcal{V} \setminus v_i) = \mathbb{P}(x(v_i) \mid x(v_j), v_j \in \mathcal{N}(v_i))$$

In other words, X is said to be a Markov random field if the joint probability distribution verifies the local Markov property.

A.1.3 Hammersley-Clifford theorem

Let $X = (X(v_1), X(v_2), \ldots, X(v_n))$ be a Markov random field and $\mathcal{C}$ the set of all maximal cliques of the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. According to Fischer and Igel (2014), a simple version[24] of Hammersley-Clifford theorem states that a strictly positive distribution $\mathbb{P}$ satisfies the Markov property with respect to the undirected graph $\mathcal{G}$ if and only if $\mathbb{P}$ factorizes

[24] The rigorous formulation of the Hammersley-Clifford theorem can be found in Koller and Friedman (2009).

over $\mathcal{G}$. This means that there exists a set of strictly positive functions $\{\psi_C, C \in \mathcal{C}\}$, such that the joint probability distribution is given by a product of factors:

$$\mathbb{P}(x) = \mathbb{P}(x(v_1), \ldots, x(v_n)) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x(v_C)) \tag{21}$$

where ψ_C is the potential function for the clique C, v_C are all the nodes belonging to the clique C and Z is the partition function given by:

$$Z = \sum_x \prod_{C \in \mathcal{C}} \psi_C(x(v_C))$$

The partition function Z is the normalization constant what ensures the overall distribution sums to 1.

A.1.4 Energy function and Boltzmann distribution

The Hammersley-Clifford theorem is valid only if each potential function ψ_C is strictly positive. Thus, we can introduce a new function E in order to rewrite the probability distribution $\mathbb{P}(x)$:

$$\begin{aligned}\mathbb{P}(x) &= \frac{1}{Z} \exp\left(\sum_{C \in \mathcal{C}} \log \psi_C(x(v_C))\right) \\ &= \frac{1}{Z} e^{-E(x)} \end{aligned} \tag{22}$$

where $E(x) = -\sum_{C \in \mathcal{C}} \log \psi_C(x(v_C))$ is called the energy function. Because natural exponential function is always positive, this guarantees that the energy function will result in a positive probability for any state. In addition, a large value of energy indicates a low probability of the state. According to LeCun *et al.* (2007), models of this form are called energy-based models.

Using the energy function described above, the strictly positive probability distribution of a Markov random field can be expressed in the form $\mathbb{P}(x) = Z^{-1}e^{-E(x)}$. This form of distribution is also called Boltzmann (or Gibbs) distribution for a system in statistical physics. It is defined as:

$$p_i = \frac{1}{Z_T} \exp\left(-\frac{E_i}{kT}\right)$$

where p_i is the probability of state i of the system, E_i is the energy of state i, k is the Boltzmann constant, and T is the temperature of the system. Since N is the number of states accessible to the system, the normalization constant is $Z_T = \sum_{i=1}^{N} e^{-E_i/(kT)}$. If we set kT to 1, we find that the Boltzmann distribution and the probability distribution of a Markov random field have the same formula. For this reason, many energy-based models are called Boltzmann machines.

A.1.5 The example of restricted Boltzmann machines

The restricted Boltzmann machine introduced in Section 2.1 on page 3 is a Markov random field associated with a bipartite undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. In other words, all visible layer units and hidden layer units can be considered as nodes in an undirected graph[25]:

$$\mathcal{V} = \{V_1, V_2, \ldots, V_m, H_1, H_2, \ldots, H_n\}$$

[25] For simplicity reasons, we make a little abuse of notation here to use V_i and H_j to represent not only nodes in the graph but also random variables associated with these nodes and we use v_i and h_j to denote respectively the possible values of the variables associated with the i^{th} visible unit and j^{th} hidden unit.

and all connections between visible layer and hidden layer are edges of $\mathcal{G}$. In the case of RBMs, we know that there are only cliques of size 1 (one visible unit or one hidden unit) and cliques of size 2 (a pair of one visible unit and one hidden unit) in the graph $\mathcal{G}$. In addition, it is easy to show that all these cliques are maximal. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be respectively the set of all the cliques of size 1 and the set of all the cliques of size 2. We obtain:

$$\mathcal{C}_1 = \{\{V_1\}, \{V_2\}, \ldots, \{V_m\}, \{H_1\}, \{H_2\}, \ldots, \{H_n\}\}$$

and:

$$\mathcal{C}_2 = \{\{V_1, H_1\}, \ldots, \{V_i, H_j\}, \ldots, \{V_m, H_n\}\}$$

According to the Hammersley-Clifford theorem, the probability distribution of an RBM is given by:

$$\begin{aligned}
\mathbb{P}(v,h) &= \frac{1}{Z} \prod_{C \in \{\mathcal{C}_1, \mathcal{C}_2\}} \psi_C \\
&= \frac{1}{Z} \prod_{i=1}^{m} \psi_{V_i}(v_i) \prod_{j=1}^{n} \psi_{H_j}(h_j) \prod_{i=1}^{m} \prod_{j=1}^{n} \psi_{V_i,H_j}(v_i, h_j) \\
&= \frac{1}{Z} e^{-E(v,h)}
\end{aligned}$$

where:

$$\begin{aligned}
E(v,h) &= -\log\left(\prod_{i=1}^{m} \psi_{V_i}(v_i) \prod_{j=1}^{n} \psi_{H_j}(h_j) \prod_{i=1}^{m} \prod_{j=1}^{n} \psi_{V_i,H_j}(v_i, h_j)\right) \\
&= -\sum_{i=1}^{m} \log \psi_{V_i}(v_i) - \sum_{j=1}^{n} \log \psi_{H_j}(h_j) - \\
&\quad \sum_{i=1}^{m} \sum_{j=1}^{n} \log \psi_{V_i,H_j}(v_i, h_j) \\
&= \sum_{i=1}^{m} E_i(v_i) + \sum_{j=1}^{n} E_j(h_j) + \sum_{i=1}^{m} \sum_{j=1}^{n} E_{i,j}(v_i, h_j)
\end{aligned}$$

In the case of Bernoulli RBMs introduced in Section 2.1.1 on page 4, we defined $E_i(v_i) = -a_i v_i$, $E_j(h_j) = -b_j h_j$ and $E_{i,j}(v_i, h_j) = -w_{i,j} v_i h_j$. It follows that the energy function of a Bernoulli RBM is equal to:

$$E(v,h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m} \sum_{j=1}^{n} w_{i,j} v_i h_j$$

where a_i and b_j are bias terms associated with the visible and hidden variables V_i and H_j, and $w_{i,j}$ is the weight associated with the edge between V_i and H_j.

A.2 Calculus formulas for restricted Boltzmann machines

A.2.1 Conditional probability $\mathbb{P}(h_j = 1 \mid v)$

We have:

$$\begin{aligned}
\sum_h \mathbb{P}(h \mid v) h_j &= \sum_h \prod_{k=1}^n \mathbb{P}(h_k \mid v) h_j \\
&= \sum_h \mathbb{P}(h_{-j} \mid v) \mathbb{P}(h_j \mid v) h_j \\
&= \sum_{h_j \in \{0,1\}} \sum_{h_{-j}} \mathbb{P}(h_{-j} \mid v) \mathbb{P}(h_j \mid v) h_j \\
&= \sum_{h_j \in \{0,1\}} \mathbb{P}(h_j \mid v) h_j \cdot \underbrace{\sum_{h_{-j}} \mathbb{P}(h_{-j} \mid v)}_{=1} \\
&= \mathbb{P}(h_j = 1 \mid v)
\end{aligned} \tag{23}$$

where $h_{-j} = (h_1, h_2, \ldots, h_{j-1}, h_{j+1}, \ldots, h_n)$ denotes the state of all hidden units except the j^{th} one.

A.2.2 Bernoulli RBMs and neural networks

Following Fischer and Igel (2014), we divide the energy function $E(v, h)$ into two parts: one collecting all terms involving v_i and one collecting all the other terms v_{-i}:

$$\begin{aligned}
E(v, h) &= -\sum_{k=1}^m a_k v_k - \sum_{j=1}^n b_j h_j - \sum_{k=1}^m \sum_{j=1}^n w_{k,j} v_k h_j \\
&= -a_i v_i - \sum_{j=1}^n w_{i,j} v_i h_j - \sum_{k=1, k \neq i}^m a_k v_k - \sum_{j=1}^n b_j h_j - \sum_{k=1, k \neq i}^m \sum_{j=1}^n w_{k,j} v_k h_j \\
&= v_i \alpha_i(h) + \beta(v_{-i}, h)
\end{aligned}$$

where:

$$\alpha_i(h) = -a_i - \sum_{j=1}^n w_{i,j} h_j$$

and:

$$\beta(v_{-i}, h) = -\sum_{k=1, k \neq i}^m a_k v_k - \sum_{j=1}^n b_j h_j - \sum_{k=1, k \neq i}^m \sum_{j=1}^n w_{k,j} v_k h_j$$

The Bayes theorem gives:

$$\begin{aligned}
\mathbb{P}(v_i = 1 \mid h) &= \mathbb{P}(v_i = 1 \mid v_{-i}, h) \\
&= \frac{\mathbb{P}(v_i = 1, v_{-i}, h)}{\mathbb{P}(v_{-i}, h)} \\
&= \frac{\mathbb{P}(v_i = 1, v_{-i}, h)}{\mathbb{P}(v_i = 0, v_{-i}, h) + \mathbb{P}(v_i = 1, v_{-i}, h)}
\end{aligned}$$

We deduce that the conditional probability $\mathbb{P}(v_i = 1 \mid h)$ is equal to:

$$\begin{aligned}
\mathbb{P}(v_i = 1 \mid h) &= \frac{e^{-E(v_i=1,v_{-i},h)}}{e^{-E(v_i=0,v_{-i},h)} + e^{-E(v_i=1,v_{-i},h)}} \\
&= \frac{e^{-1\cdot\alpha_i(h)-\beta(v_{-i},h)}}{e^{-0\cdot\alpha_i(h)-\beta(v_{-i},h)} + e^{-1\cdot\alpha_i(h)-\beta(v_{-i},h)}} \\
&= \frac{e^{-\alpha_i(h)}}{1+e^{-\alpha_i(h)}} \\
&= \frac{1}{1+e^{\alpha_i(h)}} \\
&= \sigma(-\alpha_i(h))
\end{aligned}$$

where $\sigma(x)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Similarly, we can divide the energy function $E(v,h)$ into two parts: one collecting all terms involving h_j and one collecting all the other terms h_{-j}:

$$E(v,h) = h_j\gamma_j(v) + \delta(v, h_{-j})$$

where:

$$\gamma_j(v) = -b_j - \sum_{i=1}^{m} w_{i,j}v_i$$

and:

$$\delta(v,h_{-j}) = -\sum_{i=1}^{m} a_i v_i - \sum_{k=1,k\neq j}^{n} b_k h_k - \sum_{i=1}^{m}\sum_{k=1,k\neq j}^{n} w_{i,k}v_i h_k$$

Using the same approach as previously, we can show that:

$$\mathbb{P}(h_j = 1 \mid v) = \sigma(-\gamma_j(v))$$

A.2.3 Gradient of the Bernoulli RBM log-likelihood function

We have:

$$\ell(\theta \mid v) = \log\left(\sum_h e^{-E(v,h)}\right) - \log\left(\sum_{v',h} e^{-E\left(v',h\right)}\right)$$

Fischer and Igel (2014) computed the log-likelihood gradient $\nabla_\theta(v) = \partial_\theta \ell(\theta \mid v)$:

$$\begin{aligned}
\nabla_\theta(v) &= \frac{\partial}{\partial\,\theta}\left(\log\sum_h e^{-E(v,h)}\right) - \frac{\partial}{\partial\,\theta}\left(\sum_{v',h} e^{-E\left(v',h\right)}\right) \\
&= \nabla_\theta^{(1)}(v) + \nabla_\theta^{(2)}(v') \qquad (24)
\end{aligned}$$

We have[26]:

$$
\begin{aligned}
\nabla_\theta^{(1)}(v) &= -\frac{\sum_h e^{-E(v,h)} \cdot \partial_\theta E(v,h)}{\sum_h e^{-E(v,h)}} \\
&= -\sum_h \frac{e^{-E(v,h)}}{\sum_{h'} e^{-E(v,h')}} \partial_\theta E(v,h) \\
&= -\sum_h \mathbb{P}(h \mid v) \frac{\partial E(v,h)}{\partial \theta}
\end{aligned}
$$

and:

$$
\begin{aligned}
\nabla_\theta^{(2)}(v') &= \frac{\sum_{v',h} e^{-E\left(v',h\right)} \cdot \partial_\theta E(v',h)}{\sum_{v',h} e^{-E(v',h)}} \\
&= \sum_{v',h} \frac{e^{-E\left(v',h\right)}}{\sum_{v'',h'} e^{-E(v'',h')}} \frac{\partial E(v',h)}{\partial \theta} \\
&= \sum_{v',h} \mathbb{P}(v' \mid h) \frac{\partial E(v',h)}{\partial \theta} \\
&= \sum_{v'} \sum_h \mathbb{P}(v') \mathbb{P}(h \mid v') \frac{\partial E(v',h)}{\partial \theta} \\
&= \sum_{v'} \mathbb{P}(v') \sum_h \mathbb{P}(h \mid v') \frac{\partial E(v',h)}{\partial \theta}
\end{aligned}
$$

We recall that $\partial_{a_i} E(v,h) = -v_i$ and $\partial_{b_j} E(v,h) = -h_j$. It follows that:

$$
\begin{aligned}
\frac{\partial \ell(\theta \mid v)}{\partial a_i} &= -\sum_h \mathbb{P}(h \mid v) \frac{\partial E(v,h)}{\partial a_i} + \sum_{v'} \mathbb{P}(v') \sum_h \mathbb{P}(h \mid v') \frac{\partial E(v',h)}{\partial a_i} \\
&= \sum_h \mathbb{P}(h \mid v) v_i - \sum_{v'} \mathbb{P}(v') \sum_h \mathbb{P}(h \mid v') v'_i \\
&= v_i - \sum_{v'} \mathbb{P}(v') v'_i
\end{aligned}
$$

and[27]:

$$
\begin{aligned}
\frac{\partial \ell(\theta \mid v)}{\partial b_j} &= \sum_h \mathbb{P}(h \mid v) h_j - \sum_{v'} \mathbb{P}(v') \sum_h \mathbb{P}(h \mid v') h_j \\
&= \mathbb{P}(h_j = 1 \mid v) - \sum_{v'} \mathbb{P}(v') \mathbb{P}(h_j = 1 \mid v')
\end{aligned}
$$

For the gradient with respect to W, we have $\partial_{w_{i,j}} E(v,h) = -v_i h_j$ and:

$$
\begin{aligned}
\frac{\partial \ell(\theta \mid v)}{\partial w_{i,j}} &= \sum_h \mathbb{P}(h \mid v) v_i h_j - \sum_{v'} \mathbb{P}(v') \sum_h \mathbb{P}(h \mid v') v'_i h_j \\
&= \mathbb{P}(h_j = 1 \mid v) v_i - \sum_{v'} \mathbb{P}(v') \mathbb{P}(h_j = 1 \mid v') v'_i
\end{aligned}
$$

[26] Using the Bayes theorem, the conditional probability distribution $\mathbb{P}(h \mid v)$ is equal to:

$$
\mathbb{P}(h \mid v) = \frac{\mathbb{P}(v,h)}{\mathbb{P}(v)} = \frac{e^{-E(v,h)}}{\sum_{h'} e^{-E(v,h')}}
$$

[27] We use Equation (23) on page 51.

We notice that we have to sum over 2^m possible combinations of the visible variables when calculating the second term $\nabla_\theta^{(2)}(v)$. Therefore, we generally approximate the expectation $\mathbb{P}(v')$ by sampling from the model distribution.

A.2.4 Gradient of the contrastive divergence

The contrastive divergence function is equal to:

$$\mathrm{CD}^{(k)} = \mathrm{KL}\left(\mathbb{P}^{(0)} \parallel \mathbb{P}^{(\infty)}\right) - \mathrm{KL}\left(\mathbb{P}^{(k)} \parallel \mathbb{P}^{(\infty)}\right)$$

where:

$$\begin{aligned}\mathrm{KL}(\mathbb{P} \parallel \mathbb{Q}) &= \sum_v \mathbb{P}(v)\log\left(\frac{\mathbb{P}(v)}{\mathbb{Q}(v)}\right) \\ &= \sum_v \mathbb{P}(v)\log\mathbb{P}(v) - \sum_v \mathbb{P}(v)\log\mathbb{Q}(v)\end{aligned}$$

We have:

$$\begin{aligned}\frac{\partial\,\mathrm{KL}(\mathbb{P} \parallel \mathbb{Q})}{\partial\,\theta} &= \sum_v \frac{\partial\,\mathbb{P}(v)}{\partial\,\theta}\log\mathbb{P}(v) + \sum_v \mathbb{P}(v)\frac{\partial\,\log\mathbb{P}(v)}{\partial\,\theta} - \\ & \sum_v \frac{\partial\,\mathbb{P}(v)}{\partial\,\theta}\log\mathbb{Q}(v) - \sum_v \mathbb{P}(v)\frac{\partial\,\log\mathbb{Q}(v)}{\partial\,\theta}\end{aligned} \tag{25}$$

When $\mathbb{P}$ does not depend on the parameter set θ, we have $\partial_\theta \mathbb{P}(v) = 0$ and the derivative reduces to[28]:

$$\frac{\partial\,\mathrm{KL}(\mathbb{P} \parallel \mathbb{Q})}{\partial\,\theta} = -\sum_v \mathbb{P}(v)\frac{\partial\,\log\mathbb{Q}(v)}{\partial\,\theta}$$

Since we have:

$$\sum_v \mathbb{P}(v)\frac{\partial\,\log\mathbb{P}(v)}{\partial\,\theta} = \sum_v \frac{\partial\,\mathbb{P}(v)}{\partial\,\theta}$$

we also notice that another expression of Equation (25) is:

$$\frac{\partial\,\mathrm{KL}(\mathbb{P} \parallel \mathbb{Q})}{\partial\,\theta} = \sum_v \frac{\partial\,\mathbb{P}(v)}{\partial\,\theta}\left(\log\frac{\mathbb{P}(v)}{\mathbb{Q}(v)} + 1\right) - \sum_v \mathbb{P}(v)\frac{\partial\,\log\mathbb{Q}(v)}{\partial\,\theta}$$

Finally, we deduce that:

$$\begin{aligned}\frac{\partial\,\mathrm{CD}^{(k)}}{\partial\,\theta} &= \frac{\partial}{\partial\,\theta}\left(\sum_v \mathbb{P}^{(0)}(v)\log\left(\frac{\mathbb{P}^{(0)}(v)}{\mathbb{P}^{(\infty)}(v)}\right)\right) - \frac{\partial}{\partial\,\theta}\left(\sum_v \mathbb{P}^{(k)}(v)\log\left(\frac{\mathbb{P}^{(k)}(v)}{\mathbb{P}^{(\infty)}(v)}\right)\right) \\ &= -\sum_v \mathbb{P}^{(0)}(v)\frac{\partial\,\log\mathbb{P}^{(\infty)}(v)}{\partial\,\theta} + \sum_v \mathbb{P}^{(k)}(v)\frac{\partial\,\log\mathbb{P}^{(\infty)}(v)}{\partial\,\theta} - \\ & \sum_v \frac{\partial\,\mathbb{P}^{(k)}(v)}{\partial\,\theta}\left(\log\frac{\mathbb{P}^{(k)}(v)}{\mathbb{P}^{(\infty)}(v)} + 1\right)\end{aligned} \tag{26}$$

In Equation (26), it is often possible to compute the exact values for the first two terms, but not for the third term. However, Hinton (2002) showed experimentally that this last term

[28] This is the case when $\mathbb{P}$ is equal to $\mathbb{P}^{(0)}$.

is so small compared the other two terms that it can be ignored. Thus, we can consider the following approximation:

$$\begin{aligned}
\frac{\partial \text{ CD}^{(k)}}{\partial \theta} &\approx \sum_v \mathbb{P}^{(k)}(v) \frac{\partial \log \mathbb{P}^{(\infty)}(v)}{\partial \theta} - \sum_v \mathbb{P}^{(0)}(v) \frac{\partial \log \mathbb{P}^{(\infty)}(v)}{\partial \theta} \\
&= \sum_v \mathbb{P}^{(k)}(v) \frac{\partial \log \mathbb{P}_\theta(v)}{\partial \theta} - \sum_v \mathbb{P}^{(0)}(v) \frac{\partial \log \mathbb{P}_\theta(v)}{\partial \theta} \\
&= \frac{1}{N} \sum_s \left(\frac{\partial \log \mathbb{P}_\theta\left(v_{(s)}^{(k)}\right)}{\partial \theta} - \frac{\partial \log \mathbb{P}_\theta\left(v_{(s)}^{(0)}\right)}{\partial \theta} \right)
\end{aligned}$$

where $v_{(s)}^{(0)}$ is the s^{th} sample of the training set and $v_{(s)}^{(k)}$ is the associated sample after running k steps of Gibbs sampling[29]. By noticing that $\log \mathbb{P}_\theta(v) = \ell(\theta \mid v)$ and using Equation (24), we obtain:

$$\begin{aligned}
\frac{\partial \text{ CD}^{(k)}}{\partial \theta} &= \frac{1}{N} \sum_{s=1}^N \left(\frac{\partial \ell\left(\theta \mid v_{(s)}^{(k)}\right)}{\partial \theta} - \frac{\partial \ell\left(\theta \mid v_{(s)}^{(0)}\right)}{\partial \theta} \right) \\
&= \frac{1}{N} \sum_{s=1}^N \left(\nabla_\theta \left(v_{(s)}^{(k)}\right) - \nabla_\theta \left(v_{(s)}^{(0)}\right) \right) \\
&= \frac{1}{N} \sum_{s=1}^N \left(\nabla_\theta^{(1)} \left(v_{(s)}^{(k)}\right) - \nabla_\theta^{(1)} \left(v_{(s)}^{(0)}\right) \right) \\
&= \frac{1}{N} \sum_{s=1}^N \sum_h \left(\mathbb{P}\left(h \mid v_{(s)}^{(0)}\right) \frac{\partial E\left(v_{(s)}^{(0)}, h\right)}{\partial \theta} - \mathbb{P}\left(h \mid v_{(s)}^{(k)}\right) \frac{\partial E\left(v_{(s)}^{(k)}, h\right)}{\partial \theta} \right)
\end{aligned}$$

Therefore, we can compute the derivatives with respect to the parameters a_i, b_j and $w_{i,j}$:

$$\begin{aligned}
\frac{\partial \text{ CD}^{(k)}}{\partial a_i} &= \frac{1}{N} \sum_{s=1}^N \left(v_{(s),i}^{(k)} - v_{(s),i}^{(0)} \right) \\
\frac{\partial \text{ CD}^{(k)}}{\partial b_j} &= \frac{1}{N} \sum_{s=1}^N \left(\mathbb{P}\left(h_j = 1 \mid v_{(s)}^{(k)}\right) - \mathbb{P}\left(h_j = 1 \mid v_{(s)}^{(0)}\right) \right) \\
\frac{\partial \text{ CD}^{(k)}}{\partial w_{i,j}} &= \frac{1}{N} \sum_{s=1}^N \left(\mathbb{P}\left(h_j = 1 \mid v_{(s)}^{(k)}\right) \cdot v_{(s),i}^{(k)} - \mathbb{P}\left(h_j = 1 \mid v_{(s)}^{(0)}\right) \cdot v_{(s),i}^{(0)} \right)
\end{aligned}$$

A.2.5 Gradient of the Gaussian-Bernoulli RBM log-likelihood function

By updating Equation (24), we can easily show that the gradient of the log-likelihood function is equal to:

$$\frac{\partial \ell(\theta \mid v)}{\partial \theta} = -\sum_h \mathbb{P}(h \mid v) \frac{\partial E_g(v, h)}{\partial \theta} + \int_{v'} p(v') \sum_h \mathbb{P}(h \mid v') \frac{\partial E_g(v', h)}{\partial \theta} \, \mathrm{d}v' \tag{27}$$

where:

$$E_g(v, h) = \sum_{i=1}^m \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^n b_j h_j - \sum_{i=1}^m \sum_{j=1}^n w_{i,j} \frac{v_i h_j}{\sigma_i^2}$$

[29] In this case, $v_{(s)}^{(0)}$ is the starting value of the Gibbs sampler.

We have:

$$\begin{aligned}
\frac{\partial E_g(v,h)}{\partial a_i} &= \frac{v_i - a_i}{\sigma_i^2} \\
\frac{\partial E_g(v,h)}{\partial b_j} &= -h_j \\
\frac{\partial E_g(v,h)}{\partial w_{i,j}} &= -\frac{v_i h_j}{\sigma_i^2} \\
\frac{\partial E_g(v,h)}{\partial \sigma_i} &= -\frac{(v_i - a_i)^2}{\sigma_i^3} + v_i \sum_j w_{i,j} \frac{h_j}{2\sigma_i^3}
\end{aligned}$$

We deduce that the derivative of $\ell(\theta \mid v)$ with respect to the weight $w_{i,j}$ is given by:

$$\begin{aligned}
\frac{\partial \ell(\theta \mid v)}{\partial w_{i,j}} &= \sum_h \mathbb{P}(h \mid v) \frac{v_i h_j}{\sigma_i^2} - \int_{v'} p(v') \sum_h \mathbb{P}(h \mid v') \frac{v_i' h_j}{\sigma_i^2} \, \mathrm{d}v' \\
&= \mathbb{P}(h_j = 1 \mid v) \frac{v_i}{\sigma_i^2} - \int_{v'} p(v') \, \mathbb{P}(h_j = 1 \mid v') \frac{v_i'}{\sigma_i^2} \, \mathrm{d}v'
\end{aligned}$$

Similarly, we can compute the other derivatives. For a_i and b_j, we obtain:

$$\frac{\partial \ell(\theta \mid v)}{\partial a_i} = -\frac{v_i}{\sigma_i^2} + \int_{v'} p(v') \frac{v_i'}{\sigma_i^2} \, \mathrm{d}v'$$

and:

$$\frac{\partial \ell(\theta \mid v)}{\partial b_j} = \mathbb{P}(h_j = 1 \mid v) - \int_{v'} p(v') \, \mathbb{P}(h_j = 1 \mid v') \, \mathrm{d}v'$$

Finally, we obtain for the parameter σ_i:

$$\begin{aligned}
\frac{\partial \ell(\theta \mid v)}{\partial \sigma_i} &= -\frac{(v_i - a_i)^2}{\sigma_i^3} + \frac{v_i}{2\sigma_i^3} \sum_j w_{i,j} \mathbb{P}(h_j = 1 \mid v) + \\
&\quad \int_{v'} p(v') \left(\frac{(v_i' - a_i)^2}{\sigma_i^3} - \frac{v_i'}{2\sigma_i^3} \sum_j w_{i,j} \mathbb{P}(h_j = 1 \mid v') \right) \mathrm{d}v'
\end{aligned}$$

A.2.6 Gradient of the conditional RBM log-likelihood function

The log-likelihood function for the conditional RBM is:

$$\ell(\theta \mid v_t) = \log p_\theta(v_t \mid c_t)$$

where the set of parameters becomes $\theta = (a, b, W, P, Q)$. We can then compute the partial derivative of the energy function by using the chain rule:

$$\begin{aligned}
\frac{\partial \tilde{E}_g(v_t, h_t, c_t)}{\partial q_{k,i}} &= \frac{\partial \tilde{E}_g(v_t, h_t, c_t)}{\partial \tilde{a}_t} \cdot \frac{\partial \tilde{a}_t}{\partial q_{k,i}} \\
&= \left(\frac{v_{t,i} - \tilde{a}_{t,i}}{\sigma_i^2} \right) c_{t,k}
\end{aligned}$$

and we obtain:

$$\frac{\partial \ell(\theta \mid v_t)}{\partial q_{k,i}} = -\left(\frac{v_{t,i} - \tilde{a}_{t,i}}{\sigma_i^2} \right) c_{t,k} + \int_{v'} p(v') \left(\frac{v_{t,i}' - \tilde{a}_{t,i}}{\sigma_i^2} \right) c_{t,k} \, \mathrm{d}v'$$

Similarly, we have:

$$\begin{aligned}\frac{\partial\,\tilde{E}_g\left(v_t,h_t,c_t\right)}{\partial\,p_{k,j}} &= \frac{\partial\,\tilde{E}_g\left(v_t,h_t,c_t\right)}{\partial\,\tilde{b}_t}\cdot\frac{\partial\,\tilde{b}_t}{\partial\,p_{k,j}} \\ &= -h_{t,j}c_{t,k}\end{aligned}$$

and:

$$\begin{aligned}\frac{\partial\,\ell\left(\theta\mid v_t\right)}{\partial\,p_{k,j}} &= \mathbb{P}\left(h_{t,j}=1\mid v_t,c_t\right)c_{t,k} - \\ &\quad \int_{v'} p\left(v'_t\mid c_t\right)\mathbb{P}\left(h_j=1\mid v'_t,c_t\right)c_{t,k}\,\mathrm{d}v'\end{aligned}$$

The calculation for the other derivatives remains unchanged and are the same as those obtained for the Gaussian-Bernoulli RBM.

A.3 A unified approach of GAN models

Generative models are trained to perform a mapping from a latent space to some specified data manifold, which is generally represented by the empirical distribution of real data. The problem consists then in finding a mapping function that best matches the target data in the sense of a certain discrepancy measure. For comparing the theoretical distribution with the empirical distribution, the two important metrics used in the context of generative modeling are divergence measures (ϕ-divergence) and integral probability metrics (IPM), which give two different families of GANs. In this section, we show that all models share many significant common points. Let $\mathcal{X}$ be a topological space that is compact, complete and separable. Let us assume that there are two probability measures $\mathbb{P}$ and $\mathbb{Q}$ which can be defined on $\mathcal{X}$. GAN optimization relies on the fact that both ϕ-divergence and IPM can be written as:

$$d_{\mathcal{F}}(\mathbb{P},\mathbb{Q}) = \sup_{\varphi\in\mathcal{F}} \left|\Delta_{\mathbb{P},\mathbb{Q}}(\varphi)\right|$$

where $\mathcal{F}$ is the class of functions φ defined on $\mathcal{X}$ and $\Delta : \mathcal{F} \to \mathcal{X}$ is a discrepancy operator. In the following, we show that the choice of the class $\mathcal{F}$ and the discrepancy operator Δ lead to different GAN models.

A.3.1 ϕ-GAN models

Variational estimation of ϕ-divergences The very first model of GAN is often presented as a minmax optimization problem (Goodfellow *et al.*, 2014). However, it is possible to find a direct correspondence between Goodfellow's saddle point problem and divergence minimization. Let us recall the definition of a divergence measure. We assume that there are two probability measures $\mathbb{P}$ and $\mathbb{Q}$ that can be defined on $\mathcal{X}$. Moreover, $\mathbb{P}$ must be absolutely continuous[30] with respect to $\mathbb{Q}$, which is denoted $\mathbb{P} \ll \mathbb{Q}$. The ϕ-divergence D_ϕ is defined as:

$$D_\phi(\mathbb{P}\parallel\mathbb{Q}) = \int_{\mathcal{X}} \phi\left(\frac{\mathrm{d}\mathbb{P}(x)}{\mathrm{d}\mathbb{Q}(x)}\right)\mathbb{Q}(\mathrm{d}x) \tag{28}$$

where $\phi : \mathbb{R}^+ \to \mathbb{R}\cup\{+\infty\}$ is a convex, lower-semi-continuous function such that[31] $\phi(1) = 0$. Looking at the closed-form solution of ϕ-divergence, we note that it can be interpreted as the likelihood ratio between two probability distributions[32].

Remark 7. *Different divergence measures can be used for modeling the function ϕ:*

- *the Kullback-Leibler divergence D_{KL} corresponds to $\phi(t) = t\log(t)$;*
- *the Jensen-Shannon divergence D_{JS} is obtained by setting $\phi(t) = -(t+1)\log\left(\frac{1+t}{2}\right) + t\log(t)$;*
- *the total variation (or energy-based) divergence D_{TV} is defined by taking $\phi(t) = \frac{1}{2}|t-1|$.*

In order to establish the link with GAN optimization problems, Nguyen *et al.* (2010) proposed to compute a variational characterization of these ϕ-divergence measures by looking at the convex dual. For that, we need to introduce the Fenchel conjugate, which is a

[30] This means that if we consider a σ-field $\mathcal{A}\subseteq\mathcal{X}$ such that $\mathbb{Q}(\mathcal{A}) = 0$, then $\mathbb{P}(\mathcal{A}) = 0$.

[31] This last condition ensures that $D_\phi(\mathbb{P}\parallel\mathbb{Q}) = 0$ if $\mathbb{P} = \mathbb{Q}$.

[32] The condition that imposes that $\mathbb{P}$ must be absolutely continuous with respect to $\mathbb{Q}$ is related to the Radon-Nikodym theorem. It states that if $\mathbb{P} \ll \mathbb{Q}$, then there is a function $f : \mathcal{X} \to \mathbb{R}$ that satisfies $\mathbb{P}(A) = \int_A f(x)\,\mathbb{Q}(\mathrm{d}x)$ for all σ-field $\mathcal{A}$. The function f is often denoted by $\dfrac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$.

fundamental tool in convex analysis (Barbu and Precupanu, 2012). Let us consider a function $f : \mathcal{X} \to \mathbb{R} \cup \{+\infty\}$. According to the Riesz representation theorem, it is possible to identify the dual space $\mathcal{X}^*$ of the Banach space $\mathcal{X}$. Therefore, we can work on the product space $\mathcal{X}^* \times \mathcal{X}$ associated with the scalar product $\langle x^*, x\rangle$. The Fenchel transform is then defined on the dual space such that:

$$f^* : \left\{ \begin{array}{l} \mathcal{X}^* \to \mathbb{R} \\ x^* \to \sup_{x \in \operatorname{dom} f} \left\{\langle x^*, x\rangle - f(x)\right\} \end{array} \right.$$

where $\operatorname{dom} f = \{x \in \mathcal{X} \mid f(x) < +\infty\}$. According to the Fenchel-Moreau theorem, if f is convex and continuous, then $f^{**} = f^* \circ f^* = f$ and we obtain:

$$f(x) = \sup_{x^* \in \operatorname{dom} f^*} \left\{\langle x, x^*\rangle - f^*(x^*)\right\}$$

for all $x \in \mathcal{X}$. If we consider the function ϕ associated with a given ϕ-divergence, the space $\mathcal{X}$ is $\mathbb{R}$ and the dual space is also $\mathbb{R}$. In order to express the ϕ-divergence in terms of loss, Nguyen *et al.* (2010) simply expressed ϕ in term of its conjugate:

$$\phi(x) = \sup_{x^* \in \operatorname{dom} \phi^*} \left\{\langle x, x^*\rangle - \phi^*(x^*)\right\}$$

Using Jensen inequality and considering that ϕ^* is convex, it follows that:

$$\begin{aligned} D_\phi(\mathbb{P} \parallel \mathbb{Q}) &= \int_{\mathcal{X}} \phi\left(\frac{\mathrm{d}\mathbb{P}(x)}{\mathrm{d}\mathbb{Q}(x)}\right) \mathbb{Q}(\mathrm{d}x) \\ &= \int_{\mathcal{X}} \left(\sup_{x^* \in \operatorname{dom}\phi^*} \left\{ \left\langle \frac{\mathrm{d}\mathbb{P}(x)}{\mathrm{d}\mathbb{Q}(x)}, x^* \right\rangle - \phi^*(x^*) \right\} \mathrm{d}\mathbb{Q}(x) \right) \mathrm{d}x \\ &\geq \sup_{x^* \in \operatorname{dom}\phi^*} \left\{ \int_{\mathcal{X}} \left(x^* \, \mathrm{d}\mathbb{P}(x) - \phi^*(x^*) \, \mathrm{d}\mathbb{Q}(x)\right) \mathrm{d}x \right\} \end{aligned}$$

We then introduce a class of functions $\mathcal{F}$ that maps $\mathcal{X}$ to $\operatorname{dom}\phi^*$:

$$\begin{aligned} D_\phi(\mathbb{P} \parallel \mathbb{Q}) &\geq \sup_{\varphi \in \mathcal{F}} \left\{ \int_{\mathcal{X}} \varphi(x) \, \mathbb{P}(\mathrm{d}x) - \int_{\mathcal{X}} \phi^*(\varphi(x)) \, \mathbb{Q}(\mathrm{d}x) \right\} \\ &\geq \sup_{\varphi \in \mathcal{F}} \left\{ \mathbb{E}[\varphi(X) \mid X \sim \mathbb{P}] - \mathbb{E}[\phi^*(\varphi(X)) \mid X \sim \mathbb{Q}] \right\} \end{aligned}$$

where $\{\mathcal{F} = \varphi : \mathcal{X} \to \mathbb{R} \mid \phi(\mathcal{X}) \subseteq \operatorname{dom}\phi^*\}$. We define the discrepancy operator for this specific model as follows:

$$\Delta_{\mathbb{P},\mathbb{Q}}(\varphi) = \mathbb{E}[\varphi(X) \mid X \sim \mathbb{P}] - \mathbb{E}[\phi^*(\varphi(X)) \mid X \sim \mathbb{Q}] \tag{29}$$

To find the optimal function such that equality in the supremum is obtained, we introduce the notation $\tilde{x} = \varphi(x)$ and the quantity $\mathcal{C}(\tilde{x})$ defined by:

$$\begin{aligned} \mathcal{C}(\tilde{x}) &= C(\varphi(x)) \\ &= \mathbb{E}[\varphi(X) \mid X \sim \mathbb{P}] - \mathbb{E}[\phi^*(\varphi(X)) \mid X \sim \mathbb{Q}] \end{aligned} \tag{30}$$

By computing the derivative $\mathcal{C}(\tilde{x})$, Nowozin *et al.* (2016) found that the optimal function $\varphi^\star$ is[33]:

$$\varphi^\star(x) = \phi'\left(\frac{\mathrm{d}\mathbb{P}(x)}{\mathrm{d}\mathbb{Q}(x)}\right) \tag{31}$$

[33] See Broniatowski and Keziou (2006, Theorem 4.4) for a formal proof.

However, evaluating this quantity is impossible because the distribution function $\mathbb{P}$ is unknown. Therefore, φ should be flexible enough to approximate the derivative ϕ' everywhere. This is why GAN models use deep neural networks to estimate it. This leads us to introduce the parameter θ_d that will be optimized during the training process. In this context, we write the parameterized function $\boldsymbol{\mathcal{D}}(x, \theta_d)$, which aims to estimate the function $\varphi^\star(x)$:

$$\hat{\varphi}^\star(x) = \boldsymbol{\mathcal{D}}(x, \theta_d)$$

Consequently, Nowozin *et al.* (2016) proposed to use the resulting lower bound in order to train GANs. $\boldsymbol{\mathcal{G}}(z, \theta_g)$ represents the generative model that is also a neural network and allows us to build the probability distribution $\mathbb{P}_{\text{model}}$, which should estimate the given probability distribution $\mathbb{P}_{\text{data}}$. Thus, we obtain the saddle point problem:

$$\min_{\phi} \Delta_{\mathbb{P}_{\text{data}}, \mathbb{P}_{\text{model}}}(\varphi^\star) \approx \min_{\theta_g} \max_{\theta_d} \mathcal{C}(\boldsymbol{\mathcal{D}}(X, \theta_d)) \tag{32}$$

where:

$$\mathcal{C}(\boldsymbol{\mathcal{D}}(X, \theta_d)) = \mathbb{E}[\boldsymbol{\mathcal{D}}(X, \theta_d) \mid X \sim \mathbb{P}_{\text{data}}] - \mathbb{E}[\phi^*(\boldsymbol{\mathcal{D}}(X, \theta_d)) \mid X \sim \boldsymbol{\mathcal{G}}(z, \theta_g)] \tag{33}$$

$\boldsymbol{\mathcal{D}}(x, \theta_d) \in \mathcal{F}$ and $\mathbb{P}_{\text{model}} = \boldsymbol{\mathcal{G}}(z, \theta_g)$. Therefore, we have constrained the generator to be in a smaller class of functions belonging to $\mathcal{F}$ even if the neural network can approximate any function.

Remark 8. *In order to obtain the minimax problem of Goodfellow et al. (2014), Nowozin et al. (2016) considered the function:*

$$\phi(x) = x \log x - (x+1)\log(x+1)$$

We deduce that:

$$\phi'(x) = \log\left(\frac{x}{x+1}\right)$$

and[34]*:*

$$\phi^*(x) = -\log(1 - e^x)$$

[34] We have:

$$\phi^*(t) = \sup_{x \in \operatorname{dom} \phi} \langle x, t \rangle - \phi(x)$$

It follows that $h(x) = xt - x\log x + (x+1)\log(x+1)$ and:

$$h'(x) = t - \log\left(\frac{x}{x+1}\right)$$

The supremum is reached at the point $h'(x^\star) = 0$, implying that:

$$x^\star = \frac{e^t}{1 - e^t}$$

Finally, we obtain:

$$\begin{aligned}
\phi^*(t) &= t\left(\frac{e^t}{1-e^t}\right) - \left(\frac{e^t}{1-e^t}\right)\log\left(\frac{e^t}{1-e^t}\right) + \left(\frac{1}{1-e^t}\right)\log\left(\frac{1}{1-e^t}\right) \\
&= t\left(\frac{e^t}{1-e^t}\right) - \left(\frac{e^t}{1-e^t}\right)\left(t - \log\left(1-e^t\right)\right) - \left(\frac{1}{1-e^t}\right)\log\left(1-e^t\right) \\
&= \left(\frac{e^t}{1-e^t}\right)\log\left(1-e^t\right) - \left(\frac{1}{1-e^t}\right)\log\left(1-e^t\right) \\
&= -\log\left(1-e^t\right)
\end{aligned}$$

Using Equation (30), we deduce that:

$$\mathcal{C}(\tilde{x}) = \mathbb{E}\left[\varphi(X) \mid X \sim \mathbb{P}\right] + \mathbb{E}\left[\log\left(1 - \exp\left(\varphi(X)\right)\right) \mid X \sim \mathbb{Q}\right]$$

If we set $\varphi(X) = \log \boldsymbol{\mathcal{D}}(X, \theta_d)$, we find the minimax function of Goodfellow et al. (2014):

$$\mathcal{C}\left(\boldsymbol{\mathcal{D}}(X, \theta_d)\right) = \mathbb{E}\left[\log \boldsymbol{\mathcal{D}}(X, \theta_d) \mid X \sim \mathbb{P}_{\text{data}}\right] + \mathbb{E}\left[\log\left(1 - \boldsymbol{\mathcal{D}}(X, \theta_d)\right) \mid X \sim \boldsymbol{\mathcal{G}}(z, \theta_g)\right]$$

Another representation of ϕ-divergences GAN training can be viewed as a process of successively estimating the optimal function φ and minimizing the ϕ-divergence. Chu *et al.* (2019) proposed a more general view of this process. Previously, the study has been done on the space $\mathbb{R}$ thanks to the dual form of the given function ϕ associated with the ϕ-divergence. Alternatively, Chu *et al.* (2019) proposed to directly work on the probability space $\mathcal{X}$. This imposes to define a probability functional $\mathcal{J}_{\mathbb{P}}(\mathbb{Q}) : \mathcal{B}(\mathcal{X}) \longrightarrow \mathbb{R}$, where $\mathcal{B}(\mathcal{X})$ is the space of Borel probability measures on $\mathcal{X}$. In the context of GAN optimization problems, we would like to show that:

$$\min D_\phi(\mathbb{P} \parallel \mathbb{Q}) = \min_{\mathbb{Q} \in \mathcal{B}(\mathcal{X})} \mathcal{J}_{\mathbb{P}}(\mathbb{Q}) \tag{34}$$

However, contrary to the previous case, probability functionals take value in probability spaces, where functional derivatives need to be defined. Moreover, the dimension of the space is potentially infinite. Therefore, the difficulty is to transform functional optimization into a convex optimization problem that can be solved using traditional numerical algorithms such as the gradient descent.

In order to define the derivative in $\mathcal{B}(\mathcal{X})$, Chu *et al.* (2019) used the Gâteaux derivative of the functional $\mathcal{J}_{\mathbb{P}}$ at $\mathbb{Q}$ in the direction of $\delta = \mathbb{P} - \mathbb{Q}$:

$$\begin{aligned} \mathcal{J}'_{\mathbb{P}}(\mathbb{P} - \mathbb{Q}) &= \mathrm{d}\mathcal{J}_{\mathbb{P}}(\mathbb{Q}, \delta) \\ &= \left.\frac{\mathrm{d}}{\mathrm{d}\epsilon} \mathcal{J}_{\mathbb{P}}(\mathbb{Q} + \epsilon\delta)\right|_{\epsilon=0} \\ &= \lim_{\epsilon \to 0} \frac{\mathcal{J}_{\mathbb{P}}(\mathbb{Q} + \epsilon\delta) - \mathcal{J}_{\mathbb{P}}(\mathbb{Q})}{\epsilon} \end{aligned} \tag{35}$$

This definition initially comes from Von Mises calculus, and the Gâteaux derivative is also called the '*Volterra*' derivative (Fernholz, 2012). Chu *et al.* (2019) recalled that the Gâteaux derivative has an integral representation $\mathcal{J}'_{\mathbb{P}}(\delta) = \int_{\mathcal{X}} g(x)\, \delta(\mathrm{d}x)$ where the function $g : \mathcal{X} \rightarrow \mathbb{R}$ is called the '*influence function*' or the influence curve[35]. It follows that:

$$\begin{aligned} \mathcal{J}'_{\mathbb{P}}(\mathbb{P} - \mathbb{Q}) &= \int_{\mathcal{X}} g(x)\, (\mathbb{P} - \mathbb{Q})(\mathrm{d}x) \\ &= \mathbb{E}\left[g(X) \mid X \sim \mathbb{P}\right] - \mathbb{E}\left[g(X) \mid X \sim \mathbb{Q}\right] \end{aligned} \tag{36}$$

We are now able to compute the derivative in order to recover the optimal function that satisfy the minimum of the ϕ-divergence:

$$\begin{aligned} \mathcal{J}'_{\mathbb{P}}(\mathbb{P} - \mathbb{Q}) &= \int_{\mathcal{X}} \left.\frac{\mathrm{d}}{\mathrm{d}\epsilon} \phi\left(\frac{\mathrm{d}\mathbb{Q}(x) + \epsilon\left(\mathrm{d}\mathbb{P}(x) - \mathrm{d}\mathbb{Q}(x)\right)}{\mathrm{d}\mathbb{P}(x)}\right)\right|_{\epsilon=0} \mathrm{d}\mathbb{P}(x) \\ &= \int_{\mathcal{X}} \left.\phi'\left(\frac{\mathrm{d}\mathbb{Q}(x) + \epsilon\left(\mathrm{d}\mathbb{P}(x) - \mathrm{d}\mathbb{Q}(x)\right)}{\mathrm{d}\mathbb{P}(x)}\right)\right|_{\epsilon=0} \frac{\left(\mathrm{d}\mathbb{P}(x) - \mathrm{d}\mathbb{Q}(x)\right)}{\mathrm{d}\mathbb{P}(x)} \mathrm{d}\mathbb{P}(x) \\ &= \int_{\mathcal{X}} \phi'\left(\frac{\mathrm{d}\mathbb{Q}(x)}{\mathrm{d}\mathbb{P}(x)}\right) (\mathbb{P} - \mathbb{Q})(\mathrm{d}x) \end{aligned} \tag{37}$$

[35] The influence function is not unique. Indeed, for any function g that describes the Gâteaux differential at $\mathbb{Q}$, $g + c$ also works. Thus, the influence function is uniquely defined up to an arbitrary additive constant.

We deduce that the influence function for the ϕ-divergence is equal to:

$$g_\phi : \left\{ \begin{array}{l} \mathcal{X} \to \mathbb{R} \\ x \to \phi' \left(\dfrac{\mathrm{d}\mathbb{Q}(x)}{\mathrm{d}\mathbb{P}(x)} \right) \end{array} \right. \tag{38}$$

The influence function can then be associated with the optimal function ϕ introduced before because we have $g_\phi = \varphi^\star \in \mathcal{F}$. We conclude that the GAN discriminator will try to estimate the influence function. While Nguyen *et al.* (2010) focused on the dual form of the function ϕ, Chu *et al.* (2019) used a more general approach by considering the dual form of the probability functional in order to recover the Goodfellow's saddle point problem.

In the case of probability functionals which take values in $\mathcal{B}(\mathcal{X})$, the dual space can be defined as the space $L^1(\mathcal{X})$ of all real-valued Lipschitz function that takes values in $\mathcal{X}$ (Laschos *et al.*, 2019). According to the Riesz representation theorem, it is possible to identify the space $\mathcal{B}(\mathcal{X})$ to it dual. We consider the product space $\mathcal{B}(\mathcal{X}) \times L^1(\mathcal{X})$ with the scalar product defined as:

$$\begin{aligned} \langle \varphi, \mathbb{Q} \rangle &= \int_{\mathcal{X}} \varphi(x) \; \mathbb{Q}(\mathrm{d}x) \\ &= \int_{\mathcal{X}} \varphi(x) \; \mathrm{d}\mathbb{Q}(x) \; \mathrm{d}x \end{aligned}$$

where $\varphi \in L^1(\mathcal{X})$ and $\mathbb{Q} \in \mathcal{B}(\mathcal{X}) \times \mathbb{R}$. Thus, we can rewrite $\mathcal{J}_{\mathbb{P}}$ in term of its convex transform:

$$\begin{aligned} \mathcal{J}_{\mathbb{P}}(\mathbb{Q}) &= \sup_{\varphi \in L^1(\mathcal{X})} \left\{ \langle \varphi, \mathbb{Q} \rangle - \mathcal{J}^*_{\mathbb{P}}(\varphi) \right\} \\ &= \sup_{\varphi \in L^1(\mathcal{X})} \left\{ \int_{\mathcal{X}} \varphi(x) \; \mathbb{Q}(\mathrm{d}x) - \mathcal{J}^*_{\mathbb{P}}(\varphi) \right\} \\ &= \sup_{\varphi \in L^1(\mathcal{X})} \left\{ \mathbb{E}\left[\varphi(X) \mid X \sim \mathbb{Q} \right] - \mathcal{J}^*_{\mathbb{P}}(\varphi) \right\} \end{aligned} \tag{39}$$

To bridge the gap between the two approaches, we have to demonstrate that $\mathcal{J}^*_{\mathbb{P}}(\varphi) = \mathbb{E}\left[\phi^*(\varphi(X)) \mid X \sim \mathbb{P}\right]$ for a given estimate of the function $\varphi \in \mathcal{F}$. For that, Chu *et al.* (2019) considered the Fenchel conjugate of the Jensen-Shannon divergence[36]:

$$\begin{aligned} \mathcal{J}^*_{\mathbb{P}}(\varphi) &= \mathcal{J}^*_{\mathrm{JS}}(\varphi) \\ &= -\frac{1}{2} \mathbb{E}\left[\log\left(1 - e^{2\varphi(X) - \log 2}\right) \mid X \sim \mathbb{P} \right] - \frac{1}{2}\log 2 \end{aligned} \tag{40}$$

Using $\varphi(x) = \frac{1}{2}\log\left(1 - \boldsymbol{\mathcal{D}}(x, \theta_d)\right) + \frac{1}{2}\log 2$, we obtain:

$$\begin{aligned} \mathbb{E}\left[\varphi(X) \mid X \sim \mathbb{Q}\right] - \mathcal{J}^*_{\mathbb{P}}(\varphi) &= \frac{1}{2}\mathbb{E}\left[\log\left(1 - \boldsymbol{\mathcal{D}}(x, \theta_d)\right) \mid X \sim \mathbb{Q}\right] + \\ & \quad \frac{1}{2}\mathbb{E}\left[\log \boldsymbol{\mathcal{D}}(x, \theta_d) \mid X \sim \mathbb{P}\right] + \log 2 \end{aligned} \tag{41}$$

Therefore, the descent algorithm applied to probability functionals is equivalent to the Goodfellow's saddle point problem:

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}\left[\log\left(\boldsymbol{\mathcal{D}}(X, \theta_d)\right) \mid X \sim \mathbb{P}_{\mathrm{data}}\right] + \mathbb{E}\left[\log\left(1 - \boldsymbol{\mathcal{D}}(X, \theta_d)\right) \mid X \sim \boldsymbol{\mathcal{G}}(z, \theta_g)\right]$$

[36] The derivation of this result is given in Appendix A.4 on page 64.

A.3.2 IPM-GAN models

In the case of Wasserstein generative adversarial networks introduced by Arjovsky *et al.* (2017a,b), ϕ-divergences have to be replaced by integral probability metrics (IPMs) in order to compare two different probability distributions. Let $\mathcal{F}$ be a class of functions defined on $\mathcal{X}$. Müller (1997) defined an IPM $I_{\mathcal{F}}$ between $\mathbb{P}$ and $\mathbb{Q}$ in the following way:

$$\begin{aligned} I_{\mathcal{F}}(\mathbb{P},\mathbb{Q}) &= \sup_{\varphi\in\mathcal{F}}\left\{\left|\int_{\mathcal{X}}\varphi(x)\ \mathbb{P}(\mathrm{d}x)-\int_{\mathcal{X}}\varphi(x)\ \mathbb{Q}(\mathrm{d}x)\right|\right\} \\ &= \sup_{\varphi\in\mathcal{F}}\left\{\left|\mathbb{E}\left[\varphi(X)\mid X\sim\mathbb{P}\right]-\mathbb{E}\left[\varphi(X)\mid X\sim\mathbb{Q}\right]\right|\right\} \end{aligned}$$

IPMs are looking for a *critic* function that maximizes the average discrepancy between the two distributions $\mathbb{P}$ and $\mathbb{Q}$. Contrary to ϕ-GAN models, where we have to find the variational form of the divergence $\phi(x)$, the definition of an IPM directly gives the discrepancy operator:

$$\Delta_{\mathbb{P},\mathbb{Q}}(\varphi)=\mathbb{E}\left[\varphi(X)\mid X\sim\mathbb{P}\right]-\mathbb{E}\left[\varphi(X)\mid X\sim\mathbb{Q}\right]$$

for all $\varphi\in\mathcal{F}$. The Wasserstein GAN proposed by Arjovsky *et al.* (2017a,b) considers the function class $\mathcal{F}$ such that $\varphi(x)$ is a 1-Lipschitz function. In order to present different choices for the function class $\mathcal{F}$, we consider the Lebesgue norm on the measurable space $\Omega=(\mathcal{X},\mathbb{P})$: $\|f\|_2^2=\int_{\mathcal{X}}f^2(x)\ \mathbb{P}(\mathrm{d}x)$. Let us denote the normed space by $L^2(\Omega)=\{f:\mathcal{X}\to\mathbb{R}\mid\|f\|_2<+\infty\}$ and the unit ball by $\mathcal{B}_1=\left\{f\in L^2(\Omega)\mid\|f\|_2\leq 1\right\}$. Therefore, choosing the class function such that $\mathcal{F}=\mathcal{B}_1$ is called a Fisher GAN by Mroueh and Sercu (2017). In a similar way, Mroueh *et al.* (2018) proposed to define Sobolev GAN models by considering the following class of functions $\mathcal{F}=\left\{f\in L^2(\Omega)\mid\|\nabla_x f\|_2\leq 1\right\}$.

A.4 The Jensen-Shannon divergence function

To derive the convex conjugate of $\mathcal{J}_{\mathrm{JS}}$, we follow Appendix A given in Chu *et al.* (2019). Let $\mathbb{P}$ and $\mathbb{Q}$ be two probability measures. We denote by $p(x)$ and $q(x)$ the associated density functions $\mathrm{d}\mathbb{P}(x)$ and $\mathrm{d}\mathbb{Q}(x)$. The Jensen-Shannon divergence function is defined by:

$$\begin{aligned} D_{\mathrm{JS}}(\mathbb{P} \parallel \mathbb{Q}) &= \frac{1}{2} D_{\mathrm{KL}}\left(\mathbb{P} \left\| \frac{1}{2}\mathbb{P} + \frac{1}{2}\mathbb{Q}\right.\right) + \frac{1}{2} D_{\mathrm{KL}}\left(\mathbb{Q} \left\| \frac{1}{2}\mathbb{P} + \frac{1}{2}\mathbb{Q}\right.\right) \\ &= \frac{1}{2}\int \left(p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \right) \mathrm{d}x \end{aligned}$$

where $D_{\mathrm{KL}}(\mathbb{P} \parallel \mathbb{Q})$ is the Kullback-Leibler divergence. In the case where $\mathcal{J}_{\mathrm{JS}}(\mathbb{P}) = D_{\mathrm{JS}}(\mathbb{P} \parallel \mathbb{Q})$, we obtain:

$$\begin{aligned} \mathcal{J}_{\mathrm{JS}}(\mathbb{P} + \epsilon\delta) &= \frac{1}{2}\int (p(x) + \epsilon\delta(x)) \log \frac{2(p(x)+\epsilon\delta(x))}{p(x)+q(x)+\epsilon\delta(x)} \,\mathrm{d}x + \\ & \quad \frac{1}{2}\int q(x) \log \frac{2q(x)}{p(x)+q(x)+\epsilon\delta(x)} \,\mathrm{d}x \end{aligned}$$

Since we have:

$$\log \frac{2(p(x)+\epsilon\delta(x))}{p(x)+q(x)+\epsilon\delta(x)} = \log 2 + \log(p(x)+\epsilon\delta(x)) - \log(p(x)+q(x)+\epsilon\delta(x))$$

and:

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon} \log \frac{2(p(x)+\epsilon\delta(x))}{p(x)+q(x)+\epsilon\delta(x)} = \frac{\delta(x)}{p(x)+\epsilon\delta(x)} - \frac{\delta(x)}{p(x)+q(x)+\epsilon\delta(x)}$$

it follows that:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\epsilon} \mathcal{J}_{\mathrm{JS}}(\mathbb{P} + \epsilon\delta) &= \frac{1}{2}\int \frac{\mathrm{d}}{\mathrm{d}\epsilon} \left((p(x)+\epsilon\delta(x)) \log \frac{2(p(x)+\epsilon\delta(x))}{p(x)+q(x)+\epsilon\delta(x)} \right) \mathrm{d}x + \\ & \quad \frac{1}{2}\int \frac{\mathrm{d}}{\mathrm{d}\epsilon} \left(q(x) \log \frac{2q(x)}{p(x)+q(x)+\epsilon\delta(x)} \right) \mathrm{d}x \\ &= \frac{1}{2}\int \delta(x) \log \frac{2(p(x)+\epsilon\delta(x))}{p(x)+q(x)+\epsilon\delta(x)} \,\mathrm{d}x + \\ & \quad \frac{1}{2}\int (p(x)+\epsilon\delta(x)) \frac{\delta(x)}{p(x)+\epsilon\delta(x)} \,\mathrm{d}x - \\ & \quad \frac{1}{2}\int (p(x)+\epsilon\delta(x)) \frac{\delta(x)}{p(x)+q(x)+\epsilon\delta(x)} \,\mathrm{d}x - \\ & \quad \frac{1}{2}\int q(x) \frac{\delta(x)}{p(x)+q(x)+\epsilon\delta(x)} \,\mathrm{d}x \end{aligned}$$

We deduce that:

$$\begin{aligned} \left.\frac{\mathrm{d}}{\mathrm{d}\epsilon} \mathcal{J}_{\mathrm{JS}}(\mathbb{P} + \epsilon\delta)\right|_{\epsilon=0} &= \frac{1}{2}\int \left(\delta(x) \log \frac{2p(x)}{p(x)+q(x)} \right) \mathrm{d}x \\ &= \frac{1}{2}\int \left(\log \frac{p(x)}{p(x)+q(x)} + \log 2 \right) \delta(x) \,\mathrm{d}x \end{aligned}$$

Chu *et al.* (2019) concluded that the influence function of $\mathcal{J}_{\mathrm{JS}}(\mathbb{P})$ is equal to:

$$g_{\mathrm{JS}}(x) = \frac{1}{2} \log \frac{\mathrm{d}\mathbb{P}(x)}{\mathrm{d}\mathbb{P}(x) + \mathrm{d}\mathbb{Q}(x)} + \frac{1}{2} \log 2$$

In order to find the convex conjugate of $\mathcal{J}_{\mathrm{JS}}(\mathbb{P})$, we consider the Fenchel-Moreau theorem:

$$\begin{aligned}\mathcal{J}_{\mathrm{JS}}^{*}(\varphi) &= \sup_{\mathbb{P}\in\mathcal{B}(\mathcal{X})}\left\{\langle\varphi,\mathbb{P}\rangle-\mathcal{J}_{\mathrm{JS}}(\mathbb{P})\right\} \\ &= \sup_{\mathbb{P}\in\mathcal{B}(\mathcal{X})}\left\{\int_{\mathcal{X}}\varphi(x)\;\mathbb{P}(\mathrm{d}x)-\mathcal{J}_{\mathrm{JS}}(\mathbb{P})\right\}\end{aligned}$$

We have:

$$\begin{aligned}f(\mathbb{P}) &= \int_{\mathcal{X}}\varphi(x)\;\mathbb{P}(\mathrm{d}x)-\mathcal{J}_{\mathrm{JS}}(\mathbb{P}) \\ &= \int_{\mathcal{X}}\varphi(x)\,p(x)\;\mathrm{d}x- \\ &\quad \frac{1}{2}\int_{\mathcal{X}}\left(p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}\right)\mathrm{d}x \\ &= \int_{\mathcal{X}}h(x)\;\mathrm{d}x\end{aligned}$$

where:

$$\begin{aligned}h(x) &= \varphi(x)\,p(x)-\frac{1}{2}p(x)\log 2-\frac{1}{2}p(x)\log p(x)+\frac{1}{2}p(x)\log\left(p(x)+q(x)\right)- \\ &\quad \frac{1}{2}q(x)\log 2-\frac{1}{2}q(x)\log q(x)+\frac{1}{2}q(x)\log\left(p(x)+q(x)\right)\end{aligned}$$

and:

$$\begin{aligned}\frac{\partial h(x)}{\partial p(x)} &= \varphi(x)-\frac{1}{2}\log 2-\frac{1}{2}\log p(x)-\frac{1}{2}+\frac{1}{2}\log\left(p(x)+q(x)\right)+ \\ &\quad \frac{1}{2}\frac{p(x)}{p(x)+q(x)}+\frac{1}{2}\frac{q(x)}{p(x)+q(x)} \\ &= \varphi(x)-\frac{1}{2}\log 2-\frac{1}{2}\log p(x)+\frac{1}{2}\log\left(p(x)+q(x)\right)\end{aligned}$$

Since the first-order condition $\partial_{p(x)}h(x)=0$, we deduce that the optimal solution is:

$$\begin{aligned}\varphi(x) &= \frac{1}{2}\log 2+\frac{1}{2}\log\frac{p(x)}{p(x)+q(x)} \\ &= \frac{1}{2}\log\frac{2p(x)}{p(x)+q(x)}\end{aligned}$$

Chu *et al.* (2019) noticed that:

$$\begin{aligned}\frac{q(x)}{p(x)+q(x)} &= 1-\frac{p(x)}{p(x)+q(x)} \\ &= 1-\frac{1}{2}e^{2\varphi(x)}\end{aligned}$$

In this case, we obtain:

$$\begin{aligned}h(x) &= \varphi(x)\,p(x)-\frac{1}{2}p(x)\log\frac{2p(x)}{p(x)+q(x)}-\frac{1}{2}q(x)\log\frac{2q(x)}{p(x)+q(x)} \\ &= \varphi(x)\,p(x)-\varphi(x)\,p(x)-\frac{1}{2}q(x)\log\left(2-e^{2\varphi(x)}\right)\end{aligned}$$

The convex conjugate of $\mathcal{J}_{\mathrm{JS}}$ is then:

$$\begin{aligned}
\mathcal{J}_{\mathrm{JS}}^{*}(\varphi) &= \int_{\mathcal{X}} -\frac{1}{2} q(x) \log\left(2 - e^{2\varphi(x)}\right) \mathrm{d}x \\
&= -\frac{1}{2} \int_{\mathcal{X}} q(x) \log\left(2 - e^{2\varphi(x)}\right) \mathrm{d}x \\
&= -\frac{1}{2} \int_{\mathcal{X}} \log\left(1 - \frac{1}{2} e^{2\varphi(x)}\right) q(x)\ \mathrm{d}x - \frac{1}{2}\log 2 \\
&= -\frac{1}{2} \mathbb{E}\left[\log\left(1 - e^{2\varphi(X) - \log 2}\right) \mid X \sim \mathbb{Q}\right] - \frac{1}{2}\log 2
\end{aligned}$$

Remark 9. *In order to retrieve Equation (40), we have to interchange $\mathbb{P}$ and $\mathbb{Q}$, because we need to compute $\mathcal{J}_{\mathbb{P}}(\mathbb{Q})$ and not $\mathcal{J}_{\mathbb{Q}}(\mathbb{P})$.*

A.5 Derivation of the minimax cost function

The cost function can be viewed as a binary cross-entropy measure. Let Y and $\hat{Y}$ be two random variables with probability mass function p and q. The cross-entropy function is equal to:

$$H(p,q) = \mathbb{E}\left[-\log q(x) \mid x \sim p(x)\right]$$

For discrete probability distributions, we obtain:

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

In a binary classification problem, for a given observation x_i, we have $p = y_i \in \{0,1\}$, which is the true label and $q = \hat{y}_i \in [0,1]$ which is the predicted probability of the current model. We can use binary cross entropy to get a measure of dissimilarity between y_i and $\hat{y}_i$:

$$H(p,q) = -y_i \log \hat{y}_i - (1-y_i)\log(1-\hat{y}_i)$$

In the case of m samples, the loss function is then given by:

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

Under the GAN framework, we have m samples of x_0 and m samples of x_1, which serve as the input data of the discriminator model. We note $x = \{x_0, x_1\}$ the set of the two samples. Since $\hat{y}_i = \boldsymbol{\mathcal{D}}(x_i;\theta_d)$, the formula above can be written as:

$$\mathcal{L}(\theta_g,\theta_d) = -\frac{1}{2m}\sum_{i=1}^{2m} y_i \log \boldsymbol{\mathcal{D}}(x_i;\theta_d) + (1-y_i)\log(1-\boldsymbol{\mathcal{D}}(x_i;\theta_d))$$

If x_i comes from the sample x_0, y_i takes the value 0, otherwise it takes the value 1. It follows that:

$$\begin{aligned}
\mathcal{L}(\theta_g,\theta_d) &= -\frac{1}{2m}\left(\sum_{i=1}^{m}\log \boldsymbol{\mathcal{D}}(x_{1,i};\theta_d) + \sum_{i=1}^{m}\log(1-\boldsymbol{\mathcal{D}}(x_{0,i};\theta_d))\right) \\
&= -\frac{1}{2}\left(\frac{1}{m}\sum_{i=1}^{m}\log \boldsymbol{\mathcal{D}}(x_{1,i};\theta_d) + \frac{1}{m}\sum_{i=1}^{m}\log(1-\boldsymbol{\mathcal{D}}(\boldsymbol{\mathcal{G}}(z_i;\theta_g);\theta_d))\right)
\end{aligned}$$

Therefore, the loss function $\mathcal{L}(\theta_g,\theta_d)$ corresponds to the average of the cross-entropy when considering several observations:

$$\mathcal{L}(\theta_g,\theta_d) = -\frac{1}{2}\mathbb{E}\left[\log \boldsymbol{\mathcal{D}}(x_1;\theta_d)\right] - \frac{1}{2}\mathbb{E}\left[\log(1-\boldsymbol{\mathcal{D}}(\boldsymbol{\mathcal{G}}(z;\theta_g);\theta_d))\right]$$

We notice that $\mathcal{C}(\theta_g,\theta_d)$ is equal to $-2\cdot\mathcal{L}(\theta_g,\theta_d)$. Minimizing the loss function is then equivalent to maximize $\mathcal{C}(\theta_g,\theta_d)$ with respect to θ_d.

A.6 An introduction to Monge-Kantorovich problems

A.6.1 Primal formulation of optimal transport

Optimal transport can be very powerful when comparing two probability distributions. The geometric approach that has been proposed will allow us to resolve complex optimization problems. OT problem relies on two probability spaces $(\mathcal{X}, \mathbb{P})$ and $(\mathcal{Y}, \mathbb{Q})$, and a cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$. For instance, we would like to transform a pile of sand, where particles are distributed according to $\mathbb{P}$ to a structured sand castle, where particles are distributed according to $\mathbb{Q}$. In this case, c denotes the amount of such effort. Thus, we can find an optimal path that minimizes the cost of this transformation. Let us introduce a map function $T : \mathcal{X} \to \mathcal{Y}$ that allow us to describe how $x \in \mathcal{X}$ is transported to the target space $\mathcal{Y}$. The optimal transport map function is the solution of the so-called Monge problem:

$$\inf_{T \in \mathcal{A}} \left\{ \int_{\mathcal{X}} c\left(x, T\left(x\right)\right) \,\mathrm{d}\mathbb{P}\left(x\right) \middle| T_*\left(\mathbb{P}\right) = \mathbb{Q} \right\} \tag{42}$$

where $\mathcal{A} = \left\{T : \mathcal{X} \to \mathcal{Y} \mid \mathbb{Q}\left(\mathcal{C}\right) = \mathbb{P}\left(T^{-1}\left(\mathcal{C}\right)\right), \mathcal{C} \subseteq \mathcal{Y}\right\}$. In other words, T must push-forward the probability measure $\mathbb{P}$ toward $\mathbb{Q}$, meaning that $\mathbb{Q} = T_*\left(\mathbb{P}\right)$.

However, Monge problem may be difficult to solve in some cases because the mapping function may not necessarily exist. For instance, let us consider that the source distribution $\mathbb{P}$ is a Dirac measure such as $\mathrm{d}\mathbb{P}\left(x\right) = \delta\left(x\right)\,\mathrm{d}x$ and the target distribution $\mathbb{Q}$ is a continuous measure such as a normal distribution. In this particular case, there is no map function T such that the condition $\mathbb{Q} = T_*\left(\mathbb{P}\right)$ is satisfied. Moreover, this condition is non-convex. To illustrate this, let us take $\mathbb{P}$ and $\mathbb{Q}$ two continuous Lebesgue measures of $\mathcal{X}$ such that $\mathrm{d}\mathbb{P}\left(x\right) = p\left(x\right)\,\mathrm{d}x$ and $\mathrm{d}\mathbb{Q}\left(x\right) = q\left(x\right)\,\mathrm{d}x$ for all $x \in \mathcal{X}$. It can be shown that satisfying the condition $\mathbb{Q} = T_*\left(\mathbb{P}\right)$ leads to the constraint $q\left(T\left(x\right)\right)\left|\det\left(\nabla_x T\left(x\right)\right)\right| = p\left(x\right)$ for all $x \in \mathcal{X}$. Since this constraint is non convex, the uniqueness of the minimization problem is not necessarily guaranteed. This is why Kantorovich reformulated the problem as follows:

$$\inf_{\mathbb{F} \in \mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X} \times \mathcal{Y}} c\left(x, y\right) \,\mathrm{d}\mathbb{F}\left(x, y\right) \right\} \tag{43}$$

where $\mathcal{F}\left(\mathbb{P}, \mathbb{Q}\right)$ is the Fréchet class[37]. The infimum is then obtained by considering all joint probability measures $\mathbb{F}$ on $\mathcal{X} \times \mathcal{Y}$ such that $\mathbb{P}$ and $\mathbb{Q}$ are the marginals. Such joint probability measures are called transportation plans. The Kantorovich transportation problem is now convex and becomes a linear programming problem easier to solve than the Monge transportation problem. However, searching among all join probability measures $\mathbb{F} \in \mathcal{F}\left(\mathbb{P}, \mathbb{Q}\right)$ can also be computationally intractable. For instance, Seguy *et al.* (2017) recall that solving the linear program takes $O\left(n^3 \log n\right)$ when n is the size of the support in the case of discrete probability distributions.

Remark 10. *Let us consider the particular case where $\mathbb{P}$ and $\mathbb{Q}$ are continuous Lebesgue measures and the cost function c correspond to a p-Euclidian distance $d\left(x, y\right)$. The solution to the Monge-Kantorovich problem is then defined as the p-Wasserstein distance[38]:*

$$W_p\left(\mathbb{P}, \mathbb{Q}\right) = \left(\inf_{\mathbb{F} \in \mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X} \times \mathcal{Y}} d\left(x, y\right)^p \,\mathrm{d}\mathbb{F}\left(x, y\right) \right\} \right)^{1/p}$$

Brenier (1991) showed that there is a unique solution when $p > 1$.

[37] This means that $\mathcal{F}\left(\mathbb{P}, \mathbb{Q}\right)$ collects all multivariate joint distributions, whose marginals are exactly equal to $\mathbb{P}$ and $\mathbb{Q}$.

[38] It is also known as the earth mover's distance (EMD), which has been used by Rubner *et al.* (2000) for content-based image retrieval in computer vision.

Remark 11. *In some particular cases, optimal transport problems can be easily solved. Let us consider the case where $\mathcal{X} = \mathcal{Y}$ is a one-dimensional space, and $c(x, y)$ is a convex function that satisfies the following condition: if $x_1 < x_2$ and $y_1 < y_2$, then $c(x_2, y_2) - c(x_1, y_2) - c(x_2, y_1) + c(x_1, y_1) < 0$, then the optimal transport plan respects the ordering of the elements. Consequently, the solution corresponds to a monotone rearrangement of $\mathbb{P}$ into $\mathbb{Q}$. Solving this problem is no more than sorting elements in a list (Brenier, 1991).*

A.6.2 Dual formulation of optimal transport

Another face of the optimal transport problem is the Kantorovich duality that can be easily understood from an economic point of view. Following closely the example given by Villani (2008), we consider a manufacturer that produces goods on a production site located at x and sells them on a store located at y. $c(x, y)$ is the cost to transport goods from x to y. Minimizing his transportation cost for the entire production is equivalent to solve the Monge-Kantorovich problem. Now, let us assume that he does not care about transportation. This is why he wants to hire a company specialized in goods transportation. They offer him to buy each product at the price $\varphi(x)$. They will then transport at y and sell back the product to him at the price $\psi(y)$. The manufacturer will accept the deal only if there is a financial interest such that $\psi(y) - \varphi(x) \leq c(x, y)$. Despite this constraint, the transportation company will try to maximize its profits. The new problem corresponds to the dual formulation of the Monge-Kantorovich problem and can be written as follows:

$$\sup \left\{ \int_{\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x)\ \mathrm{d}\mathbb{P}(y) : \psi(y) - \varphi(x) \leq c(x, y) \right\} \tag{44}$$

According to Villani (2008), φ and ψ are integrable such that $(\varphi, \psi) \in L^1(\mathcal{X}, \mathbb{P}) \times L^1(\mathcal{Y}, \mathbb{Q})$. The functions φ and ψ are often called Kantorovich potentials[39].

Remark 12. *Considering the viewpoint of the manufacturer who cares about the cost is equivalent to look at the solution of the Monge-Kantorovich primal problem. Considering the viewpoint of the transportation company that cares about optimizing the profit is equivalent to looking at the solution of the Monge-Kantorovich dual problem.*

A rigorous proof of dual formulation[40] is given by Villani (2008). We recall that the condition $\mathbb{F} \in \mathcal{F}(\mathbb{P}, \mathbb{Q})$ implies:

$$\begin{aligned}
\int_{\mathcal{X}\times\mathcal{Y}} (\psi(y) - \varphi(x))\ \mathrm{d}\mathbb{F}(x, y) &= \int_{\mathcal{X}\times\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{F}(x, y) - \int_{\mathcal{X}\times\mathcal{Y}} \varphi(x)\ \mathrm{d}\mathbb{F}(x, y) \\
&= \int_{\mathcal{Y}} \psi(y) \int_{\mathcal{X}} \mathrm{d}\mathbb{F}(x, y) - \int_{\mathcal{X}} \varphi(x) \int_{\mathcal{Y}} \mathrm{d}\mathbb{F}(x, y) \\
&= \int_{\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x)\ \mathrm{d}\mathbb{P}(x)
\end{aligned} \tag{45}$$

If $\mathbb{F} \notin \mathcal{F}(\mathbb{P}, \mathbb{Q})$, we assume by convention that the difference between the two members is

[39] Brenier (1991, Theorem 1.3) showed the link between Kantorovich potentials and mapping functions in the particular case where the cost function is a 2-Euclidean distance:

$$T(x) = x - \nabla_x \varphi(x) = \nabla_x \left(\frac{1}{2} \|x\|^2 - \varphi(x) \right)$$

[40] See also Xia (2008, 2009) for a geometric interpretation.

infinite and note $\mathbb{1}_{\mathcal{F}(\mathbb{P},\mathbb{Q})}(\mathbb{F})$ the convex indicator function of $\mathcal{F}(\mathbb{P},\mathbb{Q})$:

$$\begin{aligned}
(*) &= \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\ \mathrm{d}\mathbb{F}(x,y) \right\} \\
&= \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\ \mathrm{d}\mathbb{F}(x,y) \right\} + \mathbb{1}_{\mathcal{F}(\mathbb{P},\mathbb{Q})}(\mathbb{F}) \\
&= \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\ \mathrm{d}\mathbb{F}(x,y) \right\} + \\
&\quad \sup_{(\varphi,\psi)} \left\{ \int_{\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x)\ \mathrm{d}\mathbb{P}(x) - \int_{\mathcal{X}\times\mathcal{Y}} (\psi(y)-\varphi(x))\ \mathrm{d}\mathbb{F}(x,y) \right\} \\
&= \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \sup_{(\varphi,\psi)} \Gamma(\mathbb{P},\mathbb{Q},\mathbb{F},\varphi,\psi)
\end{aligned}$$

where:

$$\begin{aligned}
\Gamma(\mathbb{P},\mathbb{Q},\mathbb{F},\varphi,\psi) &= \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\ \mathrm{d}\mathbb{F}(x,y) + \int_{\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{Q}(y) - \\
&\quad \int_{\mathcal{X}} \varphi(x)\ \mathrm{d}\mathbb{P}(x) - \int_{\mathcal{X}\times\mathcal{Y}} (\psi(y)-\varphi(x))\ \mathrm{d}\mathbb{F}(x,y)
\end{aligned}$$

Since we have:

$$\inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \sup_{(\varphi,\psi)} \Gamma(\mathbb{P},\mathbb{Q},\mathbb{F},\varphi,\psi) = \sup_{(\varphi,\psi)} \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \Gamma(\mathbb{P},\mathbb{Q},\mathbb{F},\varphi,\psi)$$

it follows that:

$$\begin{aligned}
\inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \Gamma(\mathbb{P},\mathbb{Q},\mathbb{F},\varphi,\psi) &= \int_{\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x)\ \mathrm{d}\mathbb{P}(x) + \\
&\quad \inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} (c(x,y) - (\psi(y)-\varphi(x)))\ \mathrm{d}\mathbb{F}(x,y) \right\}
\end{aligned}$$

Finally, we conclude that[41]:

$$\inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\ \mathrm{d}\mathbb{F}(x,y) \right\} = \sup_{(\varphi,\psi)} \left\{ \int_{\mathcal{Y}} \psi(y)\ \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x)\ \mathrm{d}\mathbb{P}(x) \right\}$$

because:

$$\inf_{\mathbb{F}\in\mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} (c(x,y) - (\psi(y)-\varphi(x)))\ \mathrm{d}\mathbb{F}(x,y) \right\} = 0$$

A.6.3 Semi-dual formulation of optimal transport

It is possible to go deeper in the proof by introducing the notion of c-convexity (Villani, 2008). The function $\varphi : \mathcal{X} \to \mathbb{R}\cup\{+\infty\}$ is said to be c-convex if there exists a function $\zeta : \mathcal{Y} \to \mathbb{R}\cup\{\pm\infty\}$ such that:

$$\varphi(x) = \sup_{y\in\mathcal{Y}} \{\zeta(y) - c(x,y)\}$$

for all $x\in\mathcal{X}$. With this definition, it is possible to define its c-transform φ^c:

$$\varphi^c(y) = \inf_{x\in\mathcal{X}} \{\varphi(x) + c(x,y)\}$$

[41] Of course the constraint $\psi(y) - \varphi(x) \le c(x,y)$ must be satisfied.

for all $y \in \mathcal{Y}$. In the particular case where the cost function is a distance, a c-convex function is simply a 1-Lipschitz function, and is equal to its c-transform. Indeed, let us consider that φ is 1-Lipschitz such that $\varphi(x) - \varphi(y) \leq c(x,y)$. We have $\varphi(x) \leq \varphi(y) + c(x,y)$ and:

$$\begin{aligned} \varphi(x) &= \inf_y \{\varphi(y) + c(x,y)\} \\ &= \varphi^c(x) \end{aligned}$$

Again, it is possible to understand the c-transform through the economic point of view. Let us recall that the transportation company needs to satisfy the condition $\psi(y) - \varphi(x) \leq c(x,y)$ in order to remain competitive. It follows that $\psi(y) \leq \varphi(x) + c(x,y)$ and $\varphi(x) \geq \psi(y) - c(x,y)$. To maximize its profits, the company will choose the pair (φ, ψ) such that:

$$\left\{ \begin{array}{l} \psi(y) = \inf_x \{\varphi(x) + c(x,y)\} \\ \varphi(x) = \sup_y \{\psi(y) - c(x,y)\} \end{array} \right.$$

Therefore, it becomes useful to write ψ in term of φ. If we consider that the cost function is a distance and $\mathcal{X} = \mathcal{Y}$, Villani (2008) showed that:

$$\begin{aligned} (*) &= \inf_{\mathbb{F} \in \mathcal{F}(\mathbb{P},\mathbb{Q})} \left\{ \int_{\mathcal{X}\times\mathcal{Y}} c(x,y) \, \mathrm{d}\mathbb{F}(x,y) \right\} \\ &= \sup_{(\varphi,\psi)} \left\{ \int_{\mathcal{Y}} \psi(y) \, \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x) \, \mathrm{d}\mathbb{P}(x) : \psi(y) - \varphi(x) \leq c(x,y) \right\} \\ &= \sup_{\varphi} \left\{ \int_{\mathcal{Y}} \varphi^c(y) \, \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x) \, \mathrm{d}\mathbb{P}(x) \right\} \\ &= \sup_{\varphi} \left\{ \int_{\mathcal{Y}} \varphi(y) \, \mathrm{d}\mathbb{Q}(y) - \int_{\mathcal{X}} \varphi(x) \, \mathrm{d}\mathbb{P}(x) \right\} \\ &= \sup_{\varphi} \{\mathbb{E}[\varphi(Y) \mid Y \sim \mathbb{Q}] - \mathbb{E}[\varphi(X) \mid X \sim \mathbb{P}]\} \end{aligned}$$

This is the semi-dual formulation of the problem also called the Kantorovich-Rubinstein duality. This formulation is used to train the Wasserstein GAN that estimates the optimal function φ.

A.6.4 An example

If $\mathbb{P} \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $\mathbb{Q} \sim \mathcal{N}(\mu_2, \Sigma_2)$, Givens and Shortt (1984) showed that the 2-Wasserstein distance is equal to:

$$W_2(\mathbb{P}, \mathbb{Q}) = \sqrt{\|\mu_1 - \mu_2\|^2 + \operatorname{tr}\left(\Sigma_1 + \Sigma_2 - 2\left(\Sigma_1^{1/2} \Sigma_2 \Sigma_1^{1/2}\right)^{1/2}\right)} \tag{46}$$

where $A^{1/2}$ is the square root of A.

A.7 Converting real-valued samples into binary features

These transformation methods have been introduced by Kondratyev and Schwarz (2019). Algorithm (4) describes how to transform real-valued data into binary features. Each one-dimensional data sample is represented by a 16-digit binary number and in the case of n-dimensional data, we transform receptively each single value into 16-digit binary vector and concatenate them to form a $16 \times n$-digit binary vector.

Algorithm 4 Real-valued to integer to binary transformation

Result: Conversion of real-valued dataset into binary vector
Input: A real-valued dataset X_{real} with N samples
$\epsilon \geq 0$
$X_{\min} \leftarrow \min(X_{\text{real}}) - \epsilon$
$X_{\max} \leftarrow \max(X_{\text{real}}) + \epsilon$
for $l = 1, \cdots, N$ **do**
  $X_{\text{integer}}^{(l)} \leftarrow \text{int}\left(65535 \times \left(X_{\text{real}}^{(l)} - X_{\min}\right) / (X_{\max} - X_{\min})\right)$
  $X_{\text{binary}}^{(l)} \leftarrow \text{binarize}\left(X_{\text{integer}}^{(l)}\right)$
end for

Algorithm (5) performs the inverse transformation. Similarly, in the case of n-dimensional data, we transform receptively each 16 binary numbers into a real value and concatenate them to form a n-dimensional real-valued vector.

Algorithm 5 Binary to integer to real-valued transformation

Result: Conversion of binary vector into a real-valued sample
Input: A 16-digit binary vector $X = (X_1, \cdots, X_{16})$
$X_{\text{integer}} \leftarrow 0$
for $i = 1, \cdots, 16$ **do**

$$\hat{X}_{\text{integer}} \leftarrow \hat{X}_{\text{integer}} + 2^{i-1} \times \hat{X}_{16-i}$$

end for
$\hat{X}_{\text{real}} \leftarrow X_{\min} + \hat{X}_{\text{integer}} \times (X_{\max} - X_{\min}) / 65535$