Copulas as High-Dimensional Generative Models: Vine Copula Autoencoders

Natasa Tagasovska
Department of Information Systems
HEC Lausanne, Switzerland
natasa.tagasovska@unil.ch

Damien Ackerer
Swissquote Bank
Gland, Switzerland
damien.ackerer@swissquote.ch

Thibault Vatter
Department of Statistics
Columbia University, New York, USA
thibault.vatter@columbia.edu

Abstract

We introduce the vine copula autoencoder (VCAE), a flexible generative model for high-dimensional distributions built in a straightforward three-step procedure. First, an autoencoder (AE) compresses the data into a lower dimensional representation. Second, the multivariate distribution of the encoded data is estimated with vine copulas. Third, a generative model is obtained by combining the estimated distribution with the decoder part of the AE. As such, the proposed approach can transform any already trained AE into a flexible generative model at a low computational cost. This is an advantage over existing generative models such as adversarial networks and variational AEs which can be difficult to train and can impose strong assumptions on the latent space. Experiments on MNIST, Street View House Numbers and Large-Scale CelebFaces Attributes datasets show that VCAEs can achieve competitive results to standard baselines.

1 Introduction

Exploiting the statistical structure of high-dimensional distributions behind audio, images, or video data is at the core of machine learning. Generative models aim not only at creating feature representations, but also at providing means of sampling new realistic data points. Two classes are typically distinguished: *explicit* and *implicit* generative models. Explicit generative models make distributional assumptions on the data generative process. For example, *variational autoencoders* (VAEs) assume that the latent features are independent and normally distributed [37]. Implicit generative models make no statistical assumption but leverage another mechanism to transform noise into realistic data. For example, *generative adversarial networks* (GANs) use a discriminant model penalizing the loss function of a generative model producing unrealistic data [22]. Interestingly, *adversarial autoencoders* (AAEs) combined both features as they use a discriminant model penalizing the loss function of an encoder when the encoded data distribution differs from the prior (Gaussian) distribution [48]. All of these new types of generative models have achieved unprecedent results and also proved to be computationally more efficient than the first generation of deep generative models which require Markov chain Monte Carlo methods [32, 30]. However, adversarial approaches require multiple models to be trained, leading to difficulties and computational burden [62, 26, 24], and variational approaches make (strong) distributional assumptions, potentially detrimental to the generative model performance [64].

We present a novel approach to construct a generative model which is simple, makes no prior distributional assumption (over the input or latent space), and is computationally efficient: the *vine copula autoencoders* (VCAEs). Our approach, schematized in Figure 1 combines three tasks. First, an

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

autoencoder (AE) is trained to provide high-quality embeddings of the data. Second, the multivariate distribution of the encoded train data is estimated with vine copulas, namely, a flexible tool to construct high-dimensional multivariate distributions [3, 4, 1]. Third, a generative model is obtained by combining the estimated vine copula distribution with the decoder part of the AE.

In other words, new data is produced by decoding random samples generated from the vine copula. An already trained AE can thus be transformed into a generative model, where the only additional cost would be the estimation of the vine copula. We show in multiple experiments that this approach performs well in building generative models for the MNIST, Large-Scale CelebFaces Attributes, and Street View House Numbers datasets. To the best of our knowledge, this is the first time that vine copulas are used to construct generative models for very high dimensional data (such as images).

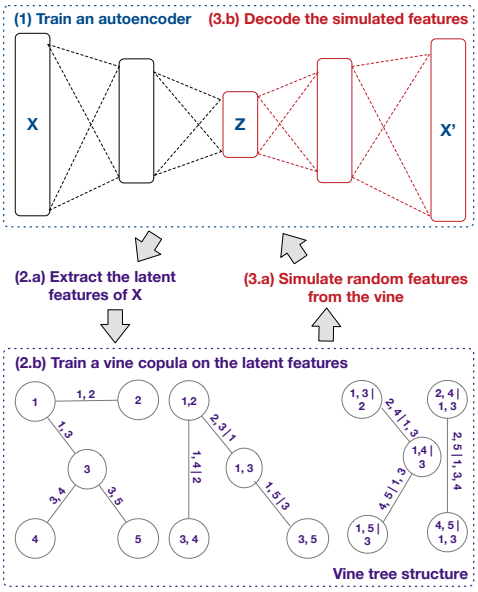


Figure 1: Conceptual illustration of a VCAE.

Next, we review the related work most relevant to our setting. The most widespread generative models nowadays focus on synthetic image generation, and mainly fall into the GAN or VAE categories, some interesting recent developments include [49, 15, 26, 76, 29, 14, 6]. These modern approaches have been largely inspired by previous generative models such as belief networks [32], independent component analysis [33] or denoising AEs [79]. Part of their success can be attributed to the powerful neural network architectures which provide high quality feature representations, often using Convolutional architectures [41]. A completely different framework to model multivariate distributions has been developed in the statistical literature: the so-called *copulas*. Thanks to their ability to capture complex dependence structures, copulas have been applied to a wide range of scientific problems, and their successes have led to continual advances in both theory and open-source software availability. We refer to [56, 35] for textbook introductions. More recently, copulas also made their way into machine learning research [43, 20, 47, 78, 45, 13, 74, 38]. However, copulas have not yet been employed in constructing high dimensional generative models. While [42, 59] use copulas for synthetic data generation, they rely on strong parametric assumptions. In this work, we illustrate how *nonparametric vine copulas* allow for arbitrary density estimation [50], which in turn can be used to sample realistic synthetic datasets.

Because their training is relatively straightforward, VCAEs have some advantages over GANs. For instance, GANs require some complex modifications of the baseline algorithm in order to avoid mode collapse, whereas vines naturally fit multimodal data. Additionally, while GANs suffer from the "exploding gradients" phenomenon (e.g., see [24]) and require careful monitoring of the training and early stopping, this is not an issue with VCAEs as they are built upon standard AEs.

To summarize, the contribution of this work is introducing a novel, competitive generative model based on copulas and AEs. There are three main advantages of the proposed approach. First, it offers modeling flexibility by avoiding most distributional assumptions. Second, training and sampling procedures for high-dimensional data are straightforward. Third, it can be used as a plug-in allowing to turn any AE into generative model, simultaneously allowing it to serve other purposes (e.g., denoising, clustering).

The remainder of the paper is as follows. Section 2 reviews vine copulas as well as their estimation and simulation algorithms. Section 3 discusses the VCAE approach. Section 4 presents the results of our experiments. Section 5 concludes and discusses future research. The supplementary material contains further information on algorithm and experiments, as well as additional experiments.

2 Vine copulas

2.1 Preliminaries and motivation

A *copula*, from the latin word *link*, flexibly "couples" marginal distributions into a joint distribution. As such, copulas allow to construct joint distributions with the same margins but different dependence structures, or conversely by fixing the dependence structure and changing the individual behaviors. Thanks to this versatility, there has been an exponentially increasing interest in copula-based models over the last two decades. One important reason lies in the following theorem.

Theorem 1 (Sklar's theorem [71]). *The continuous random vector $\boldsymbol{X} = (X_1, \dots, X_d)$ has joint distribution F and marginal distributions $F_1, \dots, F_d$ if and only if there exist a unique copula [1] C, which is the joint distribution of $\boldsymbol{U} = (U_1, \dots, U_d) = \big(F_1(X_1), \dots, F_d(X_d)\big)$.*

Assuming that all densities exist, we can write $f(x_1, \dots, x_d) = c\big\{u_1, \dots, u_d\big\} \times \prod_{k=1}^{d} f_k(x_k)$, where $u_i = F_i(x_i)$ and $f, c, f_1, \dots, f_d$ are the densities corresponding to $F, C, F_1, \dots, F_d$ respectively. As such, copulas allow to decompose a joint density into a product between the marginal densities f_i and the dependence structure represented by the copula density c.

This has an important implication for the estimation and sampling of copula-based marginal distributions: algorithms can generally be built into two steps. For instance, estimation is often done by estimating the marginal distributions first, and then using the estimated distributions to construct pseudo-observations via the probability integral transform before estimating the copula density. Similarly, synthetic samples can be obtained by sampling from the copula density first, and then using the inverse probability integral transform to transform the copula sample back to the natural scale of the data. We give a detailed visual example of both the estimation and sampling of (bivariate) copula-based distributions in Figure 2. We also refer to Appendix A.1 or the textbooks [56] and [35] for more detailed introductions on copulas.

The availability of higher-dimensional models is rather limited, yet there exists numerous parametric families in the bivariate case. This has inspired the development of hierarchical models, constructed from cascades of bivariate building blocks: the *pair-copula constructions* (PCCs), also called *vine copulas*. Thanks to its flexibility and computational efficiency, this new class of simple yet versatile models has quickly become a hot-topic of multivariate analysis [2].

2.2 Vine copulas construction

Popularized in [3, 4, 1], PCCs model the joint distribution of a random vector by decomposing the problem into modeling pairs of conditional random variables, making the construction of complex dependencies both flexible and yet tractable. Let us exemplify such constructions using a three dimensional vector of continuously distributed random variables $X = (X_1, X_2, X_3)$. The joint density f of X can be decomposed as

$$f = f_1\, f_2\, f_3\, c_{1,2}\, c_{2,3}\, c_{1,3|2}, \tag{1}$$

where we omitted the arguments for the sake of clarity, and f_1, f_2, f_3 are the marginal densities of X_1, X_2, X_3, $c_{1,2}$ and $c_{2,3}$ are the joint densities of $(F_1(X_1), F_2(X_2))$ and $(F_2(X_2), F_3(X_3))$,

$c_{1,3|2}$ is the joint density of $(F_{1|2}(X_1|X_2), F_{3|2}(X_3|X_2))|X_2$.

The above decomposition can be generalized to an arbitrary dimension d and leads to tractable and flexible probabilistic models [34, 3, 4]. While a decomposition is not unique, it can be organized as a graphical model, a sequence of $d-1$ nested trees, called *regular vine*, *R-vine*, or simply *vine*. Denoting $T_m = (V_m, E_m)$ with V_m and E_m the set of nodes and edges of tree m for $m = 1, \dots, d-1$, the sequence is a vine if it satisfies a set of conditions guaranteeing that the decomposition leads to a *valid joint density*. The corresponding tree sequence is then called the *structure* of the PCC and has important implications to design efficient algorithms for the estimation and sampling of such models (see Section 2.3 and Section 2.4).

Each edge e is associated to a bivariate copula $c_{j_e,k_e|D_e}$ (a so-called *pair-copula*), with the set $D_e \in \{1, \cdots, d\}$ and the indices $j_e, k_e \in \{1, \cdots, d\}$ forming respectively its *conditioning set* and the *conditioned set*. Finally, the joint copula density can be written as the product of all pair-copula densities $c = \prod_{m=1}^{d-1} \prod_{e \in E_m} c_{j_e,k_e|D_e}$. In the following two sections, we discuss two topics that are

[1] A copula is a distribution function with uniform margins.

important for the application of vines as generative models: estimation and simulation. For further details, we refer to the numerous books and surveys written about them [16, 39, 72, 18, 2], as well as Appendix A.2.

2.3 Sequential estimation

To estimate vine copulas, it is common to follow a sequential approach [1, 27, 50], which we outline below. Assuming that the vine structure is known, the pair-copulas of the first tree, T_1, can be directly estimated from the data. But this is not as straightforward for the other trees, since data from the densities $c_{j_e,d_e|D_e}$ are not observed. However, it is possible to sequentially construct "pseudo-observations" using appropriate data transformations, leading to the following estimation procedure, starting with tree T_1: for each edge in the tree, estimate all pairs, construct pseudo-observations for the next tree, and iterate. The fact that the tree sequence $T_1, T_2, \dots, T_{d-1}$ is a regular vine guarantees that at any step in this procedure, all required pseudo-observations are available. Additionally to Appendix A.2.1 and Appendix A.2.2, we further refer to [1, 12, 18, 19, 9, 36] for model selection methods and to [17, 73, 11, 27, 69] for more details on the inference and computational challenges related to PCCs.

Importantly, vines can be truncated after a given number of trees [12, 8, 10] by setting pair-copulas in further trees to independence.

Complexity Because there are d pair-copulas in T_1, $d-1$ pair-copulas in T_2, ..., and a single pair-copula in T_{d-1}, the complexity of this algorithm is $O(f(n) \times d \times \text{truncation level})$, where $f(n)$ is the complexity of estimating a single pair and the truncation level is at most $d-1$. In our implementation, described Section 2.5, $f(n) = O(n)$.

2.4 Simulation

Additionally to their flexibility, vines are easy to sample from using inverse transform sampling. Let C be a copula and $U = (U_1, \dots, U_d)$ is a vector of independent $U(0,1)$ random variables. Then, define $V = (V_1, \dots, V_d)$ through $V_1 = C^{-1}(U_1)$, $V_2 = C^{-1}(U_2|U_1)$, and so on until $V_d = C^{-1}(U_d|U_1, \dots, U_{d-1})$, with $C(v_k|v_1, \dots, v_{k-1})$ is the conditional distribution of V_k given $V_1, \dots, V_{k-1}$, $k = 2, \dots, d$. In other words, V is the inverse Rosenblatt transform [65] of U. It is then straightforward to notice that $V \sim C$, which can be used to simulate from C. As for the sequential estimation procedure, it turns out that

- the fact that the tree sequence $T_1, T_2, \dots, T_{d-1}$ is a vine guarantees that all the required conditional bivariate copulas are available (see Algorithm 2.2 of [19]),
- the complexity of the algorithm $O(n \times d \times \text{truncation level})$, since $f(n)$ is trivially the complexity required for one inversion multiplied by the number of generated samples.

Furthermore, there exist analytical expressions or good numerical approximations of such inverses for common parametric copula families. We refer to Section 2.5 for a discussion of the inverse computations for nonparametric estimators.

2.5 Implementation

To avoid specifying the marginal distributions, we estimate them using a Gaussian kernel with a bandwidth chosen using the direct plug-in methodology of [70]. The observations can then be mapped to the unit square using the probability integral transform (PIT). See steps 1 and 2 of Figure 2 for an example.

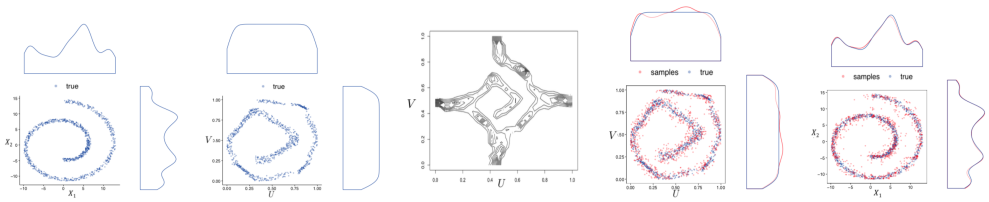
Regarding the copula families used as building blocks for the vine, one can contrast parametric and nonparametric approaches. As is common in machine learning and statistics, the default choice is the Gaussian copula. In Section 2.6, we show empirically why this assumption (allowing for dependence between the variables but still in the Gaussian setting) can be too simplistic, resulting in failure to deliver even for three dimensional datasets.

Alternatively, using a nonparametric bivariate copula estimator provides the required flexibility. However, the bivariate Gaussian kernel estimator, targeted at densities of unbounded support, cannot be directly applied to pair-copulas, which are supported in the unit square. To get around this issue, the trick is to transform the data to standard normal margins before using a bivariate Gaussian

kernel. Bivariate copulas are thus estimated nonparametrically using the transformation estimator [67, 47, 50, 21] defined as

$$\widehat{c}(u,v) = \frac{1}{n}\sum_{j=1}^{n} \frac{\mathcal{N}(\Phi^{-1}(u), \Phi^{-1}(v) | \Phi^{-1}(u_j), \Phi^{-1}(v_j), \Sigma)}{\phi\left(\Phi^{-1}(u)\right)\phi\left(\Phi^{-1}(v)\right)}, \tag{2}$$

where $\mathcal{N}(\cdot,\cdot|\upsilon_1,\upsilon_2,\Sigma)$ is a two-dimensional Gaussian density with mean υ_1,υ_2, and covariance matrix $\Sigma = n^{-1/3}\,\mathrm{Cor}(\Phi^{-1}(U),\Phi^{-1}(V))$. For the notation we let ϕ, Φ and Φ^{-1} to be the standard Gaussian density, distribution and quantile function respectively. See step 3 of Figure 2 for an example.



(Step 1)
Use a Gaussian kernel density estimator for the marginals:

$$\widehat{f}_i(x) = \frac{1}{n}\sum_{j=1}^{n}\mathcal{N}_{X_{i,j},\sigma^2}(x),$$

for $i \in \{1,2\}$ and obtain the CDF $\widehat{F}_i$ by replacing $\mathcal{N}$ with its cumulative counterpart.

(Step 2)
Use the probability integral transform (PIT) to get pseudo-data supported in $[0,1]$:

$$U_j = \widehat{F}_1(X_{1,j}),$$
$$V_j = \widehat{F}_2(X_{2,j})$$

for $j \in \{1,\dots,n\}$.

(Step 3)
Use the pseudo-data to estimate the copula density $\widehat{c}(u,v)$ nonparametrically with (2). An estimator of the conditional distribution of V given U is obtained by

$$\widehat{C}(v \mid u) = \int_0^v \widehat{c}(u,z)dz.$$

(Step 4)
Sample $\tilde{U}_j, \tilde{V}_j \sim U(0,1)$ independently and use a multivariate version of inversion sampling to get a sample of the copula:

$$U_j^s = \tilde{U}_j$$
$$V_j^s = \widehat{C}^{-1}(\tilde{V}_j \mid U_j^s),$$

for $j \in \{1,\dots,n\}$.

(Step 5)
Use the inverse PIT to transform back the simulated pseudo-sample:

$$X_{1,j}^s = \widehat{F}_1^{-1}(U_j^s),$$
$$X_{2,j}^s = \widehat{F}_2^{-1}(V_j^s)$$

for $j \in \{1,\dots,n\}$.

Figure 2: Estimation and sampling algorithm for a pair copula.

Along with vines-related functions (i.e., for sequential estimation and simulation), the Gaussian copula and (2) are implemented in C++ as part of `vinecopulib` [51], a header-only C++ library for copula models based on `Eigen` [25] and `Boost` [68]. In the following experiments, we use the R interface [61] interface to `vinecopulib` called `rvinecopulib` [53], which also include `kde1d` [52] for univariate density estimation.

Note that inverses of partial derivatives of the copula distribution corresponding to (2) are required to sample from a vine, as described in Section 2.4. Internally, `vinecopulib` constructs and stores a grid over $[0,1]^2$ along with the evaluated density at the grid points. Then, bilinear interpolation is used to efficiently compute the copula distribution $\widehat{C}(u,v)$ and its partial derivatives. Finally, `vinecopulib` computes the inverses by numerically inverting the bilinearly interpolated quantities using a vectorized version of the bisection method, and we show a copula sample example as step 4 of Figure 2. The consistency and asymptotic normality of this estimator are derived in [21] under assumptions described in Appendix A.3.

To recover samples on the original scale, the simulated copulas samples, often called pseudo-samples, are then transformed using the inverse PIT, see step 5 of Figure 2. In Appendix C.1, we show that this estimator performs well on two toy bivariate datasets that are typically challenging for GANs: a grid of isotropic Gaussians and the swiss roll.

2.6 Vines as generative models

To exemplify the use of vines as generative models, let us consider as a running example a three dimensional dataset X_1, X_2, X_3 with $X_1, X_2 \sim U[-5,5]$ and $X_3 = \sqrt{X_1^2 + X_2^2} + U[-0.1, 0.1]$. The joint density can be decomposed as in the right-hand side of (1), and estimated following the procedures described in Section 2.5 and Section 2.3. With the structure and the estimated pair copulas, we can then use vines as generative models.

In Figure 3, we showcase three models. C1 is a nonparametric vine truncated after the first tree. In other words, it sets $c_{2,3|1}$ to independence. C2 is a nonparametric vine with two trees. C3 is a Gaussian vine with two trees. On the left panel, we show their vine structure, namely the trees and the pair copulas. On the right panel, we present synthetic samples from each of the models in blue, with the green data points corresponding to $\sqrt{X_1^2 + X_2^2}$.

Comparing C1 to C2 allows to understand the truncation effect: C2, being more flexible (fitting richer/deeper model), captures better the features of the joint distribution. It can be deduced from the fact that data generated by C2 looks like uniformly spread around the $\sqrt{X_1^2 + X_2^2}$ surface, while data generated by C1 is spread all around. It should be noted that, in both cases, the nonparametric estimator captures the fact that X_1 and X_2 are independent, as can be seen from the contour densities on the left panel. Regarding C3, it seems clear that Gaussian copulas are not suited to handle this kind of dependencies: for such nonlinearities, the estimated correlations are (close to) zero, as can be seen from the contour densities on the left panel.

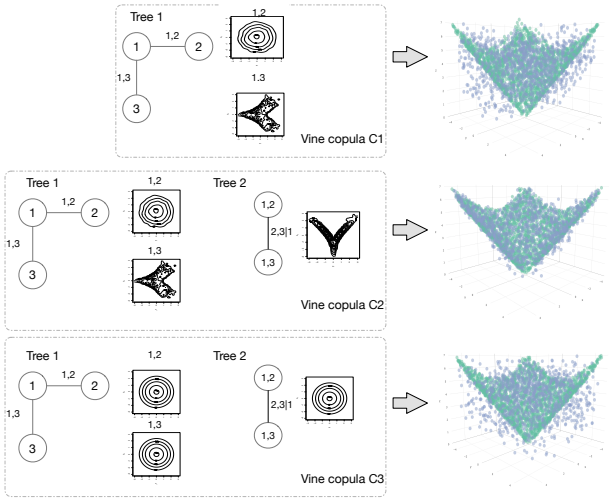


Figure 3: Simulation with different truncation levels, top to bottom - 1 level truncated vine, 2 levels non-parametric vine, 2 levels Gaussian vine.

With this motivation, the next section is dedicated to extending the vine generative approach to high dimensional data. While vines are theoretically suitable for fitting and sampling in high dimensions, they have been only applied to model a few thousands of variables. The reason is mainly that state-of-the-art implementations were geared towards applications such as climate science and financial risk computations. While software such a `vinecopulib` satisfies the requirements of such problems, even low-resolution images (e.g., $64 \times 64 \times 3$) are beyond its current capabilities. To address this challenge, we can rely on the embedded representations provided by neural networks.

3 Vine copula autoencoders

The other building block of the VCAE is an *autoencoder* (AE) [7, 31]. These neural network models typically consist of two parts: an *encoder* f mapping a datum X from the original space $\mathcal{X}$ to the latent space $\mathcal{Y}$, and a decoder g mapping a latent code Y from the latent space $\mathcal{Y}$ to the original space $\mathcal{X}$. The AE is trained to reconstruct the original input with minimal reconstruction loss, that is $X' \approx g(f(X))$.

However, AEs simply learn the most informative features to minimize the reconstruction loss, and therefore cannot be considered as generative models. In other words, since they do not learn the distributional properties of the latent features [5], they cannot be used to sample new data points. Because of the latent manifold's complex geometry, attempts using simple distributions (e.g., Gaussian) for the latent space may not provide satisfactory results.

Nonparametric vines naturally fill this gap. After training an AE, we use its encoder component to extract lower dimensional feature representations of the data. Then, we fit a vine without additional restrictions on the latent distribution. With this simple step, we transform AEs into generators, by systematically sampling data from the vine copula, following the procedure from Section 2.4. Finally, we use the decoder to transform the samples from vine in latent space into simulated images in pixel space. A schematic representation of this idea is given in Figure 1 and pseudo-code for the VCAE algorithm can be found in Appendix B.

The vine copula is fitted post-hoc for two reasons. First, since the nonparametric estimator is consistent for (almost) any distribution, the only purpose of the AE is to minimize the reconstruction error. The AE's latent space is unconstrained and the same AE can be used for both conditional and unconditional sampling. Second, it is unclear how to train a model that includes a nonparametric estimator since it has no parameters, there is no loss function to minimize or gradients to propagate. One possibility would be using spline estimators, which would allow to train the model end-to-end by fitting the basis expansion's coefficients. However, spline estimators of copula densities have been empirically shown to have inferior performance than the transformation kernel estimator [55].

There is some leeway in modeling choices related to the vine. For instance, the number of trees as well as the choice of copula family (i.e., Gaussian or nonparametric) have an impact of the synthetic samples, as sharper details are expected from more flexible models. Note that one can adjust the characteristics of the vine until an acceptable fit of the latent features even after the AE is trained.

4 Experiments

To evaluate VCAEs as generative models, we follow an experimental setup similar as related works on GANs and VAEs. We compare vanilla VAEs to VCAEs using the same architectures, but replacing the variational part of the VAEs by vines to obtain the VCAEs. From the generative adversarial framework, we compare to DCGAN [62]. The architectures for all networks are described in Appendix D.

Additionally, we explore two modifications of VCAE, (i) Conditional VCAE, that is sampling from a mixture obtained by fitting one vine per class label, and (ii) DEC-VCAE, namely adding a clustering-related penalty as in [81]. The rationale behind the clustering penalty was to better disentangle the features in the latent space. In other words, we obtain latent representations where the different clusters (i.e., classes) are better separated, thereby facilitating their modeling.

4.1 Experimental setup

Datasets and metrics

We explore three real-world datasets: two small scale - MNIST [40] and Street View House Numbers (SVNH) [57], and one large scale - CelebA [44]. While it is generally common to evaluate models by comparing their log-likelihood on a test dataset, this criterion is known to be unsuitable to evaluate the quality of sampled images [75]. As a result, we use an evaluation framework recently developed for GANs [82]. According to [82], the most robust metrics for two sample testing are the *classifier two sample test* (C2ST, [46]) and *mean maximum discrepancy* score (MMD, [23]). Furthermore, [82] proposes to use these metrics not only in the pixel space, but over feature mappings in convolution space. Hence, we also compare generative models in terms of Wasserstein distance, MMD score and C2ST accuracy over ResNet-34 features. Additionally, we also use the *common inception score* [66] and *Fréchet inception distance* (FID, [28]). For all metrics, lower values are better, except for inception. We refer the reader to [82] for further details on the metrics and the implementation.

Architectures, hyperparameters, and hardware

For all models, we fix the AE's architecture as described in Appendix D. Parameters of the optimizers and other hyperparameters are fixed as follows. Unless stated otherwise, all experiments were run with nonparametric vines and truncated after 5 trees. We use deep CNN models for the AEs in all baselines and follow closely DCGAN [62] with batch normalization layers for natural image datasets. For all AE-based methods, we use the Adam optimizer with learning rate 0.005 and weight decay 0.001 for all the natural image experiments, and 0.001 for both parameters on MNIST. For DCGAN, we use the recommended learning rate 0.0002 and $\beta_1 = 0.5$ for Adam. The size of the latent spaces z was selected depending on the dataset's size and complexity. For MNIST, we present results with $z = 10$, SVHN $z = 20$ and for CelebA $z = 100$. We chose to present the values that gave reasonable results for all baselines. For MNIST, we used batch size of 128, for SVHN 32, and for CelebA batches of 100 samples for training. All models were trained on a separate train set, and evaluated on hold out test sets of 2000 samples, which is the evaluation size used in [82]. We used Pythorch 4.1 [58], and we provide our code in Appendix E. All experiments were executed on an AWS instance *p2.xlarge* with an NVIDIA K80 GPU, 4 CPUs and 61 GB of RAM.

4.2 Results

MNIST

In Figure 4, we present results from VCAE to understand how different copula families impact the quality of the samples. The independence copula corresponds to assuming independence between the latent features as in VAEs. And the images generated using nonparametric vines seem to improve over the other two. Within our framework, the training of the AE and the vine fit are independent. And we can leverage this to perform conditional sampling by fitting a different vine for each class of digit. We show results of vine samples per digit class in Figure 4.

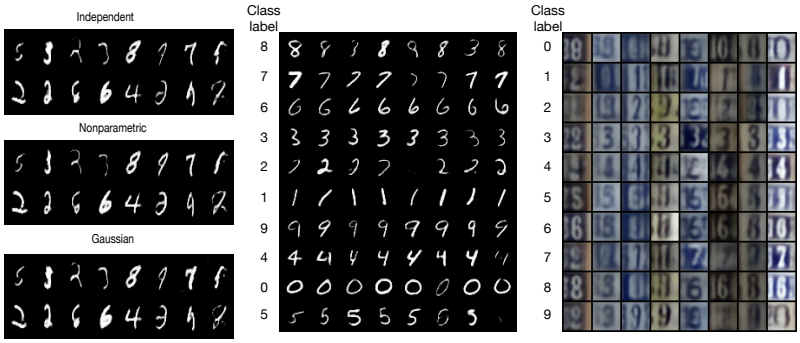


Figure 4: Left - impact of copula family selection on **MNIST**. Middle and Right - random samples of Conditional *VCAE* on **MNIST** and **SVHN**.

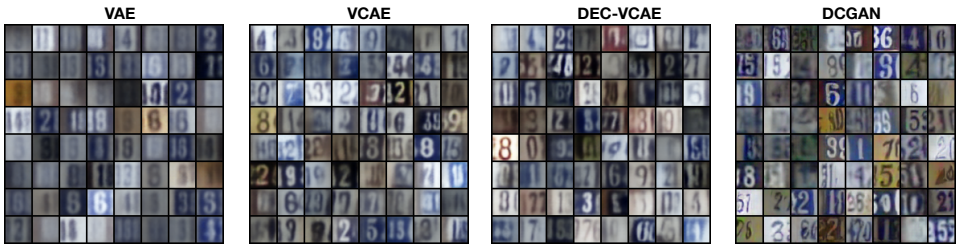


Figure 5: Left to right, random samples of *VAE*, *VCAE*, *DEC-VCAE*, and *DCGAN* for **SVHN**.

SVHN

The results in Figure 5 show that the variants of vine generative models visually provide sharper images than vanilla VAEs when architectures and training hyper-parameters are the same for all models. All AE-based methods were trained on latent space $z = 20$ for 200 epochs, while for DCGAN we use $z = 100$ and evaluate it at its best performance (50 epochs). In Figure 6, we can see that VCAE and DEC-VCAE have very similar and competitive results to DCGAN (at its best) across all metrics, and both clearly outperform vanila VAE. Finally, the FID score calculated with regards to 10^4 real test samples are has 0.205 for VAE, 0.194 for DCGAN and 0.167 for VCAE which shows that VCAE also has slight advantage using this metric. In Appendix C.2, Figure 12 and Figure 13 show similar results respectively for the MNIST and CelebA datasets.

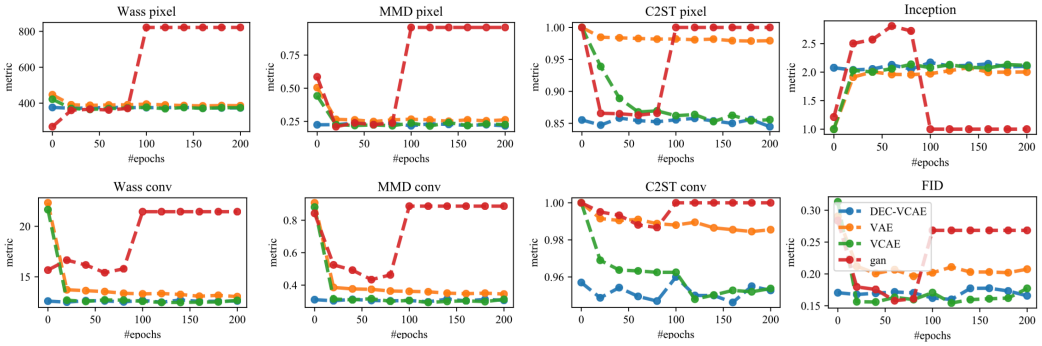


Figure 6: Various evaluation scores for all baselines on the **SVHN** dataset.

CelebA

In the large scale setting, we present results for VCAE, VAE, and DCGAN only, because our GPU ran out of memory on DEC-VCAE. From the random samples in Figure 7, we see that, for the same amount of training (in terms of epochs), VCAE results is not only sharper but also produce more diverse samples. VAEs improve using additional training, but vine-based solutions achieve better results with less resources and without constraints on the latent space. Note that, in Appendix C.3, we also study the quality of the latent representation.

To see the effect of the number of trees in the vine structure, we include Figure 8, where we can see that from the random sample the vine with five trees provides images with sharper details.

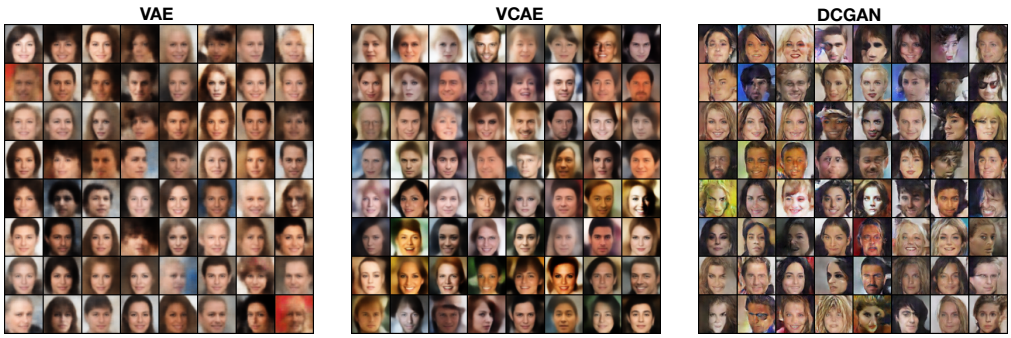


Figure 7: Random samples for models trained on the **CelebA** dataset, for *VAE* and *VCAE* at 200 epochs, and for *DCGAN* best results at 30 epochs.

Since, as stated in Section 2.3 and Section 2.4, the algorithms complexity increases linearly with the number of trees, we explore the trade-off between computation time and quality of the samples in Appendix C.4. Results show that, as expected, deeper vines, and hence longer computation times, improve the quality of the generated images. Finally, as for SVHN, the FID score shows an advantage of the vine-base method over VAEs as we find 0.247 for VAE and 0.233 for VCAE. For DCGAN the FID score is 0.169 which is better than VCAE, however, looking at the random batch samples in Figure 7 although GANs outputs sharper images, it is clear that VCAE produces more realistic faces.

Figure 8: Higher truncation - sharper images.

Execution times

We conclude the experimental section with Table 1 comparing execution times. We note that VCAE compares favorably to VAE, which is a "fair" observation given that the architectures are alike. Comparison to DCGAN is more difficult, due to the different nature of the two frameworks (i.e., based respectively on AEs or adversarial).

Table 1: Execution times.

	MNIST (200 epochs)	**SVHN (200 epochs)**	**CelebA (100 epochs)**
VAE	50 min	4h 7 min	7h
VCAE	55 min	1h 32 min	6.5h
DEC VCAE	101 min	2h 35 min	/
DCGAN	120 min (40 epochs)	3h 20 min (50 epochs)	5h (30 epochs)

It should also be noted that the implementation of VCAE is far from optimal for two reasons. First, we use the `R` interface to `vinecopulib` in `Python` through `rpy2`. As such, there is a communication overhead resulting from switching between `R` and `Python`. Second, while `vinecopulib` uses native `C++11` multithreading, it does not run on GPU cores. From our results, this is not problematic, since the execution times are satisfactory. But VCAE could be much faster if nonparametric vines were implemented in a tensor-based framework.

5 Conclusion

In this paper, we present vine copula autoencoders (VCAEs), a first attempt at using copulas as high-dimensional generative models. VCAE leverage the capacities of AEs at providing compressed representations of the data, along with the flexibility of nonparametric vines to model arbitrary probability distributions. We highlight the versatility and power of vines as generative models in high-dimensional settings with experiments on various real datasets. VCAEs results show that they are comparable to existing solutions in terms of sample quality, while at the same time providing straightforward training along more control over flexibility at modeling and exploration (tuning truncation level, selection of copula families/parameter values). Several directions for future work and extensions are being considered. First, we started to experiments with VAEs having flexible distributional assumptions (i.e., by using a vine on the variational distribution). Second, we plan on studying hybrid models using adversarial mechanisms. In related work [38] (see Appendix F), we have also investigated the method's potential for sampling *sequential data* (artificial mobility trajectories). There can also be extensions to text data, or investigating which types of vines synthesize best samples for different data types.

References

[1] K. Aas, C. Czado, A. Frigessi, and H. Bakken. Pair-Copula Constructions of Multiple Dependence. *Insurance: Mathematics and Economics*, 44(2):182–198, 2009.

[2] Kjersti Aas. Pair-copula constructions for financial applications: A review. *Econometrics*, 4(4): 43, October 2016.

[3] Tim Bedford and Roger M. Cooke. Probability Density Decomposition for Conditionally Dependent Random Variables Modeled by Vines. *Annals of Mathematics and Artificial Intelligence*, 32(1-4):245–268, 2001.

[4] Tim Bedford and Roger M. Cooke. Vines – A New Graphical Model for Dependent Random Variables. *The Annals of Statistics*, 30(4):1031–1068, 2002.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

[6] Diane Bouchacourt, Ryota Tomioka, and Sebastian Nowozin. Multi-level variational autoencoder: Learning disentangled representations from grouped observations. In *AAAI*, 2018.

[7] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

[8] Eike C. Brechmann and Harry Joe. Parsimonious parameterization of correlation matrices using truncated vines and factor analysis. *Computational Statistics and Data Analysis*, 77:233–251, 2014.

[9] Eike Christian Brechmann and Claudia Czado. COPAR—multivariate time series modeling using the copula autoregressive model. *Applied Stochastic Models in Business and Industry*, 31 (4):495–514, 2015.

[10] Eike Christian Brechmann and Harry Joe. Truncation of vine copulas using fit indices. *Journal of Multivariate Analysis*, 138:19–33, 2015.

[11] Eike Christian Brechmann and Ulf Schepsmeier. Modeling dependence with C-and D-vine copulas: The R-package CDVine. *Journal of Statistical Software*, 52(3):1–27, 2013.

[12] Eike Christian Brechmann, Claudia Czado, and Kjersti Aas. Truncated regular vines in high dimensions with application to financial data. *Canadian Journal of Statistics*, 40(1):68–85, March 2012.

[13] Yale Chang, Yi Li, Adam Ding, and Jennifer Dy. A robust-equitable copula dependence measure for feature selection. *AISTATS*, 2016.

[14] Tatjana Chavdarova and François Fleuret. Sgan: An alternative training of generative adversarial networks. In *CVPR*, 2018.

[15] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[16] Claudia Czado. Pair-Copula Constructions of Multivariate Copulas. In Piotr Jaworski, Fabrizio Durante, Wolfgang Karl Härdle, and Tomasz Rychlik, editors, *Copula Theory and Its Applications*, Lecture Notes in Statistics, pages 93–109. Springer Berlin Heidelberg, 2010.

[17] Claudia Czado, Ulf Schepsmeier, and Aleksey Min. Maximum likelihood estimation of mixed C-vines with application to exchange rates. *Statistical Modelling*, 12(3):229–255, 2012.

[18] Claudia Czado, Eike Christian Brechmann, and Lutz Gruber. Selection of Vine Copulas. In Piotr Jaworski, Fabrizio Durante, and Wolfgang Karl Härdle, editors, *Copulae in Mathematical and Quantitative Finance: Proceedings of the Workshop Held in Cracow, 10-11 July 2012*, volume 36. Springer New-York, 2013.

[19] J. Dissmann, Eike Christian Brechmann, Claudia Czado, Dorota Kurowicka, J Dißmann, Eike Christian Brechmann, Claudia Czado, and Dorota Kurowicka. Selecting and estimating regular vine copulae and application to financial returns. *Computational Statistics & Data Analysis*, 59:52–69, March 2013.

[20] Gal Elidan. Copulas in machine learning. In *Copulae in mathematical and quantitative finance*, pages 39–60. Springer, 2013.

[21] Gery Geenens, Arthur Charpentier, and Davy Paindaveine. Probit transformation for nonparametric kernel estimation of the copula density. *Bernoulli*, 23(3):1848–1873, 2017.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

[23] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *NeurIPS*, 2007.

[24] Paulina Grnarova, Kfir Y Levy, Aurelien Lucchi, Nathanael Perraudin, Thomas Hofmann, and Andreas Krause. Evaluating gans via duality. *arXiv preprint arXiv:1811.05512*, 2018.

[25] Gaël Guennebaud, Benoît Jacob, and Others. Eigen v3, 2010.

[26] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017.

[27] Ingrid Hobæk Haff. Parameter estimation for pair-copula constructions. *Bernoulli*, 19(2): 462–491, 2013.

[28] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

[29] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2017.

[30] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[31] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *NeurIPS*, pages 3–10, 1994.

[32] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[33] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[34] Harry Joe. *Multivariate Models and Dependence Concepts*. Chapman & Hall/CRC, 1997.

[35] Harry Joe. *Dependence modeling with copulas*. Chapman and Hall/CRC, 2014.

[36] Matthias Killiches, Daniel Kraus, and Claudia Czado. Model distances for vine copulas in high dimensions. *Statistics and Computing*, pages 1–19, 2017.

[37] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[38] Vaibhav Kulkarni, Natasa Tagasovska, Thibault Vatter, and Benoit Garbinato. Generative models for simulating mobility trajectories. 2018.

[39] Dorota Kurowicka and Harry Joe. *Dependence Modeling*. World Scientific Publishing Company, Incorporated, 2010. ISBN 978-981-4299-87-9.

[40] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[41] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[42] Haoran Li, Li Xiong, and Xiaoqian Jiang. Differentially Private Synthesization of Multi-Dimensional Data using Copula Functions. In *Proc. of the 17th International Conference on Extending Database Technology*, number c, pages 475–486, 2014.

[43] Han Liu, John Lafferty, and Larry Wasserman. The Nonparanormal: semiparametric estimation of high dimensional undirected graphs. *JMLR*, 10:2295–2328, 2009.

[44] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015.

[45] David Lopez-Paz. *From Dependence to Causation*. PhD thesis, University of Cambridge, 2016.

[46] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *ICLR*, 2016.

[47] David Lopez-Paz, J M Hernandez-Lobato, and Bernhard Schölkopf. Semi-supervised domain adaptation with copulas. *NeurIPS*, 2013.

[48] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. In *ICLR*, 2016.

[49] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *ICLR*, 2016.

[50] Thomas Nagler and Claudia Czado. Evading the curse of dimensionality in nonparametric density estimation with simplified vine copulas. *Journal of Multivariate Analysis*, 151:69–89, 2016.

[51] Thomas Nagler and Thibault Vatter. vinecopulib: High Performance Algorithms for Vine Copula Modeling in C++, 2017.

[52] Thomas Nagler and Thibault Vatter. *kde1d: Univariate Kernel Density Estimation*, 2018. R package version 0.2.1.

[53] Thomas Nagler and Thibault Vatter. rvinecopulib: high performance algorithms for vine copula modeling, 2018.

[54] Thomas Nagler, Christian Schellhase, and Claudia Czado. Nonparametric estimation of simplified vine copula models: comparison of methods. *Dependence Modeling*, 5(1):99–120, 2017.

[55] Thomas Nagler, Christian Schellhase, and Claudia Czado. Nonparametric estimation of simplified vine copula models: comparison of methods. *Dependence Modeling*, 5(1):99–120, 2017.

[56] Roger B Nelsen. *An introduction to copulas*. Springer Science & Business Media, 2007.

[57] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[58] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[59] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. *Proceedings - 3rd IEEE International Conference on Data Science and Advanced Analytics*, pages 399–410, 2016.

[60] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[61] R Core Team. R: A language and environment for statistical computing, 2017.

[62] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2015.

[63] Alfréd Rényi. On measures of dependence. *Acta mathematica hungarica*, 10(3-4):441–451, 1959.

[64] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

[65] Murray Rosenblatt. Remarks on a multivariate transformation. *The annals of mathematical statistics*, 23(3):470–472, 1952.

[66] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NeurIPS*, 2016.

[67] Olivier Scaillet, Arthur Charpentier, and Jean-David Fermanian. The estimation of copulas: Theory and practice. Technical report, Ensae-Crest and Katholieke Universiteit Leuven, NP-Paribas and Crest; HEC Geneve and Swiss Finance Institute, 2007.

[68] Boris Schäling. *The Boost C++ Libraries*. 2011.

[69] Ulf Schepsmeier and Jakob Stöber. Derivatives and Fisher information of bivariate copulas. *Statistical Papers*, 55(2):525–542, May 2014.

[70] Simon J Sheather and Michael C Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 683–690, 1991.

[71] A. Sklar. Fonctions de Répartition à n Dimensions et Leurs Marges. *Publications de L'Institut de Statistique de L'Université de Paris*, 8:229–231, 1959.

[72] Jakob Stöber and Claudia Czado. Sampling Pair Copula Constructions with Applications to Mathematical Finance. In Jan-Frederik Mai and Matthias Scherer, editors, *Simulating Copulas: Stochastic Models, Sampling Algorithms and Applications*, Series in quantitative finance. World Scientific Publishing Company, Incorporated, 2012.

[73] Jakob Stöber and Ulf Schepsmeier. Estimating standard errors in regular vine copula models. *Computational Statistics*, 28(6):2679–2707, 2013.

[74] Natasa Tagasovska, Thibault Vatter, and Valérie Chavez-Demoulin. Nonparametric quantile-based causal discovery. *arXiv:1801.10579*, 2018.

[75] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *ICLR*, 2015.

[76] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *ICLR*, 2018.

[77] Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. Adagan: Boosting generative models. In *NeurIPS*, pages 5424–5433, 2017.

[78] Dustin Tran, David M Blei, and Edoardo M Airoldi. Copula variational inference. In *NeurIPS*, 2015.

[79] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103, 2008.

[80] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[81] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, 2016.

[82] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755*, 2018.

Appendix

A Introduction to (vine) copulas

A.1 Copulas

Recall that the components of the random vector $(X_1, \dots, X_d)$ are said to be independent if and only if its joint distribution F is given by the product of the d marginals F_i for $i \in \{1, \dots, d\}$, that is

$$F(x_1, \dots, x_d) = \prod_{i=1}^{d} F_i(x_i), \tag{3}$$

for any $(x_1, \dots, x_d) \in \mathcal{R}^d$. If the random variables are absolutely continuous, then differentiating (3) with respect to $(x_1, \dots, x_d)$ implies that a similar statement hold for the densities, that is

$$f(x_1, \dots, x_d) = \prod_{i=1}^{d} f_i(x_i), \tag{4}$$

where f is the joint density, and f_i for $i \in \{1, \dots, d\}$ are the marginal densities.

However, when the variables are dependent, this statement is no longer true. In this case, the celebrated Sklar's theorem (see Theorem 1 for the precise statement) says that the joint distribution can be written as

$$F(x_1, \dots, x_d) = C(F_1(x_1), \dots, F_d(x_d)), \tag{5}$$

where C is a *copula* that acts as a coupling mechanism between the d marginals.

Definition 1. *A d-dimensional copula* copula *is a multivariate cumulative distribution function $C : [0, 1]^d \to [0, 1]$ for which all the marginal distributions are uniform.*

In other words, for $d = 2$, C is a distribution such that $C(1, u) = C(u, 1) = u$ for any $u \in [0, 1]$. Note that the simplest copulas is arguably the independence copula, namely plugging $C(u_1, \dots, u_d) = \prod_{i=1}^{d} u_i$ into (5) leads to (3).

An intuitive way to understand the copula corresponding to a given joint distribution F and marginal distributions F_i for $i \in \{1, \dots, d\}$ is as the distribution of the so-called probability integral transform (PIT) of the marginals.

Definition 2. *The* probability integral transform *(PIT) of a random variable X with distribution F_X is the random variable $U = F_X(X)$.*

Because the PIT of any random variable is uniformly distributed[2], the joint distribution of the vector of PITs $(U_1, \dots, U_d)$ with $U_i = F_i(X_i)$ for $i \in \{1, \dots, d\}$ is a copula, namely C. A similar idea has an important consequence when one aims at sampling from the joint distribution F. Because it is well known that, if $U \sim \mathcal{U}[0, 1]$ and F_X^{-1} is the inverse cumulative distribution of X, then $F_X^{-1}(U) \sim X$, transforming samples from C into samples from F is straightforward: if $(U_1, \cdots, U_d) \sim C$, then $X_i = F_i^{-1}(U_i)$ for $i \in \{1, \dots, d\}$ implies that $(X_1, \dots, X_d) \sim F$. While it looks like simply transforming a d-dimensional sampling problem into another d-dimensional sampling problem, vine copulas represent a model class for C that is flexible and yet easy to sample from.

Viewing any joint distribution through this copula lens further yields a useful factorization: differentiating (5) with respect to $(x_1, \dots, x_d)$ leads to

$$f(x_1, \dots, x_d) = \frac{\partial^d F(x_1, \dots, x_2)}{\partial x_1 \cdots \partial x_d} = \frac{\partial^d C(u_1, \dots, u_d)}{\partial u_1 \cdots \partial u_d} \prod_{i=1}^{d} \frac{\partial F_i(x_i)}{\partial x_i} = c(u_1, \dots, u_d) \prod_{i=1}^{d} f_i(x_i), \tag{6}$$

where c is the so-called *copula density*, and $u_i = F_i(x_i)$ for $i \in \{1, \dots, d\}$. Hence, we can see that the joint density factorize into a product between the marginal densities, similarly as in (4), with the copula density, which encodes the dependence. Taking the logarithm on both sides of (6), one obtains

$$\log f(x_1, \dots, x_d) = \log c(u_1, \dots, u_d) + \sum_{i=1}^{d} \log f_i(x_i).$$

[2] $\mathbb{P}[U \leq u] = \mathbb{P}[F_X(X) \leq u] = \mathbb{P}[X \leq F_X^{-1}(u)] = F_X(F_X^{-1}(u)) = u$

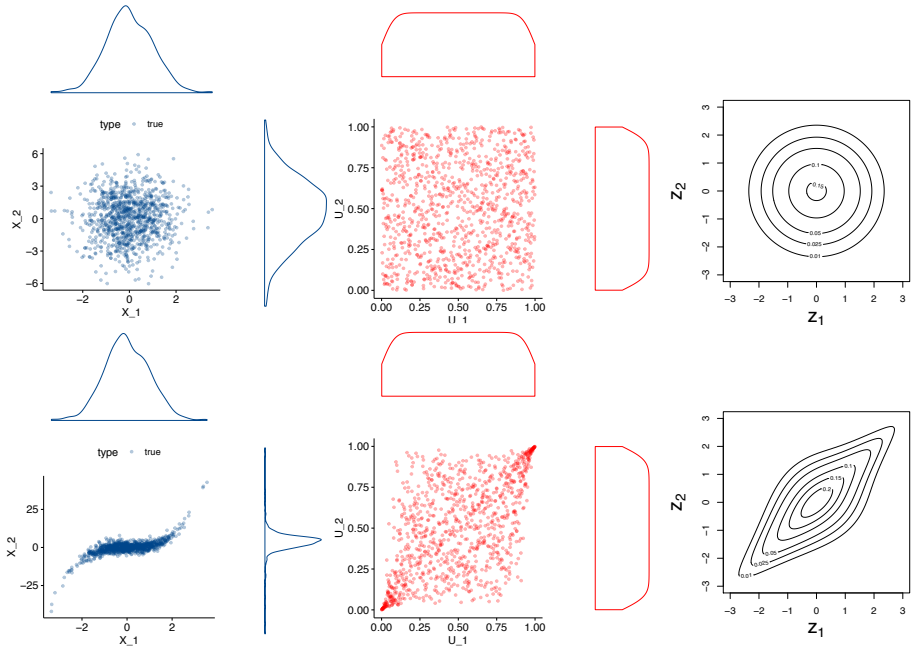


Figure 9: Copula estimation. By row - (top) independent, (bottom) dependent variables. By column - (left) original data, (middle) pseudo-data after PIT, (right) estimated copula density.

In other words, the factorization implies that the joint log-likelihood is the sum of the marginal log-likelihoods and the copula log-likelihood. This observation can be conveniently leveraged for estimation via a two-step procedure where f_i is first estimated by $\widehat{f}_i$ for $i \in \{1, \dots, d\}$. Then, pseudo-observations of the copula are recovered using the estimated PITs, that is $u_i \approx \widehat{F}_i(x_i)$ for $i \in \{1, \dots, d\}$, and c is then estimated by $\widehat{c}$ using the pseudo-sample. This procedure is exemplified in Figure 9.

To summarize, copulas are a tool allowing to represent any multivariate distribution through the individual variables' marginal behaviors as well as their inter-dependencies. While lesser known in the machine learning community, copulas have been widely exploited by in other fields, from economics to quantitative finance, insurance and environmental sciences; in particular when capturing the joint tail behavior is of high importance. In financial risk management for instance, so-called tail events can trigger large and simultaneous losses (or gains) on portfolios. Consequently, multiple parametric copula families have been studied to capture lower/upper tail dependence, or no tail dependence at all. Similarly, other families have been developed to handle asymmetries or other dependence patterns. But such parametric families, which usually imply that the dependence between all pairs of variables is of the same kind, are seldom flexible enough in higher dimensions. Such limitations have led to the development of *pair-copulas constructions* (PCCs) or *vines* - hierarchical structures which allow to flexibly model high dimensional distributions by decomposing the dependence structure into *pairs of (bivariate) copulas*.

A.2 Vines

According to [34, 4, 16], any copula density can be decomposed into a product of $\frac{d(d-1)}{2}$ bivariate (conditional) copula densities. While a decomposition is not unique, it can be organized as a graphical model, a sequence of $d-1$ nested trees, called *regular vine*, *R-vine*, or simply *vine*. Denoting $T_m = (V_m, E_m)$ with V_m and E_m the set of nodes and edges of tree m for $m = 1, \dots, d-1$, the sequence is a vine if it satisfies the following set of conditions guaranteeing that the decomposition leads to a *valid joint density*:

- T_1 is a tree with nodes $V_1 = \{1, \dots, d\}$ and edges E_1.
- For $m \geq 2$, T_m is a tree with nodes $V_m = E_{m-1}$ and edges E_m.
- (Proximity condition) Whenever two nodes in $T_m + 1$ are joined by an edge, the corresponding edges in T_m must share a common node.

The corresponding tree sequence is then called the *structure* of the PCC and has important implications to design efficient algorithms for the estimation and sampling of such models.

Each edge e is associated to a bivariate copula $c_{j_e,k_e|D_e}$ (a so-called *pair-copula*), with the set $D_e \in \{1, \cdots, d\}$ and the indices $j_e, k_e \in \{1, \cdots, d\}$ forming respectively its *conditioning set* and the *conditioned set*. Finally, the joint copula density can be written as the product of all pair-copula densities

$$c(u_1, \cdots, u_d) = \prod_{m=1}^{d-1} \prod_{e \in E_m} c_{j_e,k_e|D_e}(u_{j_e|D_e}, u_{k_e|D_e}), \tag{7}$$

where

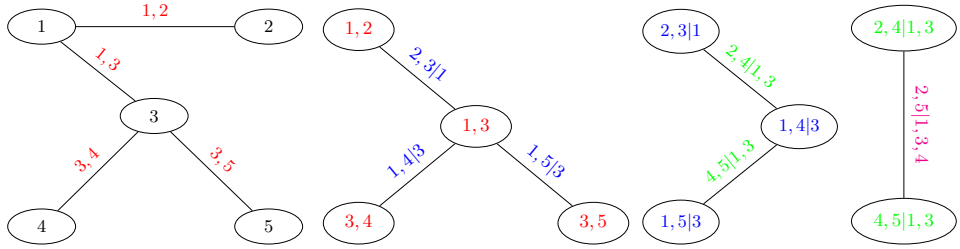
$$u_{j_e|D_e} = \mathbb{P}\left[U_{j_e} \leq u_{j_e} \mid \boldsymbol{U}_{D_e} = \boldsymbol{u}_{D_e}\right],$$

and similarly for $u_{j_e|D_e}$, with $\boldsymbol{U}_{D_e} = \boldsymbol{u}_{D_e}$ understood as component-wise equality for all components of $(U_1, \ldots, U_d)$ and $(u_1, \ldots, u_d)$ included in the conditioning set D_e. In Example Example 1 we present a full example of an R vine for a 5 dimensional density.

Example 1. *The density of a PCC corresponding to the tree sequence in Figure 10 is*

$$c = c_{1,2}\, c_{1,3}\, c_{3,4}\, c_{3,5}\, c_{2,3|1}\, c_{1,4|3}\, c_{1,5|3} c_{2,4|1,3}\, c_{4,5|1,3}\, c_{2,5|1,3,4}, \tag{8}$$

where the colors correspond to the edges E_1, E_2, E_3, E_4.



(a) Tree T_1 (b) Tree T_2 (c) Tree T_3 (d) Tree T_4

Figure 10: A vine tree sequence: the numbers represent the variables, x, y the bivariate distribution of x and y, and $x, y|z$ the bivariate distribution of x and y conditional on z. Each edge corresponds to a bivariate pair-copula in the PCC.

To summarize this section, in order to construct a vine, one has to choose two components:

- The structure, namely the set of trees $T_m = (V_m, E_m)$ for $m = 1, \ldots, d-1$.
- The pair-copulas, namely the models for $c_{j_e,k_e|D_e}$ for $e \in E_m$ and $m = 1, \ldots, d-1$.

To fix ideas, it is easier to start by assuming the structure to be known.

A.2.1 Estimating the pair-copulas

To answer how one could estimate the pair-copulas is closely related whether one can evaluate the density in (7): if one can evaluate the density, then taking it's logarithm and finding the MLE would be straightforward. While it would be impractical for high-dimensional data, the factorization as a product of pair-copulas paves the way for a sequential procedure. Indeed, taking the logarithm of both sides of (7), we have

$$\log c(u_1, \cdots, u_d) = \sum_{m=1}^{d-1} \sum_{e \in E_m} \log c_{j_e,k_e|D_e}(u_{j_e|D_e}, u_{k_e|D_e}). \tag{9}$$

One can thus use (9) to proceed in a tree-wise fashion, starting with $m = 1$, with all pairs in a given tree, that is $e \in E_m$, being estimated in parallel.

Assuming the marginal distributions to be known, one can simply proceed with pseudo-observations $(U_1, \cdots, U_d)$ with $U_i = F_i(X_i)$ to estimate the pairs in the first tree (i.e., when $m = 1$). It works because, for those pairs, the conditioning set is empty, that is $D_e = \emptyset$. When the marginal distributions are unknown, one can proceed similarly using $\widehat{F}_i(X_i)$. But for the higher trees (i.e., when $m > 1$), the decomposition involves conditional distributions like $U_{j_e}|\boldsymbol{U}_{D_e}$ with a non-empty conditioning set, that is $D_e \neq \emptyset$.

It turns out that the arguments for pair-copulas in any tree $m > 1$ can be expressed recursively using conditional distributions corresponding to bivariate copulas in the previous tree (i.e., $m-1$) as follows. Let $e \in E_m$ be an edge of tree m and $l_e \in D_e$ be another index such that $c_{j_e,l_e|D_e \setminus l_e}$ is a pair-copula in tree $m-1$, and define $D'_e = D_e \setminus l_e$. Then we have that

$$u_{j_e|D_e} = h_{j_e,l_e|D'_e}(u_{j_e|D'_e}, u_{l_e|D'_e})$$

where the so-called h-function is defined as

$$h_{j_e,l_e|D'_e}(u_1,u_2) := \int_0^{u_1} c_{j_e,l_e|D'_e}(v,u_2)dv = \frac{\partial C_{j_e,l_e|D'_e}(u_1,u_2)}{\partial u_2}.$$

In each step of this recursion the conditioning set D_e is reduced by one element, until we eventually reach the first tree with $D_e = \emptyset$. Note that, in a vine, for any edge e, the existence of an index l_e such that $c_{j_e,l_e|D_e \setminus l_e}$ is a pair-copula in tree $m-1$ is guaranteed. This allows us to write any of the required conditional distributions as a recursion over h-functions that directly linked to the pair-copula densities in previous trees. As such, assuming the structure to be known, a sequential algorithm to estimate the pair-copulas can be described as follow:

1. Set $m = 1$ and estimate all pair-copulas for the first tree using $(U_1, \cdots, U_d)$.
2. Set $m = m + 1$ and compute the conditional distributions $u_{j_e|D_e}$ and $u_{k_e|D_e}$ for $e \in E_m$.
3. Estimate all pair-copulas in tree m using $u_{j_e|D_e}$ and $u_{k_e|D_e}$ for $e \in E_m$.
4. If $m = d-1$, all pairs have been estimated. Otherwise, go to step 2.

The procedure is generic in the sense that it can be used with any bivariate copula estimator, and we refer to Algorithm 1 in [54] for its pseudocode. Note that the decomposition can also be truncated by replacing the termination condition at step 4 using any truncation level smaller than $d-1$. Finally, for each pair-copula, one could also estimate different models at step 3 and select the best one according to some suitable criterion (e.g., AIC or BIC). One important question that we brushed aside is: given that the structure is generally unknown, how can we also select it?

A.2.2 Selecting the structure

To learn the structure for a dataset where it is unknown, multiple solutions have been proposed. In this paper, as it is most common in the vine literature, we use the so-called Dissmann algorithm, first proposed in [19]: This algorithm represents a greedy heuristic aiming at capturing higher dependencies in the lower trees. The intuition is that higher-tree represent higher-order interactions, which are harder to estimate. As such, one should prioritize modeling the most important patterns in lower trees. This is achieved by finding the maximum spanning tree (MST) using a dependence measure as edge weights. For instance, the absolute value of the empirical Kendall's τ for monotone dependencies or the maximal correlation [63] for more general patterns are popular choices. To compute the MST, most implementations use Prim's algorithm [60]. Letting τ denote a generic bivariate dependence measure, the sequential algorithm mentioned above can thus be modified in a straightforward manner:

1. Set $m = 1$ and compute the dependence $\tau(U_i, U_j)$ for all pairs $1 \leq i < j \leq d$. While this defines a complete graph, only keep the edges corresponding to the MST in E_m. Finally, estimate all pair-copulas for the first tree as before.
2. Set $m = m + 1$ and compute the conditional distributions $u_{j_e|D_e}$ and $u_{k_e|D_e}$, as well as the dependence $\tau(u_{j_e|D_e}, u_{k_e|D_e})$ for all pairs where e is an edge allowed by the proximity condition. Only keep the edges corresponding to the MST in E_m.
3. Estimate all pair-copulas in tree m using $u_{j_e|D_e}$ and $u_{k_e|D_e}$ for $e \in E_m$.
4. If $m = d-1$, all pairs have been estimated. Otherwise, go to step 2.

Note that step 2 can be implemented efficiently by observing that, while conditional distributions might appear in multiple candidate edges, they can be computed only once and stored for further use. The resulting estimation and structure selection procedure is summarized in Algorithm 2 of [54].

A.3 Assumptions for the consistency and asymptotic normality of the kernel bivariate copula estimator

(B1) $\partial_u C(u,v)$ and $\partial_{uu} C(u,v)$ exist and are continuous on $(u,v) \in (0,1) \times [0,1]$, and there exists a constant Q_1 such that $|\partial_{uu} C(u,v)| \leq Q_1/u(1-u)$ for $(u,v) \in (0,1) \times [0,1]$.

(B2) $\partial_v C(u,v)$ and $\partial_{vv} C(u,v)$ exist and are continuous on $(u,v) \in [0,1] \times (0,1)$, and there exists a constant Q_2 such that $|\partial_{vv} C(u,v)| \leq Q_2/v(1-v)$ for $(u,v) \in [0,1] \times (0,1)$.

(B3) The density $c(u,v) = \partial_{uv} C(u,v)$ admits continuous second-order partial derivatives in $(0,1)^2$ and there exists a constant Q_0 such that, for $(u,v) \in (0,1)^2$, $c(u,v) \leq Q_0 \min\left(\frac{1}{u(1-u)}, \frac{1}{v(1-v)}\right)$.

B The VCAE algorithm

The algorithm for vine copula autoencoders is given in Algorithm 1.

Algorithm 1 Vine Copula Autoencoder

Input: train set X of $\{x_1, x_2, ...x_n\}$ images.
1. Train AE component with X:
$f \leftarrow encoder$
$g \leftarrow decoder$

2. Encode train set with f :
$\phi(X) \leftarrow f(X)$

3. Fit a vine copula c using encoded features:
$c \leftarrow \{\phi_1, \phi_2, ...\phi_n\}$ (as described in Section 2.2 and Section 2.3).

4. Sample random observations form c:
$\phi' \leftarrow c(\phi)$ (as in Section 2.4)

5. Decode the random features:
$X' \leftarrow g(\phi')$

Output: generated images X'.

B.1 Variations of VCAE

Conditional VCAE Since the vine estimation and the AE training are independent in our approach, we can do steps 3–5 in Algorithm 1 per class label (fit a vine per class feature) which makes the implementation of Conditional VCAE straightforward.

DEC-VCAE For the implementation of the DEC-VCAE we followed the instructions from the authors in [81]. A difficulty with AEs is that the encoded features are typically entangled, even when the AE reconstruction is accurate. Therefore we enforce some clustering. We start with an pre-trained AE and then optimize a two-term loss function: the clustering and the reconstruction loss.

C Additional experiments

C.1 Toy datasets

Similarly to related generative model literature [26, 77], we test our method on two-dimensional toy datasets. Since this is a 2D case, we use bivariate copulas with nonparametric marginal densities for the estimation and sampling. The three datasets are ring of isotropic Gaussians with 8 modes, 5×5 grid of isotropic Gaussians and the swiss roll dataset. These datasets have proven to be challenging for GANs due to the mode collapse issues [26, 77]. They motivate how the flexibility of nonparametric copulas can be leveraged, and we additionally compare to a baseline Gaussian copula. From Figure 11, we observe the benefits of using nonparametrics; while fitting such datasets is easy, it is clear that the Gaussian assumption is not suitable in such cases (except for the grid of Gaussians).

We further confirm this quantitatively in Table 2, where we repeat the experiment on 100 random datasets of each type, and present the average and standard deviations for both copula families. To evaluate the sampled images, additionally to the MMD, we use the negative log-likelihood (NLL)

Table 2: Evaluation on toy datasets for nonparametric and Gaussian copula. Average and standard deviations from 100 repetitions.

	Ring	Grid	Swiss roll
nonparametric			
NLL ↑	-2.47(0.15)	-3.77(0.2)	-5.23(0.05)
Coverage ↑	0.93(0.02)	0.94(0.02)	0.99(0.01)
MMD ↓	0.18(0.02)	0.15(0.16)	0.32(0.03)
Gaussian			
NLL ↑	-2.98(0.05)	-3.34(0.07)	-6.21(0.05)
Coverage ↑	0.95(0.02)	0.96(0.014)	0.93(0.03)
MMD ↓	0.33(0.02)	0.14(0.02)	0.38(0.02)

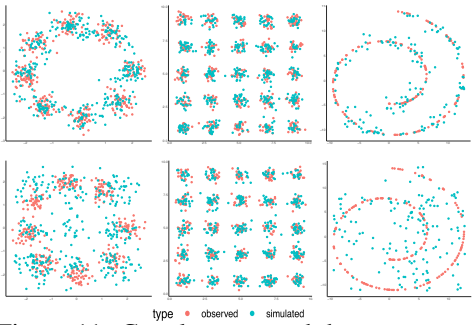


Figure 11: Copula generated data - top row nonparametric, bottom row Gaussian copula.

and *coverage*[3], a closely related metric [77]. As expected, nonparametrics provide better samples according to the three two-sample metrics.

C.2 Various evaluation metrics for the MNIST and CelebA dataset

In Figure 12 and Figure 13, we present the evaluation scores for all baselines on the **MNIST** and **CelebA** datasets. Note that, in the evaluation framework that we use [82], the Inception Score and FID are based on ImageNet features. Therefore those scores are not suitable for binary images and excluded from Figure 12.

The results in Figure 13 show that DCGAN has a slight advantage over VAE and VCAE when methods are evaluated in feature space, while VCAE outperforms VAE on all metrics. In this experiment we used adaptive learning rate for DCGAN [4] to evaluate the scores on more than 30 epochs.

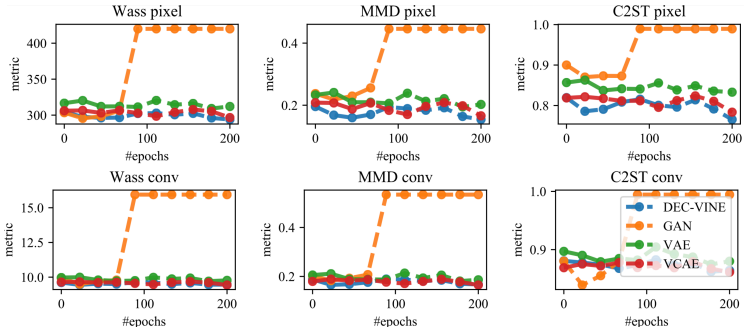


Figure 12: Various evaluation scores for all baselines on the **MNIST** dataset.

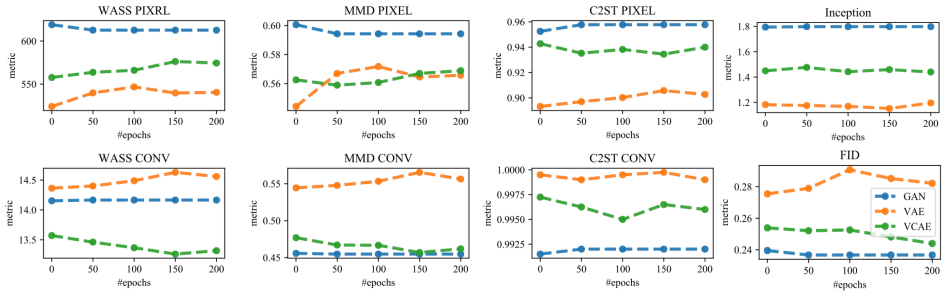


Figure 13: Various evaluation scores for all baselines on the **CelebA** dataset.

[3] Coverage measures the probability mass of the true data covered by the approximate density of the learned model as $C := \mathbb{P}_{\text{data}}[d\mathbb{P}_{\text{model}} > t]$ where t is selected such that $\mathbb{P}_{\text{model}}[d\mathbb{P}_{\text{model}} > t] = \alpha$ and where $d\mathbb{P}_{\text{model}}$ denotes the model density function. We set $\alpha = 0.95$ as in the original paper.

[4] reducing the learning rate by 10 after 30th epoch

C.3 Interpolation in latent space

Figure 14 shows that the transitions for VCAE are smooth and without any sharp changes or unexpected samples in-between when walking the latent space by linear interpolation between two test samples as in [62]. This is not explicitly related to VCAE generative models since we do not train an end-to-end model, however it is important to show that the AE network we use did not simply memorize images.

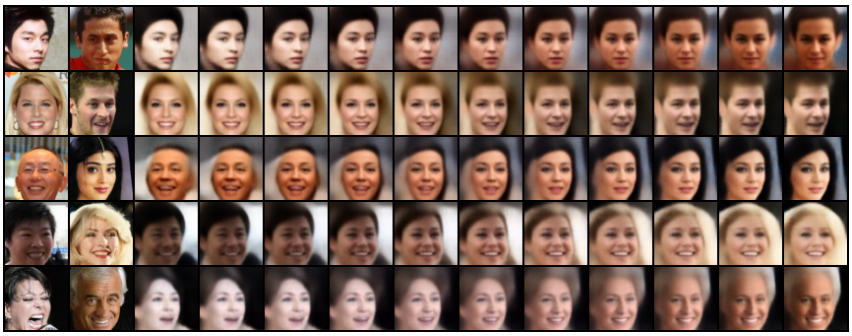


Figure 14: Interpolation in latent space between two real samples (shown in the first two columns) with a VCAE trained on **CelebA**

C.4 The trade-off between time complexity and sample quality

To explore the effect of the choice for truncation level, i.e. the depth of the vine (number of trees) over the quality of the produced VCAE samples, we include an ablation study on the FashionMNIST dataset [80]. The quantitative and qualitative evaluation in Figure 15 and Figure 16 suggest that higher level of truncation provide better samples, at the expected cost of longer computation times.

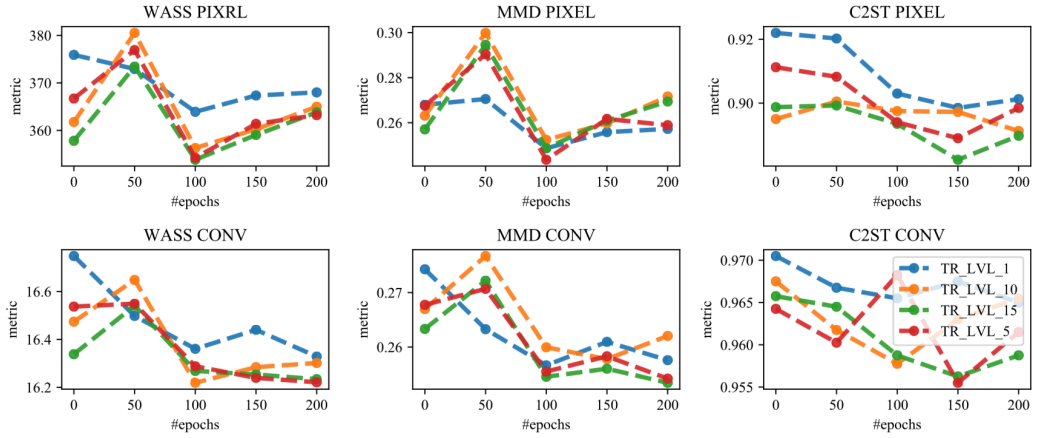


Figure 15: Quantitative evaluation of various truncation levels for the VCAE on **FashionMNIST**.

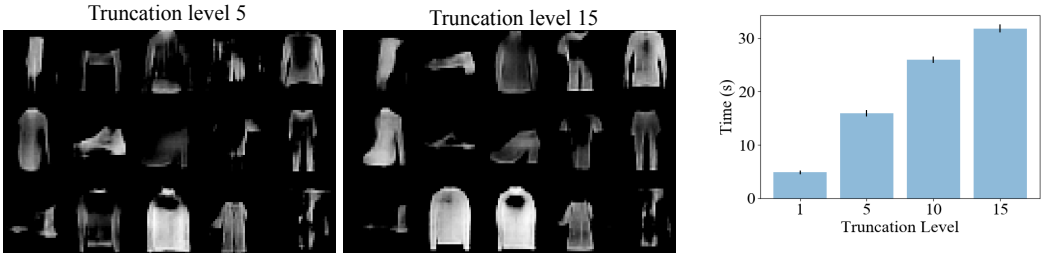


Figure 16: Qualitative evaluation of various truncation levels (left panel) and computation time with respect to the vine depth (right panel) for the VCAE on **FashionMNIST**.

D Additional details on the experiments

We use the same AE architecture for VCAE, DEC-VCAE and VAE as described below. All the AEs were trained by minimizing the Binary Cross Entropy Loss.

D.1 MNIST

The only transformation performed on this dataset is a padding of 2. By doing so we are able to use the same architecture for multiple datasets. We use CNNs for the encoder and the decoder whose architectures are as follows:

- Encoder:

$$\begin{aligned} x \in R^{32\times 32} &\rightarrow Conv_{32} \rightarrow BN \rightarrow ReLU \\ &\rightarrow Conv_{64} \rightarrow BN \rightarrow ReLU \\ &\rightarrow Conv_{128} \rightarrow BN \rightarrow ReLU \\ &\rightarrow FC_{10} \end{aligned}$$

- Decoder:

$$\begin{aligned} z \in R^{10} \rightarrow FC_{100} &\rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ &\rightarrow ConvT_{64} \rightarrow BN \rightarrow ReLU \\ &\rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ &\rightarrow FC_1 \end{aligned}$$

- DCGAN Generator:

$$\begin{aligned} z \in R^{100} \rightarrow\rightarrow ConvT_1 &\rightarrow BN \rightarrow ReLU \\ &\rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ &\rightarrow ConvT_{64} \rightarrow BN \rightarrow ReLU \\ &\rightarrow ConvT_{32} \rightarrow BN \rightarrow ReLU \\ &\rightarrow ConvT_{16} \rightarrow BN \rightarrow ReLU \\ &\rightarrow Tanh_1 \end{aligned}$$

- DCGAN Discriminator:

$$\begin{aligned} Conv_1 &\rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{16} &\rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{32} &\rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{64} &\rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{128} &\rightarrow BN \rightarrow LeakyReLU \\ &\rightarrow Sigmoid_1 \end{aligned}$$

with all (de)convolutional layers have 4×4 filters, a stride of 2, and a padding of 1. We use BN to denote batch normalization and ReLU for rectified linear units and FC for fully connected layers. We denote $Conv_k$ the convolution with k filters. Leaky ReLU was used with negative slope = 0.2 everywhere.

D.2 SVHN

For SVHN we use the data as is without any preprocessing. The architectures are:

- Encoder:

$$\begin{aligned} x \in R^{3\times 32\times 32} &\rightarrow Conv_{64} \rightarrow BN \rightarrow LeakyReLU \\ &\rightarrow Conv_{128} \rightarrow BN \rightarrow LeakyReLU \\ &\rightarrow Conv_{256} \rightarrow BN \rightarrow LeakyReLU \\ &\rightarrow FC_{100} \rightarrow FC_{20} \end{aligned}$$

- Decoder:

$$\begin{aligned} z \in R^{20} \rightarrow FC_{100} \rightarrow ConvT_{256} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{64} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{32} \rightarrow BN \rightarrow ReLU \\ \rightarrow FC_1 \end{aligned}$$

- DCGAN Generator:

$$\begin{aligned} z \in R^{100} \rightarrow ConvT_{256} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{64} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{32} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{3} \rightarrow BN \rightarrow ReLU \\ \rightarrow Tanh_1 \end{aligned}$$

- DCGAN Discriminator:

$$\begin{aligned} Conv_3 \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{32} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{64} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{128} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{256} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Sigmoid_1 \end{aligned}$$

where all (de)convolutional the layers have 4×4 filters, a stride of 2, and a padding of 1. The rest of the notations are the same as before.

D.3 CelebA

For CelebA we first took central crops of 140×140 and then resized to resolution 64×64. Note that only Fig. 9 in the main text is not a result of this preprocessing. The architectures used are as follows:

- Encoder:

$$\begin{aligned} x \in R^{3 \times 64 \times 64} \rightarrow Conv_{64} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{128} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{256} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{512} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow FC_{100} \rightarrow FC_{100} \end{aligned}$$

- Decoder:

$$\begin{aligned} z \in R^{100} \rightarrow FC_{100} \rightarrow ConvT_{512} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{256} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{64} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{32} \rightarrow BN \rightarrow ReLU \\ \rightarrow FC_1 \end{aligned}$$

- DCGAN Generator:

$$\begin{aligned} z \in R^{100} \rightarrow ConvT_{512} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{256} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{128} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{64} \rightarrow BN \rightarrow ReLU \\ \rightarrow ConvT_{3} \rightarrow BN \rightarrow ReLU \\ \rightarrow Tanh_1 \end{aligned}$$

- DCGAN Discriminator:

$$\begin{aligned} Conv_3 \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{64} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{128} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{256} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Conv_{512} \rightarrow BN \rightarrow LeakyReLU \\ \rightarrow Sigmoid_1 \end{aligned}$$

where all the (de)convolutional layers have 4×4 filters, a stride of 2, and a padding of 1. Padding was set to 0 only for the last convoluitional layer of the encoder and the first layer of the decoder. The rest of the notations are the same as before.

E Code

Our code is available at the following link: https://github.com/tagas/vcae.

F Simulating Mobility Trajectories with copulas

In related work [38], we have also compared to adversarial and recurrent based methods for sampling *sequential data* (artificial mobility trajectories). We evaluate the generated trajectories with respect to their geographic and semantic similarity, circadian rhythms, long-range dependencies, training and generation time. We also include two sample tests to assess statistical similarity between the observed and simulated distributions, and we analyze the privacy trade-offs with respect to membership inference and location-sequence attacks. The results show that copulas surpass all baselines in terms of MMD score and training + simulation time. For more details please see [38].