Neural Likelihoods via Cumulative Distribution Functions

Pawel Chilinski
University College London
pawel.chilinski.14@ucl.ac.uk

Ricardo Silva[*]
University College London and The Alan Turing Institute
ricardo@stats.ucl.ac.uk

Abstract

We leverage neural networks as universal approximators of monotonic functions to build a parameterization of conditional cumulative distribution functions (CDFs). By the application of automatic differentiation with respect to response variables and then to parameters of this CDF representation, we are able to build black box CDF and density estimators. A suite of families is introduced as alternative constructions for the multivariate case. At one extreme, the simplest construction is a competitive density estimator against state-of-the-art deep learning methods, although it does not provide an easily computable representation of multivariate CDFs. At the other extreme, we have a flexible construction from which multivariate CDF evaluations and marginalizations can be obtained by a simple forward pass in a deep neural net, but where the computation of the likelihood scales exponentially with dimensionality. Alternatives in between the extremes are discussed. We evaluate the different representations empirically on a variety of tasks involving tail area probabilities, tail dependence and (partial) density estimation.

1 CONTRIBUTION

We introduce a novel parameterization of multivariate cumulative distribution functions (CDFs) using deep neural networks. We explain how training can be done by a straightforward adaptation of standard methods for neural networks. The main motivations behind our work include: a direct evaluation of tail area probabilities; coherent estimation of low dimensional marginals of a joint distribution without the requirement of fitting a full joint; and supervised/unsupervised density estimation.

[*]Partially supported by EPSRC grant EP/N510129/1.

The first two tasks benefit directly from a CDF computed by a forward pass in a neural network, as tail probabilities and marginal CDFs can be read-off essentially directly in this representation. The latter has been tackled by an increasingly large literature on neural density estimators. This dates back at least to Bishop (1994), who used multilayer perceptrons to encode conditional means, variances, and mixture probabilities for a (conditional) mixture of Gaussians. Recently, models using transformations of a simple distribution into more complex ones were proposed. Dinh et al. (2015),Dinh et al. (2017),Papamakarios et al. (2017), Huang et al. (2018) and De Cao and Titov (2019) are examples of the state of the art for density estimation. They use invertible transformations from simple base distributions where the determinant of the Jacobian is easy to compute and Monte Carlo approaches for computing gradients become feasible. Depending on the architecture, they are optimized for density estimation or sampling. We show we remain competitive against these methods while keeping a comparatively simple uniform structure with few hyperparameters. For instance, compared to the method of Bishop (1994), there is no need to choose the base distribution of the mixture nor the number of mixtures.

All of the above is predicated on how we construct multivariate CDFs. The most direct extension from univariate to multivariate CDFs is conceptually simple, but the calculation of the likelihood grows exponentially in the dimensionality. This is essentially the counterpart to computing a partition function in an undirected graphical model, where here the problem is differentiation as opposed to integration. Compromises are discussed, including the relationship to Gaussian copula models and other CDF constructions based on small dimensional marginals. One extreme sacrifices the ability of representing a CDF by a single forward pass in exchange for scalability to high dimensions, where we can compare it against state-of-the-art neural density estimators.

This paper is organized as follows. Section 2 describes our main approach. Related work is described in Sections 3 and S1. Experiments are discussed in Section 4. We show that our models are competitive for density estimation while able to directly tackle some modelling problems where recent neural-based models could not be applied in an obvious manner.

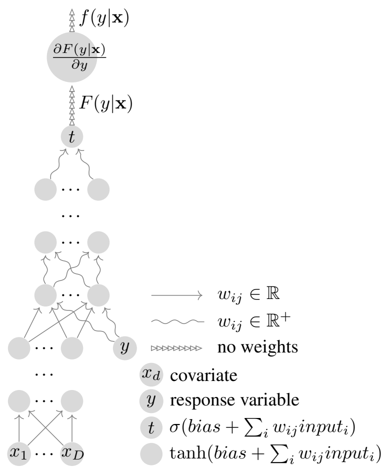
2 THE MONOTONIC NEURAL DENSITY ESTIMATOR

We now introduce the Monotonic Neural Density Estimator (MONDE), inspired by neural network methods for parameterizing monotonic functions. The primary usage of MONDE is to compute conditional CDFs with a single forward pass in a deep neural network, while allowing for the calculation of the corresponding conditional densities by tapping into existing methods for computing derivatives in deep learning. The latter is particularly relevant for likelihood-based fitting methods such as maximum (composite) likelihood. We will focus on the continuous case only, where probability density functions (pdfs) are defined, although extensions to include mixed combinations of discrete and continuous variables are straightforward by considering difference operations as opposed to differentiation operations. We start with the simplest but important univariate case, where dependency between variables does not have to be modelled. We progress through more complex constructions to conclude with the most flexible but computationally demanding case, where we deal with multivariate data without assuming any specific families of distributions for the data generating process.

Here we use the following notation.
$F(\mathbf{y}|\mathbf{x})$: multivariate conditional CDF, where $\mathbf{y} \in \mathbb{R}^K$ is the response vector and $\mathbf{x} \in \mathbb{R}^D$ is a covariate vector;
$F_k(y_k|\mathbf{x})$: k-th marginal conditional CDF;
$f(\mathbf{y}|\mathbf{x})$: multivariate conditional pdf;
$f_k(y_k|\mathbf{x})$: k-th marginal conditional pdf;
w: the set of parameters of a neural network, where w_{ij}^l represents a particular weight connecting two nodes i and j, with i located at layer l of the network.

2.1 UNIVARIATE CASE

The structure of MONDE for univariate responses is sketched in Figure1 as a directed graph[1] with two types



$f(y|\mathbf{x})$
$\frac{\partial F(y|\mathbf{x})}{\partial y}$
$F(y|\mathbf{x})$
t
y
$x_1 \cdots x_D$
$w_{ij} \in \mathbb{R}$
$w_{ij} \in \mathbb{R}^+$
no weights
x_d covariate
y response variable
t $\sigma(bias + \sum_i w_{ij} input_i)$
$\tanh(bias + \sum_i w_{ij} input_i)$

Figure 1: The graph representing Univariate Monotonic Neural Density Estimator computational structure. The last node symbolizes the operation of differentiating parameterized conditional distribution function $F(y|\mathbf{x})$ with respect to the input y. Its output $f(y|\mathbf{x})$ encodes the conditional density function. The legend explains the symbols used.

of edges and a layered structure so that two consecutive layers are fully connected, with no further edges. The definition of layer in this case follows immediately from the topological ordering of the graph. Covariates $x_i, \ldots, x_D$ and response variable y are nodes without parents in the graph, with the layer of the covariates defined to be layer 1. The response variable y is positioned in some layer $1 < l_y < L$, where L is the final layer. Each intermediate node i at layer$l$, h_i^l, returns a non-linear transformation of a weighted sum of all nodes in layer $l-1$. Here, as commonly used in the neural network literature and based on preliminary results from our experiments, we use the hyperbolic tangent function such that $h_i^l = \tanh\left(\sum_{v_j \in \mathcal{V}_{l-1}} w_{ij}^l v_j + w_{i0}^l\right)$, where $1 < l < L$ and $\mathcal{V}_l$ is the set of nodes in layer l. The final layer L consists of a single node $t(y, \mathbf{x}) \equiv \text{sigmoid}\left(\sum_{v_j \in \mathcal{V}_{L-1}} w_{ij}^L v_j + w_{i0}^L\right)$, representing the probability $P(Y \leq y \mid \mathbf{X} = \mathbf{x})$ as encoded by the weights of the neural net. In other words, $t(y, \mathbf{x})$ is interpreted as a CDF $F_w(y \mid \mathbf{x})$ encoded by some w.

Assuming w parameterizes a valid CDF, we can use an automatic differentiation method to generate the density function $f_w(y \mid \mathbf{x})$ corresponding to $F_w(y \mid \mathbf{x})$ by differentiating $t(y, \mathbf{x})$ with respect to y. The same principle behind backpropagation applies here, and in our implementation we use Tensorflow[2] to construct the computation graph that generates $t(y, \mathbf{x})$. Once the pdf is

[1]This graph is to be interpreted as a high-level simplified computation graph as opposed to a graphical model. It does not illustrate a generative process, hence the placement of output random variable y as an input to other variables.

[2]https://www.tensorflow.org

constructed, automatic differentiation can once again be used, now with respect to w, to generate gradients to be plugged into any gradient-based learning algorithm.

To guarantee that $t(y, \mathbf{x})$ is a valid CDF, we must enforce three constraints: (i) $\lim_{y \to -\infty} t(y, \mathbf{x}) = 0$; (ii) $\lim_{y \to +\infty} t(y, \mathbf{x}) = 1$; (iii) $\partial t(y, \mathbf{x})/\partial y \geq 0$. We chose the following design to meet these conditions: for every w_{ij}^l where $v_j \in \mathcal{V}_{l-1}$ is a descendant of y in the corresponding directed graph, enforce $w_{ij}^l \geq 0$. This means that for all layers $l > l_y$, all weights $\{w_{ij}^l\}$ are constrained to be non-negative, while $\{w_{i0}^l\}$, representing bias parameters, are unconstrained. Meanwhile, $w_{ij}^l \in \mathbb{R}$ for $l \leq l_y$. In Figure 1, the constrained weights are represented as squiggled edges. This guarantees monotonicity condition (iii). Due to the range of the logistic function being $[0, 1]$ and $t(y, \mathbf{x})$ being monotonic with respect to y, (i) and (ii) are also guaranteed to some extent: an arbitrary choice of parameter vector w will not imply e.g. that $\lim_{y \to +\infty} t(y, \mathbf{x}) = 1$ since the contribution of y to the final layer is bounded by the tanh nonlinearity. However, by learning w, this limit is satisfied approximately since a likelihood-based fitting method will favour $\sum_{i=1}^n F(y_{max} \mid \mathbf{x}^{(i)})/n \approx 1$[3], where n is the sample size, i indexes the training sample, and y_{max} is the maximum observed training value of Y in the sample. There are ways of "normalizing" $t(y, \mathbf{x})$ so that the limit is achieved exactly for all parameter configurations (see Section 2.4). We have not found it mandatory in order to obtain satisfactory empirical results. This happens even though our "unnormalized" implementation is at a theoretical disadvantage, as it can potentially represent densities with the total mass less than 1. This is verified by the experiments described in Section 4.

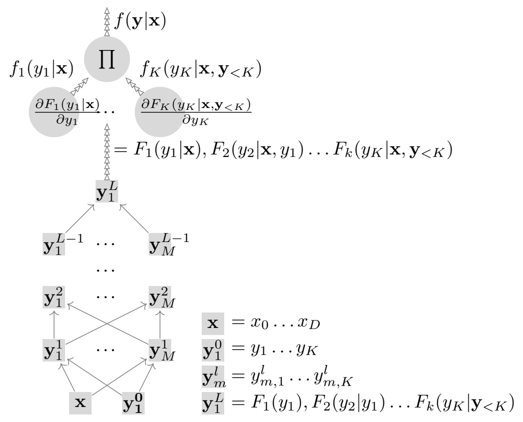
2.2 AUTOREGRESSIVE MONDE

The first variation of the MONDE model, capable of efficiently encoding multivariate distributions, is presented in the Figure 2. It uses a similar approach to univariate output distributions, as described in Section 2.1, to parameterize each factor according to a fully connected probabilistic directed acyclic graph (DAG) model. That is, we assume a given ordering $y_1, \ldots, y_K$ defining the fully connected DAG model:

$$f(\mathbf{y} \mid \mathbf{x}) = \prod_{k=1}^{K} f_k(y_k \mid \mathbf{x}, \mathbf{y}_{<k}), \tag{1}$$

where $\mathbf{y}_{<k}$ is set of response variables with index smaller than k. In theory, the indexing of variables can be chosen

[3]The expression is an empirical estimate of the marginal $F(y_{max})$ which is 1 in the empirical distribution. The claim follows as our estimator is chosen to minimize the KL divergence with respect to the empirical distribution.



$f(\mathbf{y}|\mathbf{x})$
$\prod$
$f_1(y_1|\mathbf{x})$
$f_K(y_K|\mathbf{x}, \mathbf{y}_{<K})$
$\frac{\partial F_1(y_1|\mathbf{x})}{\partial y_1}$
$\frac{\partial F_K(y_K|\mathbf{x},\mathbf{y}_{<K})}{\partial y_K}$
$= F_1(y_1|\mathbf{x}), F_2(y_2|\mathbf{x}, y_1) \ldots F_k(y_K|\mathbf{x}, \mathbf{y}_{<K})$
$\mathbf{y}_1^L$
$\mathbf{y}_1^{L-1}$ $\mathbf{y}_M^{L-1}$
$\mathbf{y}_1^2$ $\mathbf{y}_M^2$
$\mathbf{y}_1^1$ $\mathbf{y}_M^1$
$\mathbf{x}$ $\mathbf{y}_1^0$
$\mathbf{x} = x_0 \ldots x_D$
$\mathbf{y}_1^0 = y_1 \ldots y_K$
$\mathbf{y}_m^l = y_{m,1}^l \ldots y_{m,K}^l$
$\mathbf{y}_1^L = F_1(y_1), F_2(y_2|y_1) \ldots F_k(y_K|\mathbf{y}_{<K})$

Figure 2: Autoregressive Model, MONDE MADE, The Multivariate Monotonic Neural Density Estimator architecture with shared parametrization inspired by MADE. The square nodes of the computational graph contain vectors, to differentiate from the oval nodes representing scalar values.

arbitrarily. In this work, we do not try to optimize it. This type of DAG parameterization was called "autoregressive" in the neural density estimator of Uria et al. (2013), a nomenclature we use here to emphasize that this is a related method. Our implementation of the autoregressive model uses parameter sharing inspired by MADE (Germain et al., 2015).

The input to the computational graph is a K-dimensional vector $\mathbf{y}$ of response variables and a D-dimensional vector $\mathbf{x}$ of covariate variables. These vectors comprise the first layer of the network. Each consecutive hidden layer is an affine transformation of the previous layer proceeded by a nonlinear elementwise map transforming its inputs via sigmoid function. Each hidden layer is composed of M K-dimensional vectors $\mathbf{y}_m^l$, where l indexes the layer and m is vector index within layer l. The affine transformation matrix is constrained so that the k-th element of m-th vector, i.e. $\mathbf{y}_{m,k}^l$, depends on a subset of the elements of the previous layer, i.e. $\mathbf{y}_{.,<k}^{l-1}$, and is monotonically non-decreasing with respect to $\mathbf{y}_{.,k}^{l-1}$. Here, dot . represents all possible indices $m \in \{1, 2, \ldots, M\}$. Monotonicity is preserved using non-negative weights in the respective elements of the transformation matrix.

Finally, the L-th layer consists of a single K-dimensional vector $\mathbf{y}_1^L$, with each element representing a CDF factor, $F_1(y_1), \ldots, F_k(y_k|\mathbf{x}, \mathbf{y}_{<k})$. Each CDF factor is differentiated with respect to its respective response variable to obtain its pdf. The product of all pdf factors provide the density function $f(\mathbf{y}|\mathbf{x})$. We provide the implementation details in the supplement, Section S2.1.

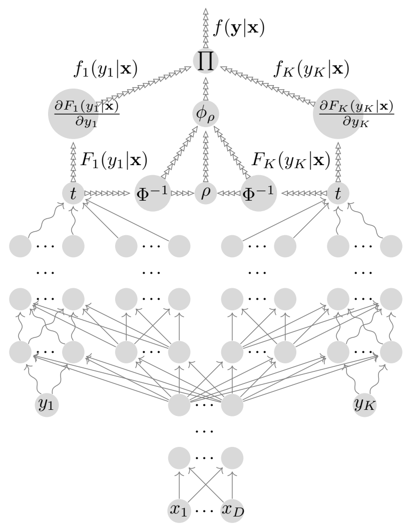


Figure 3: Multivariate Monotonic Neural Density Estimator with Gaussian Copula Dependency and Constant Covariance.

2.3 GAUSSIAN COPULA MODELS

A standard way of extending univariate models to multivariate models is to use a copula model (Sklar, 1959; Schmidt, 2006). In a nutshell, we can write a multivariate CDF $F(\mathbf{y})$ as $F(\mathbf{y}) = P(\mathbf{Y} \leq \mathbf{y}) = P(F^{-1}(F(\mathbf{Y})) \leq \mathbf{y}) = P(\mathbf{U} \leq F(\mathbf{y}))$. Here, $\mathbf{U}$ is a random vector with uniformly distributed marginals in the unit hypercube and $F^{-1}(\cdot)$ is the inverse CDF, applied elementwise to $\mathbf{Y}$, which will be unique for continuous data as targeted in this paper. The induced multivariate distribution with uniform marginals, $P(\mathbf{U} \leq \mathbf{u})$, is called the *copula* of $F(\cdot)$. Elidan (2013) presents an overview of copulas from a machine learning perspective.

This leads to a way of creating new distributions. Starting from a multivariate distribution, we extract its copula. We then replace its uniform marginals with any marginals of interest, forming a *copula model*. In the case of the multivariate Gaussian distribution, the density function

$$f(\mathbf{y}) = \phi_\rho(\Phi^{-1}(F_1(y_1)), \ldots, \Phi^{-1}(F_K(y_K))) \prod_{k=1}^{K} f_k(y_k) \tag{2}$$

is a Gaussian copula model where ϕ_ρ is a Gaussian density function with zero mean and correlation matrix ρ, Φ^{-1} is the inverse CDF of the standard Gaussian, $f_k(\cdot)$ is any arbitrary univariate density function and $F_k(\cdot)$ its respective distribution function. We can show that the k-th marginal of this density is indeed $f_k(\cdot)$.

We extend the density estimator from the previous section to handle a K-dimensional multivariate output $\mathbf{y}$ by exploiting two (conditional) copula variations. The first variation is shown in Figure 3. Weight sharing is done so that all output variables y_k are placed in layer l_y, with all weights $w_{ij}^{l'}$, $l' \leq l_y$, producing transformations of the input $\mathbf{x}$ that is shared by all conditional marginals $F_k(y_k \mid \mathbf{x})$. From layers $l_y + 1, \ldots, L$, the neural network is divided into K disjoint blocks, each composed of two partitions: the first depending monotonically on its respective y_k, and the second depending on shared transformation of $\mathbf{x}$. The first partition depends on the second but not vice versa so monotonicity with respect to y_k is preserved (all the paths from y_k to t_k in the computational graph use non-negative parameters, as shown in the diagram). Each of the K blocks generates output $t_k(y_k, \mathbf{x})$ representing an estimate of the corresponding marginal $F_k(y_k \mid \mathbf{x})$. The k-th marginal pdf can be obtained by applying backpropagation with respect to y_k: $f_k(y_k) = \partial t_k(y_k, \mathbf{x}) / \partial y_k$. Next, the individual marginal distributions evaluated at each training point are transformed via standard normal quantile functions. Such quantiles $\Phi^{-1}(F_k(y_k \mid \mathbf{x}))$ are standard normal variables, which we use to estimate the correlation matrix for the entire training set. The estimated marginals and correlation matrix fully define our model. Taking the product of the estimated copula and estimated marginal densities (as shown in Equation 2) gives us an estimate of the joint density with a correlation matrix that does not change with $\mathbf{x}$ but which is simple to estimate by re-using the univariate MONDE. We call this the Constant Covariance Copula Model.

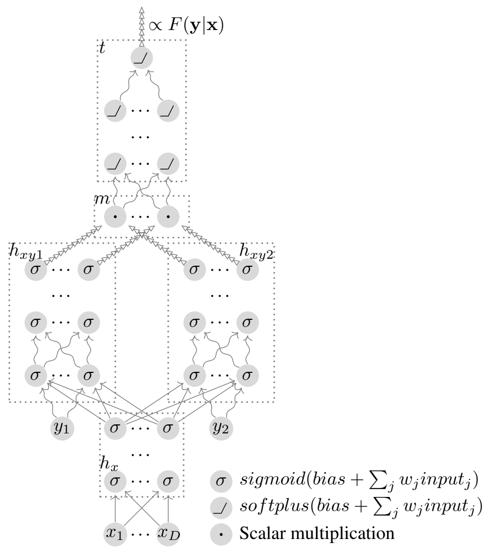
The next improvement, achieved at a higher computational cost, consists of parameterizing the correlation matrix using a covariate transformation. The diagram of this model is presented in Figure S2 in the supplement. This time the correlation matrix is parameterized via a low rank factorization of the covariance matrix which is a function of the covariates, allowing for a model with heteroscedasticity in the copula of the output variables. The correlation matrix parameterization is as follows:

$$\mathbf{\Sigma}(\mathbf{x}) = \mathbf{u}(\mathbf{x}) \cdot \mathbf{u}(\mathbf{x})^T + diag(\mathbf{d}(\mathbf{x})) \tag{3}$$

$$\mathbf{D}(\mathbf{x}) \equiv \sqrt{diag(\mathbf{\Sigma}(\mathbf{x}))}, \tag{4}$$

$$\rho(\mathbf{x}) \equiv \mathbf{D}^{-1}(\mathbf{x}) \cdot \mathbf{\Sigma}(\mathbf{x}) \cdot \mathbf{D}^{-1}(\mathbf{x}), \tag{5}$$

where $\mathbf{\Sigma}(\mathbf{x})$ is the covariate-parameterized low rank covariance matrix; $\mathbf{u}(\mathbf{x}) \in \mathbb{R}^K$ and $\mathbf{d}(\mathbf{x}) \in \mathbb{R}_+^K$ are covariate-parameterized vectors; *diag* is an operator which extracts a diagonal vector from the square matrix or creates a diagonal matrix from a vector (according to context); $\rho(\mathbf{x})$ is the resulting covariate-parameterized correlation matrix.



$\propto F(\mathbf{y}|\mathbf{x})$
t
m
h_{xy1}
h_{xy2}
y_1
y_2
h_x
$x_1 \cdots x_D$
σ $sigmoid(bias + \sum_j w_j input_j)$
$softplus(bias + \sum_j w_j input_j)$
Scalar multiplication

Figure 4: Graph of an "Unnormalized" Distribution Function of PUMONDE, Pure Monotonic Neural Density Estimator. It shows two response variables y_1, y_2 and covariates $\mathbf{x}$ transformed via computational graphs: h_x, h_{xy1}, h_{xy2}, m and t.

2.4 PUMONDE: PURE MONOTONIC NEURAL DENSITY ESTIMATOR

Our final model family is a flexible multivariate CDF parameterization. It can be combined with multivariate differentiation, with respect to multiple response variables, to provide a likelihood function. The higher order derivative $\partial^K t(\mathbf{y}, \mathbf{x})/\partial y_1 \dots \partial y_K$ has to be non-negative so that the model can represent a valid density function[4]. The graph representing a monotone function with respect to each response variable with no finite upper bound (to be later "renormalized") is presented in Figure 4. It is composed of several transformations, each of them represented in the computational graph as dashed rectangle containing the nodes and edges symbolizing its computations: h_x, a transformation the covariates using a standard multilayer network of sigmoid transformations; h_{xyi}, a sequential composition of monotonic nonlinear mappings starting with h_x and response variable y_i; m, the element-wise multiplication $\odot_{i=1}^{K} h_{xyi}$ assuming all h_{xyi} have the same dimensionality; and t, a monotonic transformation with respect to all its inputs that returns a positive real valued scalar. The last transformation t uses only softplus as it non-linear transformations $(\text{softplus}(x) \equiv \log(1 + \exp x))$. The function t is non-decreasing with respect to any response variable on the same premises as previous models.

In this model, we replaced $\tanh$ with sigmoid and softplus, where a hidden unit uses softplus if it has more than one ancestor in $y_1, \dots, y_K$ and sigmoid otherwise. This is because non-convex activation functions such as the sigmoid will not guarantee e.g. $\partial^2 t(\mathbf{y}, \mathbf{x})/\partial y_1 \partial y_2 \geq 0$ for units which have more than one target variable as an ancestor. Higher order derivatives with respect to the same response variable can take any real number because of the properties of the computational graph i.e., using products of non-decreasing functions which are always positive and noting the fact that second order derivative with respect to the same response variable transformed by sigmoid can take any real value.

The density is then computed from the following transformations, here exemplified for a bivariate model:

$$F_w(y_1, y_2 \mid \mathbf{x}) = \frac{t(m(h_{xy1}(y_1, h_x(x)), h_{xy2}(y_2, h_x(x))))}{t(\mathbf{1})}, \tag{6}$$

$$f_w(y_1, y_2 \mid \mathbf{x}) = \frac{\partial^2 F_w(y_1, y_2 \mid \mathbf{x})}{\partial y_1 \partial y_2}. \tag{7}$$

All output elements of $m(\cdot)$ have values in the $[0, 1]$ range because it is element-wise multiplication of vectors with component values in $[0, 1]$. By plugging-in the maximum value $\mathbf{1}$ as input of the t transformation (as shown in denominator of Equation 6) we normalize the output of the distribution estimator F_w to lie within $[0, 1]$ so to output a valid CDF. The guarantee of non-decreasing monotonicity and positiveness of the F_w with respect to each element of $\mathbf{y}$ assures that the range of the proposed estimator of a distribution function is in $[0, 1]$[5].

2.4.1 Composite Log-likelihood

It must be stressed that an unstructured PUMONDE with full connections will in general require an exponential number of steps (as a function of K) for the gradient to be computed, mirroring the problem of computing partition functions in undirected graphical models. Here we explore the alternative with the use of composite likelihood (Varin et al., 2011).

[4]This condition rules out sigmoid as the final transformation of the computational graph for the distribution function because $\partial^2\sigma(z)/\partial^2 z \in \mathbb{R}$, therefore this version of the CDF estimator uses different approach to map its output to be in the $(0, 1)$ range.

[5]The discussion at the end of Section 2.1, about the univariate MONDE not being able strictly attain 0 or 1 is applicable here as well, because of the $h_{xyi}(y_i, \mathbf{x})$ transformation using bounded non-linearities. However, we can modify the initial layer at l_y to simply monotonically map the real line to $[0, 1]$ (or whatever the support of each Y_k is), and do the normalization with respect to the output of l_y having value $\mathbf{1}$, as opposed to the output of m. We decided to omit this in order to make the description of the model simpler, and due to the lack of early evidence that this pre-processing was useful in practice.

We train the PUMONDE model by minimizing the objective composed of the sum of the bivariate negative log-likelihoods (LL) for each pair of response variables (composite likelihood):

$$LL = \sum_{i=1..K, j=1..K, i<j} \log \frac{\partial^2 F_w(y_i, y_j|\mathbf{x})}{\partial y_i \partial y_j}. \quad (8)$$

We compute estimates of such sums over mini-batches of data sampled from the training set. We update parameters using stochastic gradient descent as in other methods presented in this work. In the future, we want to check its role in graphical models for CDFs (Huang and Frey, 2008; Silva et al., 2011). For now we will restrict PUMONDE to small dimensional problems.

3 RELATED WORK

Our work is inspired by the literature on neural networks applied to monotonic function approximation and to density estimation which is reviewed in the supplement Section S1.

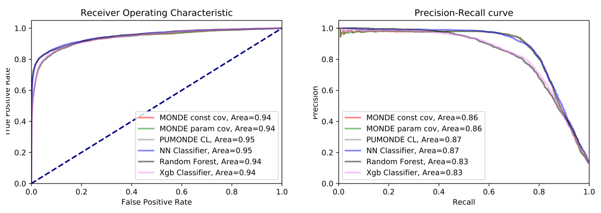
4 EXPERIMENTS

In this section, we describe experiments in which we compare our and baseline models on various datasets and five success criteria. In what follows, Tasks I, III and IV show how MONDE variations are competitive against the state-of-the-art on modelling dependencies. Given that, Tasks II and V advertise the convenience of a CDF parameterization against other approaches. As baselines, depending on the task, we use the following models: RNADE (Uria et al., 2013, 2014), MDN (Bishop, 1994), MADE (Germain et al., 2015), MAF (Papamakarios et al., 2017), TAN (Oliva et al., 2018) and NAF (Huang et al., 2018; De Cao and Titov, 2019). More experiments are included in the supplement, Section S4.

4.1 TASK I: DENSITY ESTIMATION

In this section, we show results on density estimation using UCI datasets. We use the same experimental setup as in (Papamakarios et al., 2017; Huang et al., 2018; De Cao and Titov, 2019) to compare recently proposed learning algorithms to one introduced in this work. In particular, we evaluate a MONDE MADE variant which is described in Section 2.2. It is a simple extension of our MONDE model to multivariate response variables using autoregressive factorization. Among our methods, it is the only viable option to be applied to high dimensional and large datasets that does not make use of a parametric component, as in the Gaussian copula variants. Results are presented in Table 1, which contains test log-likelihoods and error bars of 2 standard deviations on five datasets. MONDE MADE matched the performance of the state of the art NAF model from (Huang et al., 2018) for the POWER dataset, and exceeded the performance of the NAF for the GAS dataset. We achieved slightly worse results on the other UCI datasets but we noticed that our model had a tendency to overfit the training data in these cases. We have not applied techniques that could improve generalization like batch normalization which were used in the baseline models. We conclude that our models, by achieving comparable results and having a complementary inductive bias to the baselines, can be used as yet another tool for the benefit of practitioners.

4.2 TASK II: TAIL EVENT CLASSIFICATION



(a) ROC Curves (b) Precision/Recall Curves

Figure 5: ROC Curves/AUC Scores (Area) and Precision-Recall Curves/Average Precision Scores (Area). RF and Xgb clustered together at a lower TPR and precision - Classification Task (better seen in color).

We tested the Copula MONDE (Section 2.3) and PUMONDE (Section 2.4 and 2.4.1) models on a problem of detecting events falling at the tail of a distribution which, for a fixed threshold defining the tail, can be compared against standard classifiers. We use foreign exchange financial data described in section S4.5. Data for the experiment was prepared as follows: 1) Sample one minute negative log returns of 12 financial instruments. At each time t_i, we obtain a 12 element vector $\mathbf{r}(t_i) = \log \mathbf{p}(t_{i-1}) - \log \mathbf{p}(t_i)$, where $\mathbf{p}(t_i)$ is the vector of 12 instruments mid prices at time t_i. Each $\mathbf{r}(t_i)$ represents a vector of 1 minute losses. 2) For each t_i, we collect $\mathbf{y} = \mathbf{r}(t_i)_{10,11,12}$ and $\mathbf{x} = \mathbf{r}(t_i)_{1..9}, \mathbf{r}(t_{i-1})$. This composition of data encodes a 3 dimensional response variable representing 1 minute loss from the last 3 instruments at time t_i and covariates are 1 minute losses from the rest of the instruments at time t_i, combined together with the previous period t_{i-1} 1 minute losses from all the instruments (x is 21 dimensional vector). We train our estimators on such constructed data by maximizing the log-likelihood function.

Table 1: Mean Loglikelihoods - Large UCI Datasets.

	Power	Gas	Hepmass	Miniboone	Bsds300
MADE MoG	0.40 ± 0.01	8.47 ± 0.02	-15.15 ± 0.02	-12.27 ± 0.47	153.71 ± 0.28
MAF-affine (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	-11.75 ± 0.44	155.69 ± 0.28
MAF-affine (10)	0.24 ± 0.01	10.08 ± 0.02	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF-affine MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
TAN (various architectures)	0.48 ± 0.01	11.19 ± 0.02	-15.12 ± 0.02	-11.01 ± 0.48	157.03 ± 0.07
NAF	$\mathbf{0.62 \pm 0.01}$	11.96 ± 0.33	-15.09 ± 0.40	$\mathbf{-8.86 \pm 0.15}$	$\mathbf{157.73 \pm 0.04}$
B-NAF	0.61 ± 0.01	12.06 ± 0.09	$\mathbf{-14.71 \pm 0.38}$	-8.95 ± 0.07	157.36 ± 0.03
MONDE MADE	$\mathbf{0.62 \pm 0.01}$	$\mathbf{12.12 \pm 0.02}$	-15.83 ± 0.06	-10.7 ± 0.46	153.17 ± 0.29

We want to assess the models' ability to correctly rank tail events of any of the 3 assets experiencing loss at least in the 95 percentile of the historical loss in the next minute. To do this, we obtain the 95-th percentile threshold for each dimension of the $\mathbf{y}$ measured on the training set: $\mathbf{y}^{95}$. We compute the labels on the test partition as: $l = 1(\mathbf{y}_1 > \mathbf{y}_1^{95} \vee \mathbf{y}_2 > \mathbf{y}_2^{95} \vee \mathbf{y}_3 > \mathbf{y}_1^{95})$ i.e. the label is 1 whenever value at any of the dimensions is larger then its 95-th percentile, otherwise is 0. We compute the test score for the trained estimator by feeding it with test set covariates $\mathbf{x}$ and plugging in $\mathbf{y}^{95}$ as the response vector (the same response vector for each test covariate vector). The CDF output from the model is the estimate of the probability $P(\mathbf{Y} \leq \mathbf{y}^{95}|\mathbf{x}) = F(\mathbf{y}^{95}|\mathbf{x})$. We estimate the tail probability of the label being equal 1 i.e. $P(L = 1 \mid \mathbf{x}) = P(\mathbf{Y}_1 > \mathbf{y}_1^{95} \vee \mathbf{Y}_2 > \mathbf{y}_2^{95} \vee \mathbf{Y}_3 > \mathbf{y}_1^{95} \mid \mathbf{x}) = 1 - F(\mathbf{y}^{95}|\mathbf{x})$. Such computed ranks and the true labels are used to compute the Receiver Operating Characteristic curve, Area Under Curve score, Precision-Recall curve and Average Precision score on the test partition of the dataset. These performance measures are used to compare our estimators to a multilayer perceptron with sigmoid outputs, Random Forests and Gradient Boosting Trees (Chen and Guestrin, 2016). These discriminative methods are trained directly on labels predefined before training. Our estimators do not have to use a particular threshold at a test time. It can be changed after training is completed which is not possible for discriminative models[6].

ROC plots are shown in Figure 5. ROC curves for the XGBoost and Random Forest classifiers cluster at the lower level of TPR for small values of FPR. The other models have ROC curves placed slightly higher. We see that results for all models are similar, where the multilayer perceptron and PUMONDE models achieved slightly higher AUC score than the rest of the classifiers. PR plots are shown in Figure 5. PR curves and average precision score (labelled "Area" in the legend of the Figure) tell a similar story. In conclusion, we showed evidence that our method is competitive in this task against black-boxed models finely tuned to a particular choice of threshold, but where we can instantaneoulsy re-evaluate classifications by changing the decision threshold without retraining the model. This is not possible with the baseline models, which are also less interpretable as they do not show how the distribution of the original continuous measurements changes around the tails.

[6]This is analogous to a Bayesian Network providing the answer to any query, as opposed to a specialized predictor fit to answer a single predefined query.

4.3 TASK III: TAIL DEPENDENCE

In this experiment, we assess whether our models can be used to estimate a measure of extreme dependence between two random variables Y_i and Y_j, *tail dependence* (Joe, 1997):

$$\begin{aligned}\lambda_L(u) &= \lim_{u \to 0^+} P(Y_i \leq F_i^{-1}(u) | Y_j \leq F_j^{-1}(u)) \\ &= \lim_{u \to 0^+} \frac{P(Y_i \leq F_i^{-1}(u), Y_j \leq F_j^{-1}(u))}{P(Y_i \leq F_i^{-1}(u))}\end{aligned}$$

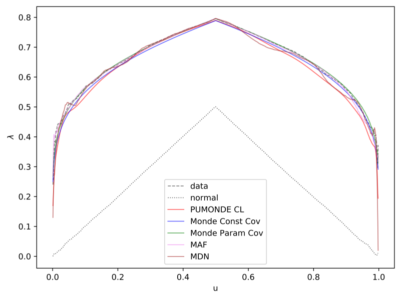
$$\begin{aligned}\lambda_R(u) &= \lim_{u \to 1^-} P(Y_i > F_i^{-1}(u) | Y_j > F_j^{-1}(u)) \\ &= \lim_{u \to 1^-} \frac{P(Y_i > F_i^{-1}(u), Y_j > F_j^{-1}(u))}{P(Y_i > F_i^{-1}(u))} \\ &= \lim_{u \to 1^-} \frac{1 - 2u + F_{ij}(F_i^{-1}(u), F_j^{-1}(u))}{1-u},\end{aligned}$$

where $\lambda_L(u)$ and $\lambda_R(u)$ are lower and upper tail dependence indices respectively, F_i is the marginal distribution function for random variable Y_i, F_{ij} is the bivariate marginal for random variables Y_i and Y_j. In our experiment we use conditional distributions so distributions depend on covariates: $F_i(y_i \mid \mathbf{x})$ and $F_{ij}(y_i, y_j \mid \mathbf{x})$.

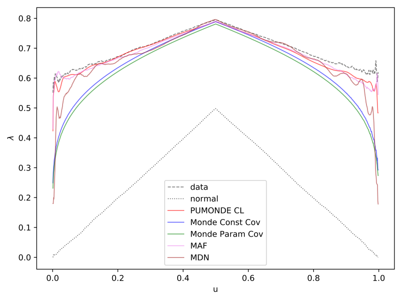
In order to have ground truth and provide some interpretability, we generate synthetic data as follows. We sample a Bernoulli random variable $C \in \{0, 1\}$ that indicates which of two components generates the covariates $\mathbf{X}$. The components are two Gaussian multivariate distributions with different means and identity covariance matrices. The choice of component also generates response variables $\mathbf{Y}$. In this case, the two distributions are such that the first is normally distributed (no tail dependence) and the second is t-distributed with 2 degrees of freedom. We repeat this process independently for each point in the dataset:

$$
\begin{aligned}
\mathbf{C} &\sim Bernouli(0.5) \\
\mathbf{X} &\sim \mathbf{X}_c \\
\mathbf{X}_0 &\sim N((-2,-3),\mathbf{I}) \\
\mathbf{X}_1 &\sim N((2,5),\mathbf{I}) \\
\mathbf{Y} &\sim \mathbf{Y}_c \\
\mathbf{Y}_0 &\sim N((0,0,0),\Sigma) \\
\mathbf{Y}_1 &\sim t((0,0,0),2,\Sigma)
\end{aligned}
\qquad
\begin{aligned}
\Sigma &= \sigma \mathrm{P} \sigma \\
\sigma &= \begin{bmatrix} 0.4 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.8 \end{bmatrix} \\
\mathrm{P} &= \begin{bmatrix} 1.0 & 0.8 & 0.1 \\ 0.8 & 1.0 & -0.5 \\ 0.1 & -0.5 & 1.0 \end{bmatrix}.
\end{aligned}
$$

To illustrate concentration in the tails of the distribution, we plot $\hat{\lambda}_L(u)$ for $u \in (0, 0.5)$ and $\hat{\lambda}_R(u)$ for $u \in (0.5, 1)$ in Figure 6. This includes models presented in this paper and also two baseline models (MAF and MDN). We describe the procedure used to compute these estimators in Section S4.6. We present two concentration plots. The first one is for the response variable generated from the multivariate Gaussians i.e. the first component, which does not exhibit tail dependence. This can be observed in the curves which tend to 0 when u gets closer to 0 and 1. The second one which depicts concentration plots for mixture component generated from the t-distributed sample with 2 degrees of freedom clearly present tail dependence which can be noticed by their limit converging to 0.6. We can see that Copula and MDN models fail to capture tail dependence (the first as expected, but the latter somehow has issues approximating non-Gaussian tails with a mixture of Gaussians). The PUMONDE model trained with composite likelihood (as described in Section 2.4.1) and MAF model are able to capture "fatter" tails in the data.



(a) Gaussian Component



(b) T Component

Figure 6: Tail Dependence Concentration Plots (better seen in colour). The triangle shaped curve in each plot is the tail dependence concentration plot for isotropic Gaussian (shown as comparison for curves depicting dependence and larger kurtosis). The first plot shows that all models correctly capture the lack of tail dependence in Gaussian distribution. The second plot shows that only PUMONDE and MAF concentration plots are close to the data concentration plot in the tails (when u tends to 0 or 1)

4.4 TASK IV: MUTUAL INFORMATION

Table 2: Mutual Information.

	Gaussian Component			T Component		
Model	$I(Y_0,Y_1)$	$I(Y_0,Y_2)$	$I(Y_1,Y_2)$	$I(Y_0,Y_1)$	$I(Y_0,Y_2)$	$I(Y_1,Y_2)$
Data	0.5108	0.0057	0.1454	0.5108	0.0057	0.1454
MAF	**0.5107** (**0.0001**)	0.0018 (−0.004)	0.1827 (0.0373)	0.5786 (0.0677)	0.0831 (0.0774)	0.199 (0.0536)
MDN	0.5172 (0.0064)	0.0359 (0.0301)	0.1718 (0.0264)	0.6112 (0.1004)	0.1143 (0.1085)	0.2356 (0.0901)
MONDE Const	0.4826 (−0.0283)	0.0078 (0.0021)	**0.1304** (−**0.015**)	0.5363 (0.0255)	**0.0414** (**0.0357**)	**0.1431** (−**0.0024**)
MONDE Param	0.5078 (−0.003)	0.0796 (0.0738)	0.1303 (−0.0151)	0.438 (−0.0728)	0.0849 (0.0792)	0.1268 (−0.0186)
PUMONDE	0.4682 (−0.0425)	**0.004** (−**0.0017**)	0.1105 (−0.0349)	**0.5307** (**0.0199**)	0.0573 (0.0515)	0.1621 (0.0167)

Pairwise mutual information measures how much information is shared between two random variables. It captures not only linear dependency, but also more complex relations. It is defined as KL-divergence between the joint bivariate marginal distribution and the product of the corresponding univariate marginals. We now show results concerning estimation of pairwise mutual information. The data generating process is the same as in Section 4.3. We compute mutual information by marginalizing distributions provided by the models using numerical quadrature. We can apply this method because of the low dimensionality of the problem.

Mutual information was computed for each pair of variables for the data generating process, two baseline models (MDN and MAF) and two of our models (Gaussian copula MONDE, PUMONDE). We compute mutual information for each mixture component separately by conditioning each model on the covariate equal to the mean vector for the given covariate mixture component. The results are presented in Table 2. For each combination of model/pair of response variables/mixture component, we obtain two values: mutual information score and the absolute difference between mutual information value for the model and value for the data generating process (the difference is shown in brackets). The smallest absolute value of the difference in a given column is highlighted in bold, indicating which model represents the closest mutual information to the one computed from the data generating process. MONDE models are better in five out of six cases. We can conclude that models presented in this paper are competitive in encoding bivariate dependency with the current state of the art methods. Having this evidence leads us to our final Task, where we exploit estimating simultaneously multiple marginals of a common joint. CDF parameterizations are particularly attractive, as marginalization takes the same time as evaluating the joint model (Joe, 1997), unlike some of the methods discussed in this section.

4.5 TASK V: BIVARIATE LIKELIHOOD

Table 3: Bivariate Likelihood Model Comparison.

	MDN	MONDE Const	MONDE Param	PUMONDE
MDN	NA	0	0	0
MONDE Const	210	NA	27	0
MONDE Param	210	183	NA	0
PUMONDE	210	210	210	NA

In many practical problems, we are interested in estimating only particular marginals. Parameters for higher order interactions are considered to be nuisance parameters. Allowing for partial likelihood specification is one of the primary motivations behind composite likelihood (Varin et al., 2011)[7].

The problem with partial specification is that in general there are no guarantees that the corresponding marginals come from any possible joint distribution. On the other hand, a fully specified likelihood has nuisance parameters. Ideally, we would like a flexible, overparameterized joint model so that parameters are not obviously responsible for any marginals a priori, with the objective function regularizing them towards the marginals of interest. PUMONDE provides such an alternative. Although high dimensional likelihoods are intractable to compute in PUMONDE, low dimensional marginals are not.

In this section, we test the ability of our models to encode coherent bivariate dependence in the data for problems of larger dimensionality. For example, in finance this can be useful to model second order dependence of returns in the portfolio of instruments as used in computation of the Value at Risk metric (Holton, 2003). We use foreign exchange financial data as described in section S4.5. This data contains a 21 dimensional response variable representing 1 minute losses for financial instruments at time t_i and a 21 dimensional covariate variable representing 1 minute losses for all the instruments at time t_{i-1}. We train the estimators on such constructed data maximizing the likelihood objective. For PUMONDE, we optimize the composite likelihood objective comprised of the sum of all combinations of bivariate likelihoods. The only neural density estimator we use is MDN, fit to the 21 dimensional distribution. Marginalization in MDN is easy as it encodes a mixture of Gaussians, while the other baseline models cannot be easily marginalized.

To assess model performance, we compute the average log-likelihood for each bivariate combination of response variables on the test partition, giving 210 unique pairs. Each cell of Table 3 contains the number of times the average log-likelihood computed for each bivariate combination of response variable was larger for model shown in the row compared to the model which is shown in the column. We can see that the best performing model on this test is the PUMONDE which obtained larger test log likelihoods in all 210 cases when compared to each other model. MONDE with parametrized covariance achieved better results than MDN and MONDE with constant covariance. The worst results were obtained by MDN.

5 DISCUSSION

We proposed a new family of methods for representing probability distributions based on deep networks. Our method stands out from other neural probability estimators by encoding directly the CDF. This complements other methods for problems where the CDF representation is particularly helpful, such as computing tail area probabilities and computing small dimensional marginals. As future work, we will exploit its relationship to graphical models for CDFs (Huang and Frey, 2008; Silva et al., 2011), using PUMONDE to parameterize small dimensional factors. Variations on soft-recursive partitioning, such as hierarchical mixture of experts (Jordan and Jacobs, 1994), can also be implemented using tail events to define the partitioning criteria. Another interesting and less straightforward venue of future research is to exploit approximations to the likelihood based on the link between differentiation, latent variable models and message passing, as exploited in the context of graphical CDF models (Silva, 2015; Huang et al., 2010) and automated differentiation (Minka, 2019).

[7]This is not to be confused with another motivation, which is to provide a tractable replacement for the likelihood function. In this case, a full likelihood is still specified and of interest. While computational tractability is a more common motivation in machine learning, partial specification is one of the main reasons for the development of composite likelihood in the statistics literature.

References

C. Bishop. Mixture density networks. Technical report, NCRG 4288, Aston University, Birmingham, 1994.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, 2016.

N. De Cao and I. Titov. Block neural autoregressive flow. In *UAI*, 2019.

L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear independent components estimation. In *ICLR*, 2015.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. In *ICLR*, 2017.

G. Elidan. Copulas in machine learning. In *Copulae in Mathematical and Quantitative Finance*. Springer, Berlin, 2013.

M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. In *ICML*, 2015.

G.A. Holton. *Value-at-risk: Theory and Practice*. Academic Press Inc, 2003.

C-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural autoregressive flows. In *ICML*, 2018.

J. Huang and B. Frey. Cumulative distribution networks and the derivative-sum-product algorithm. In *UAI*, 2008.

J. Huang, N. Jojic, and C. Meek. Exact inference and learning for cumulative distribution func- tions on loopy graphs. In *NIPS*, 2010.

H. Joe. *Multivariate models and dependence concepts*. Monographs on statistics and applied probability. CRC Press, 1997.

M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

T. Minka. From automatic differentiation to message passing. *Invited talk at the Advances and challenges in Machine Learning Languages workshop (ACMLL 2019)*, 2019. URL https://tminka.github.io/papers/acmll2019/.

J. Oliva, A. Dubey, M. Zaheer, B. Póczos, R. Salakhutdinov, E. Xing, and J. Schneider. Transformation autoregressive networks. In *ICML*, 2018.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *NIPS*, 2017.

T. Schmidt. Coping with copulas. In *Copulas – From Theory to Applications in Finance*, pages 3–34. Risk Books, 2006.

R. Silva. Bayesian inference in cumulative distribution fields. *Interdisciplinary Bayesian Statistics*, pages 83–95, 2015.

R. Silva, C. Blundell, and Y-W. Teh. Mixed cumulative distribution networks. In *AISTATS*, 2011.

A. Sklar. Fonctions de répartition à n dimensions et leurs marges. *Publications de l'Institut de Statistique de l'Université de Paris*, 8, 1959.

B. Uria, I. Murray, and H. Larochelle. RNADE: the real-valued neural autoregressive density-estimator. In *NIPS*, 2013.

B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In *ICML*, 2014.

C. Varin, N. Reid, and D. Firth. An overview of composite likelihood methods. *Statistica Sinica*, 21(1), 2011.