Supplementary Material: 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

S1 RELATED WORK

In this section, we give an overview of the ideas that inspired our work and constitute the fundamental building blocks of our method.

S1.1 MONOTONIC NEURAL NETWORKS FOR FUNCTION APPROXIMATION

The first approach to model monotonic functions with neural networks added a penalty term to the learning objective function (Sill and Abu-Mostafa, 1996). No hard constraints were imposed, which for our purposes would mean negative density functions possibly appearing during training and testing. Sill (1997) proposed a model that encodes a hard monotonicity constraint, which was deemed necessary to make learning efficient for monotone regression functions. The inputs were first transformed linearly into disjoint groups of hidden units, by using constrained weights which were positive for increasing monotonicity and negative for decreasing monotonicity (weight constraints were enforced by exponentiating each free parameter). The groups were processed by a "max" operator and a "min" operator. The max operator modelled the convex part of the monotone function and the min operator modelled the concave part. The whole model could be learned by gradient descent. Authors proved that their model could approximate any continuous monotone function with finite first partial derivative to a desired accuracy. This model had several hyper-parameters, including the number of groups or number of hyper-planes within the group.

Lang (2005) proved that if the output and hidden-to-hidden weights were positive for a single-layer network, then they could constrain input-to-hidden layers weights selectively to be positive/negative depending on monotonicity constraints on selected input variables. In this way, they could choose for which input variables they want to preserve monotonicity of the output. They also observed that min/max networks (Sill, 1997) tended to be more expensive to train than this simple neural network architecture with constrained weights.

The review by Minin et al. (2010) tested several approaches on a variety of datasets which exhibited monotonicity on some of the inputs. Authors concluded that there was no definite winner and the evaluated approaches excel in different areas and applications.

S1.2 NEURAL DENSITY ESTIMATION

One of the first methods to build conditional density estimators using neural networks was the Mixture Density Networks (MDNs) of Bishop (1994). The main motivation behind this work was the inability of standard regression models to summarize multimodal outputs with conditional means. MDNs parameterize conditional mixtures of Gaussians where the neural network outputs mixing probabilities and Gaussian parameters for each mixture.

Another approach, presented by Wang (1994), trained a density estimator by fitting a monotonic neural network to match a smoothed empirical CDF. Unlike our work, this was solely a smoother that reconstructed the empirical CDF using a function approximator. There was no likelihood function or supervised signal.

The method presented by Likas (2001) tackles the problem of normalizing the output of a neural network to directly approximate density functions. It uses numerical integration over the domain of the function.

Two methods were presented by Magdon-Ismail et al. (2002) for unconditional density estimation using neural networks and CDFs. They both rely on the empirical CDF as targets to be approximated, with no explicit likelihood function being used during training. The reliance on the empirical CDF to provide training signal also implies that there is no straightforward way of adapting it to the conditional density estimation problem.

An approach based on the autoregressive factorized representation of the density function was presented by Larochelle and Murray (2011) and Uria et al. (2013). In the continuous case, the proposed model parameterizes each conditional marginal using what is essentially a MDN (Bishop, 1994). All neural networks that parameterize the mixtures partially share parameters, in a way that also speeds up computation as the number of parameters will grow only linearly with dimensionality. The model was extended in Uria et al. (2014) to use deep neural networks to parameterize an ensemble of variable orderings.

Trentin (2016) observed that using a neural network to approximate the CDF function, and then differentiating it to obtain a density function estimate, can give poor results. This does not affect MONDE, as our objective function maximizes the likelihood function instead of directly approximating a measure of distance to the empirical CDF as done by, e.g., Magdon-Ismail et al. (2002). His proposal included a model of the density function that must be normalized numerically. Zhang (2018) improved on the work of Magdon-Ismail et al. (2002), which comes from using hard monotonicity constraints instead of the penalization approach used by the other authors.

Dinh et al. (2015) uses a transformation of density formula. This method applies an invertible transformation, with an easily computable determinant of the Jacobian, to map data with complex dependencies into simple factorized parametric distributions. The MONDE models have the advantage of providing a directly computable CDF, with the disadvantage of not providing a straightforward way to sample from the learned distribution. This estimator was extended by Dinh et al. (2017); Papamakarios et al. (2017); Huang et al. (2018); De Cao and Titov (2019) and others, mainly by using more complex transformations with tractable Jacobians.

A model that uses efficient parameter sharing to encode autoregressive dependency structure was applied in Germain et al. (2015). Authors modified the autoencoder using autoregressive transformations so the output can represent a valid density function. The model outputs a binary density by using logistic outputs or mixtures of parametric models for real valued data. The ideas presented in this work were used later in Papamakarios et al. (2017). We use similar parametrization in the autoregressive MONDE model described in section 2.2 with an additional constraint on a subset of parameters that have to be non-negative, so the output of the estimator represents valid conditional CDFs.

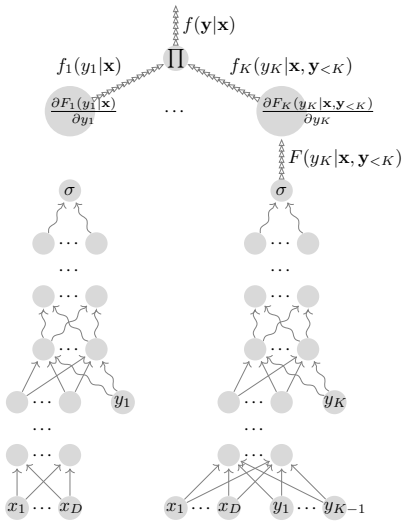


Figure S1: Autoregressive MONDE, The Multivariate Monotonic Neural Density Estimator architecture.

S2 THE MONOTONIC NEURAL DENSITY ESTIMATOR

S2.1 AUTOREGRESSIVE MONDE

S2.1.1 MADE Implementation Details

We now present implementation details about the model described in the previous section. Originally, the autoregressive model has been implemented as shown in Figure S1. But this approach has not scaled well to a larger number of dimensions because each component had its own independent computational graph. To speed up the training and decrease the number of parameters, we applied the ideas of Germain et al. (2015), resulting in the model presented in Figure 2. Each layer l has its own parameter matrix that is constrained in a way such the computational graph can represent valid conditional CDFs for each dimension, using autoregressive representation. For example, covariates $\mathbf{x}$ and response variables $\mathbf{y}$ are transformed into the first hidden layer according to the formula

$$\sigma(\begin{bmatrix} x_1 & \dots & x_D & y_1^0 & \dots & y_K^0 \end{bmatrix} \times \mathbf{W}^1) = \begin{bmatrix} y_{1,1}^1 & y_{1,2}^1 & \dots & y_{1,K}^1 & y_{2,1}^1 & \dots & y_{M,K}^1 \end{bmatrix},$$

where $\sigma(\cdot)$ is the sigmoid function (with range of $(0, 1)$) applied elementwise on the vector input; $\mathbf{W}^1$ is a constrained matrix of size $D + K \times KM$, where D is the number of dimensions of the covariate vector, K is the dimension of the response vector and M is the number of K-element vectors in the hidden layer; and matrix $\mathbf{W}^1$ has following structure,

$$\begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \dots & w^1_{1,K} & \dots & w^1_{1,KM} \\ w^1_{2,1} & w^1_{2,2} & \dots & w^1_{2,K} & \dots & w^1_{2,KM} \\ \dots & & & & & \\ w^1_{D,1} & w^1_{D,2} & \dots & w^1_{D,K} & \dots & w^1_{D,KM} \\ w^1_{D+1,1} & w^1_{D+1,2} & \dots & w^1_{D+1,K} & \dots & w^1_{D+1,KM} \\ \dots & & & & & \\ w^1_{D+K,1} & w^1_{D+K,2} & \dots & w^1_{D+K,K} & \dots & w^1_{D+K,KM} \end{bmatrix},$$

where

- $w^1_{i,.} \in \mathbb{R}$ for $i \in [1, D]$,
- $w^1_{D+k,k+mK} \in \mathbb{R}^+ \cup \{0\}$ for $k \in [1, K]$, $m \in [0, M-1]$,
- $w^1_{D+k_1,k_2+mK} \in \mathbb{R}$ for $k_1 < k_2$, $k_{1,2} \in [1, K]$, $m \in [0, M-1]$,
- $w^1_{D+k_1,k_2+mK} = 0$ for $k_1 > k_2$, $k_{1,2} \in [1, K]$, $m \in [0, M-1]$.

Non negative parameters are obtained by squaring the respective free parameters of the transformation matrix. We constrain selected parameters to be zero by multiplying the parameter matrix elementwise by a mask matrix containing 0 at locations which should be zeroed and 1 otherwise, as in the original MADE implementation. Each $l-1$-th hidden layer is then transformed into the next l-th hidden layer as follows:

$$\sigma(\begin{bmatrix} y^{l-1}_{1,1} & y^{l-1}_{1,2} & \dots & y^{l-1}_{1,K} & y^{l-1}_{2,1} & \dots & y^{l-1}_{M,K} \end{bmatrix} \times \mathbf{W}^l) = \begin{bmatrix} y^l_{1,1} & y^l_{1,2} & \dots & y^l_{1,K} & y^l_{2,1} & \dots & y^l_{M,K} \end{bmatrix},$$

where $\mathbf{W^1}$ is a constrained matrix of size $KM \times KM$,

$$\begin{bmatrix} w^l_{1,1} & w^l_{1,2} & \dots & w^l_{1,K} & \dots & w^l_{1,KM} \\ w^l_{2,1} & w^l_{2,2} & \dots & w^l_{2,K} & \dots & w^l_{2,KM} \\ \dots & & & & & \\ w^l_{KM,1} & w^l_{KM,2} & \dots & w^l_{KM,K} & \dots & w^l_{KM,KM} \end{bmatrix},$$

such that:

- $w^l_{k+m_1K,k+m_2K} \in \mathbb{R}^+ \cup \{0\}$ for $k \in [1, K]$, $m_{1,2} \in [0, M-1]$,
- $w^l_{k_1+m_1K,k_2+m_2K} \in \mathbb{R}$ for $k_1 < k_2$, $k_{1,2} \in [1, K]$, $m_{1,2} \in [0, M-1]$,
- $w^l_{k_1+m_1K,k_2+m_2K} = 0$ for $k_1 > k_2$, $k_{1,2} \in [1, K]$, $m_{1,2} \in [0, M-1]$.

The L-th layer outputs a K-dimensional vector in which each component represents the conditional CDF:

$$\sigma(\begin{bmatrix} y^{L-1}_{1,1} & y^{L-1}_{1,2} & \dots & y^{L-1}_{1,K} & y^{L-1}_{2,1} & \dots & y^{L-1}_{M,K} \end{bmatrix} \times \mathbf{W}^L) = \begin{bmatrix} y^L_{1,1} & y^L_{1,2} & \dots & y^L_{1,K} \end{bmatrix},$$

where $\mathbf{W}^L$ is constructed in similar way as hidden to hidden parameter matrices. The presented composition of layers constrains the $\mathbf{y}^L$ output to fulfil requirements that has to be met by valid autoregressive representation of the joint CDFs i.e.

- $y^L_1 = t_w(y^+_1) = F_1(y_1)$,
- $y^L_2 = t_w(y_1, y^+_2) = F_2(y_2|y_1)$,
- $\dots$
- $y^L_K = t_w(\mathbf{y}_{<k}, y^+_K) = F_1(y_k|\mathbf{y}_{<k})$.

Here, t_w represents the final output of the computational graph of the model as parameterized by $w = \mathbf{W}^1, ..., \mathbf{W}^L$. By construction, $t_w(\mathbf{y}_{<k}, y^+_k)$ is nondecreasing monotonic on input y_k, and unconstrained on inputs $y_1 \dots y_{k-1}$. Having obtained a parameterization that computes valid conditional CDFs, we can construct the density estimator by computing the product of the derivatives of conditional CDFs with respect to their respective target variables,

$$f(\mathbf{y}|\mathbf{x}) = f(y_1|\mathbf{x}) f(y_2|\mathbf{x}, y_1) \dots f(y_k|\mathbf{x}, \mathbf{y}_{<k}) = \frac{\partial F_1(y_1|\mathbf{x})}{\partial y_1} \frac{\partial F_2(y_2|\mathbf{x}, y_1)}{\partial y_2} \dots \frac{\partial F_K(y_K|\mathbf{x}, \mathbf{y}_{<K})}{\partial y_K}.$$

We optimize parameters w by maximizing expected log-likelihood using a Monte Carlo estimate i.e. we maximize the average log-likelihood over the batch of data points sampled randomly from the training dataset.

S2.1.2 Finer Remarks

In this section, we give further implementation details for the autoregressive model.

- We modified the MONDE approach to constrain the parameters matrices of Germain et al. (2015) by introducing non negative weights. This requirement is identical to the one used in univariate MONDE described in Section 2.1.
- Compared to MADE, we have the size of each hidden layer being equal to a multiple of the response vector size. In this way, each layer propagates the same number of autoregressive blocks of vectors

that are used at the top layer to construct CDFs. This would be very inefficient for very high dimensional data. We have also implemented a version of the estimator which at each hidden layer we sample the nodes like it is done in MADE.

- To stabilize the learning in the final stages of the training, we increase the batch size by a factor of 2 whenever we experience numerical problems during training (i.e. we use the last good parameters before gradient computation or network evaluation resulted in numerical problems, and restart training with doubled batch size). This method to increase batch size is inspired by Smith et al. (2018). We found this to be a very important procedure used during training so we could achieve results comparable to the ones shown by Huang et al. (2018). We also increase the batch size after the performance of the estimator on the validation set does not improve on 10 consecutive training epochs. It remains to be checked whether introducing techniques like batch or activation normalization would render this approach unnecessary.

- We found that using a scaled and translated $\tanh$ to the range of $(0, 1)$ in all layers (hidden and final) helps to stabilize the learning process. We found empirically that using it causes fewer numerical problems than using directly the sigmoid function. There is at least one possible explanation of this phenomenon: $\tanh$ has larger gradients than a regular sigmoid but we have not verified it theoretically why it helped with optimization. It was already tested empirically that using $\tanh$ instead of sigmoid can be beneficial for the final result (for example, Liew et al., 2016). We tried using different non linearities in the hidden layer transformations such as softsign, softplus, ReLU and sigmoid but modified $\tanh$ led to the best results. We are aware that the initialization of the parameters can also affect the performance of the model but we have not performed any extensive study to choose the best approach to parameter initialization and we leave it to future work.

- We tried to modify batch normalization (Ioffe and Szegedy, 2015) which seems to be used in most recent neural density estimators. After trying it with many datasets we concluded that it sped up the rate of convergence but we obtained worse results in all cases compared with MONDE variations not using it. We think it requires further research and theoretical insight how to modify the batch normalization to help with training the autoregressive MONDE estimator.

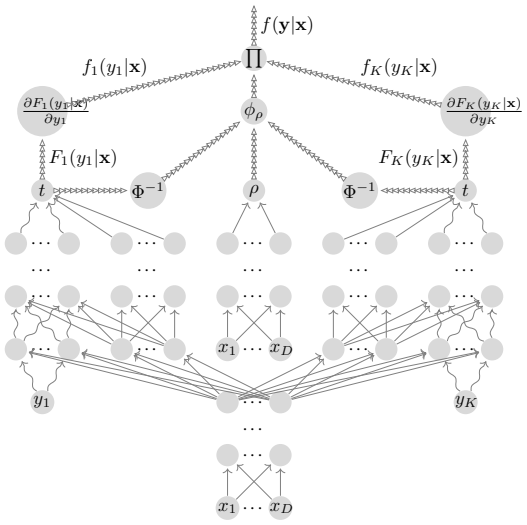


Figure S2: Multivariate Monotonic Neural Density Estimator with Gaussian Copula Dependency and Parameterized Covariance.

- We also trained a version of the model that used mixture of CDFs at the output of each autoregressive component. It is just an analogue to the Mixture Density Networks, where the base distributions are MONDE models and convex weights are given by a NN. This was achieved by the last layer being constructed from the softmax scaled parameter per component so we could compute mixture of multiple CDF outputs. We found that by using this extension we did not obtain statistically significantly better result so we decided not to include them in the experiments section.

S2.2 MULTIVARIATE COPULA MODELS

S2.2.1 MONDE Parameterized Covariance

Figure S2 contains a diagram of the MONDE Copula model with parameterized covariance. It differs from the MONDE Copula Constant Covariance only by an additional neural network which encodes the covariance of the output vector. The neural network maps covariate vector $\mathbf{x}$ into vectors $\mathbf{u}(\mathbf{x})$ and $\mathbf{d}(\mathbf{x})$ which are then used to construct a correlation matrix as shown in Equation 5.

S2.2.2 Details

When we were experimenting with MONDE Copula models, we tried different approaches which were not used in the final implementation of our algorithms:

- We tried to parameterize the correlation matrix as proposed by Rapisarda et al. (2007). However, we

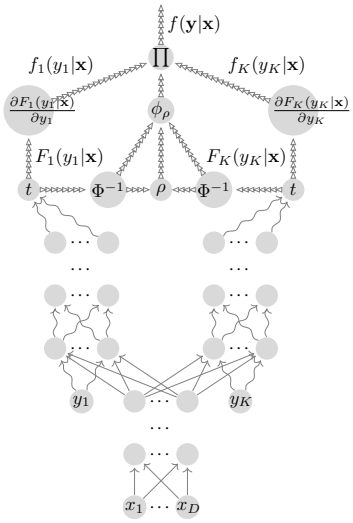


Figure S3: Multivariate Monotonic Neural Density Estimator with Gaussian Copula Dependency and Constant Covariance, model without two partitions for each marginal distribution.

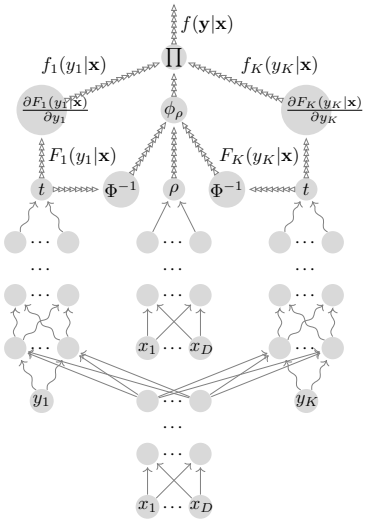


Figure S4: Multivariate Monotonic Neural Density Estimator with Gaussian Copula Dependency and Parameterized Covariance, model without two partitions for each marginal distribution.

obtained inferior results compared to the method presented in Equation 5.

- We tried to pre-train the models with first fitting each univariate marginal of the model to the unconditional empirical distributions as given by the data using the mean squared error objective, followed by maximizing likelihood objective given by the entire copula model. This method also did not bring any improvements in convergence speed nor better final results. To keep things simple, we drop it from the final implementation.
- Initially, our multivariate copula models did not have two vertical partitions for each marginal distribution (the part of the computational graph that computes $t_i(y_i, \mathbf{x})$). The initial models without two partitions are shown in Figure S3 and S4 for constant and parametrized covariance respectively. The final models are presented in Figure 3 and Figure S2. This extension is constructed in a way so that the monotonicity of t_i with respect to y_i is not destroyed i.e. we allow connections with unconstrained weights from the covariate partition into the partition processing the response variable but not the other way around. These additional connections in each of the t_i transformations allowed us to obtain better results.

S2.3 PUMONDE

S2.3.1 Architecture Justification

In this section, we show why we chose the proposed computational graph for PUMONDE model. For simplicity of exposition, we focus on the bivariate case, but the explanation applies to the general multivariate case.

To obtain a valid distribution function $F(y_1, y_2|\mathbf{x})$ and the corresponding density function $f(y_1, y_2|\mathbf{x})$, we need to constrain our neural network to meet following conditions:

1. $\lim_{y_1 \to -\infty, y_2 \to -\infty} F(y_1, y_2|x) = 0$
2. $\lim_{y_1 \to +\infty, y_2 \to +\infty} F(y_1, y_2|x) = 1$
3. $\frac{\partial^2 F}{\partial y_1 \partial y_2} \in \mathbb{R}^+$
4. $\frac{\partial^2 F}{\partial y_k^2} \in \mathbb{R}$ for $k = 1, 2$

To meet constraints 1 and 2, we parameterize the distribution function as the ratio of two non-negative monotonic functions,

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

We chose all the transforms of $t(\cdot)$ to be the softplus function, which we will denote with the symbol s_+ for the rest of this section. There are numerous reasons for this choice. First, s_+ meets the requirement for the output to be non-negative and monotonically increasing with respect to its input. It is required by the monotonicity property of the computational graph to use monotonic transformations for all computations transforming response variables. We also used s_+ and not, for example, $\tanh$ because the second order derivative of s_+ with respect to its input is positive (the softplus function is convex throughout all of its domain). The second order derivative of $\tanh$ with respect to its input can be negative, which breaks constraint 3. The multiplication layer (here symbolically written as $m(\cdot)$), which can be seen in the middle of the computation graph, receives positive valued transformations of y_1 and y_2 ($h_{xyi}(\cdot)$ uses only sigmoid non-linearities). This composition allows us to fulfil constraint 3. This comes from the fact that

$$\frac{\partial^2 s_+(\sigma(g_1(y_1)) \times \sigma(g_2(y_2)))}{\partial y_1 \partial y_2} =$$
$$\frac{\partial s_+^{'}(\cdot) \times \sigma(g_1(y_1)) \times \sigma^{'}(g_2(y_2)) \times g_2^{'}(y_2)}{\partial y_1} =$$
$$[s_+^{''}(\cdot) \times \sigma(g_2(y_2)) \times \sigma(g_1(y_1)) + s_+^{'}(\cdot)] \times$$
$$\sigma^{'}(g_1(y_1)) \times g_1^{'}(y_1) \times \sigma^{'}(g_2(y_2)) \times g_2^{'}(y_2),$$

where $\sigma(g_i(\cdot)) = h_{xyi}(\cdot)$ and g_i is the monotonic transformation with respect of y_i using only $\mathrm{sigmoids}$ which contains all but the last transformation of $h_{xyi}(\cdot)$. Symbol $s_+^{'}(\cdot)$ denotes the derivative of the function with respect to its input from the previous expression in the equation.

Therefore, that last expression always evaluates to a non-negative number. We see that, if we used $\mathrm{sigmoid}$ or $\tanh$ in the layers following the multiplication layer ($m(\cdot)$), we would not be guaranteed to obtain a non-negative result. Using the sigmoid activation function before the multiplication layer still allows to meet constraint 4 because the second derivative of σ with respect to its input can be positive and negative (because this function is convex and concave depending on the input). It is also imperative that the inputs into the multiplication layer (outputs of the $h_{xyi}(\cdot)$s transformations) of PUMONDE is non-negative, because otherwise it would break the monotonicity property of the model with respect to response variables.

S2.3.2 Implementation Remarks

We found that using the vanilla softplus activation can lead to numerical problems in models which have many layers (we found that the minimum number of layers at which the computation resulted in numerical problems also is dependent on dataset itself). After running several experiments, we concluded that the most numerically stable approximation of the softplus is $\log(1 + \exp(-|x|)) + \max(x, 0)$.

S2.4 COMPUTATIONAL COMPLEXITY

Applying auto differentiation twice on the univariate response MONDE (with respect to inputs and then with respect to parameters) is more costly by a constant factor than applying it once in the model parameterizing the pdf directly. The training of the multivariate estimators like PUMONDE mirrors the problem with the exact inference in dense Markov Networks i.e. it scales exponentially in the dimensionality of the problem. We mitigate this issue by exploiting structure in the Gaussian copula and by the use of composite likelihood. In follow-up work, we want to adapt our method to structured CDFs which were introduced in Huang and Frey (2008). PUMONDE can also motivate future message-passing views of autodiff Minka (2019).

S3 BASELINE MODELS IMPLEMENTATION

We found that the implementation of models using mixture components like RNADE and MDN requires a few tricks to make them obtain good results. First, we needed to tweak the minimum value allowed for the scale parameters of the Gaussian components. If we allow it to be arbitrarily small, the models frequently put a lot of weight on one mixture component which has very high precision, particularly on finite-resolution data in which repeated values occur to some extent. This artificially inflated the average log-likelihood, sometimes by a large amount. This is a well known problem with mixture models. We checked that to achieve the best performance of such models, the minimum allowed value of the mixture weight would have to be adjusted for each dataset separately. In practice, we set this threshold to be the smallest one not causing the optimization process to misbehave, taking into consideration the already large dimension of the hyperparameter space we optimize over.

S4 EXPERIMENTS

S4.1 EXPERIMENTS SETUP

We split each dataset into train, validation and test partitions. Models are only trained on the train partition. Hyperparameters are chosen by selecting the best model with respect to the log-likelihood computed on the vali-

Table 1: Datasets dimensions.

DATA SET	OBSERVATIONS	X DIM	Y DIM
Sin Normal	10000	1	1
Sin T	10000	1	1
Inv Sin Normal	10000	1	1
Inv Sin T	10000	1	1
UCI Redwine 2D	1599	7	2
UCI Redwine Unsupervised	1599	0	9
UCI Whitewine 2D	4898	1	2
UCI Parkinsons 2D	5875	13	2
MV Nonlinear	10000	1	2
UCI Whitewine Unsupervised	4898	0	3
UCI Parkinsons Unsupervised	5875	0	15
ETF 1D	1073	1	1
ETF 2D	1073	2	2
FX All Assets Predicted	28781	16	8
FX EUR And GBP Predicted	28773	32	2
FX EUR Predicted	28749	80	1
Classification (FX)	91910	21	3
UCI Power	2049280	0	6
UCI Gas	1052065	0	8
UCI Hepmass	525123	0	21
UCI Miniboone	36488	0	43
UCI Bsds300	1300000	0	63
Mixture Process	100000	2	3
FX (Bivariate Likelihood)	91549	21	21

dation set. The search is done via exhaustive search over a predefined grid of hyperparameters. Table 5 shows the search space that was used for different models and experiments/datasets. We use the early stopping technique on the validation set to prevent overfitting with patience of 30 epochs i.e. we stop training when the likelihood on validation dataset does not improve for 30 consecutive epochs. The best model on the validation set is chosen and log-likelihoods of test points are computed. We compare the models performance by running pairwise t-tests on these values or use other performance metric adequate for a given experiment. We use the ADAM (Diederik and Jimmy, 2015) version of stochastic gradient descent to optimize the models. We tried different parameters of the optimizer but we settled with default ones that are used in the Tensorflow i.e. beta_1=0.9, beta_2=0.999, epsilon=$1e-07$. The learning rate and the batch size were set as given in Table 5. In some of the experiments, we used a modified learning process where we increase the batch size with a pre-defined schedule. We give the rationale behind using it in Section S2.1.2.

S4.2 EXPERIMENTS

All dimensions of the datasets used in our experiments are provided in Table 1. Experiments described in Sections S4.3, S4.4 and S4.5 have datasets arranged in a way to test the model performance on the following learning tasks: 1) unconditional/unsupervised: $dim(\mathbf{x}) = 0$ and $dim(\mathbf{y}) > 0$, 2) univariate: $dim(\mathbf{x}) > 0$ and $dim(\mathbf{y}) = 1$, 3) small dimensional multivariate: $dim(\mathbf{x}) > 0$ and $dim(\mathbf{y}) > 1$ probability density functions. In other experiments the dimensionality of the response and covariate vectors depends on the task performed.

S4.3 SYNTHETIC DATA

Results from experiments with synthetic data are shown in Table 2. The table displays the average log-likelihood computed on the test partition of the dataset. In each row, we highlight the best result for the model which achieved significantly better average log-likelihood than the rest of the models. We use three datasets in this experiment. The univariate covariate is generated unconditionally from the uniform $[0, 1]$ distribution for all three datasets. Then, depending on the dataset, the response variables are sampled from the Gaussian or Student-t distributions with means parameterized by a sinusoid function with an input that is a linear transformation of the covariate:

$$
\begin{aligned}
&X \sim Uniform(-1.5, 1.5) \\
&Y \sim N(\sin(4X) + 0.5X, 0.2) \text{ or} \\
&Y \sim t(3, \sin(4X) + 0.5X, 0.2).
\end{aligned}
$$

We also test the models by swapping the response variable with the covariate variable, so that we can check whether they can encode the multimodality in Y. Those inverted datasets have "Inv" added to their names. We show scatter plots of these datasets in Figure S5.

The next dataset has a bivariate response variable distributed according to the Gaussian distribution. The mean of each dimension is parameterized by a different nonlinear function with respect to the covariate. This bivariate response variable has correlated dimensions, each of them having a different variance:

$$
\begin{aligned}
&X \sim Uniform(-10, 10) \\
&\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 3 \end{bmatrix} \\
&Y \sim N([0.1 * \sqrt{X} + X - 5, 10sin(3X)], \Sigma).
\end{aligned}
$$

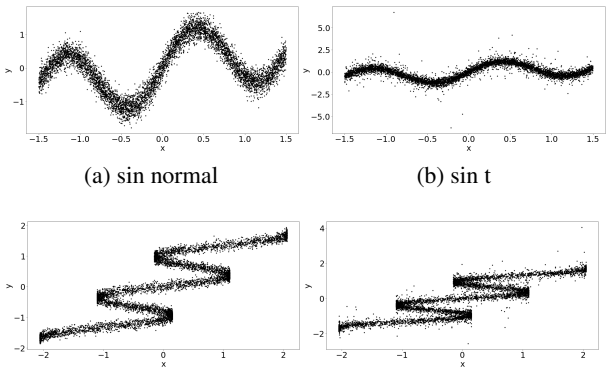
We present data generated from this random process in Figure S6 in a grid, with pairwise scatter plots off the diagonal and marginal densities on the diagonal cells.

Fitting the data by conditional mean models, such as a regular neural network with mean squared error, is not suitable in the case when we deal with multimodal output. In this case, we should be using models that can encode the probabilistic structure of the data. We show in Figure S7 that proposed and baseline models can capture the multimodality correctly.

In Figure S8, we show density heatmaps computed from different density estimators and data generating processes. In these plots, we show density $f(y_0, y_1 = 0 \mid x)$ of baseline and proposed models for a grid of points spanned over y_0 and x. We see that RNADE and

Table 2: Synthetic Data - Mean Loglikelihood.

	RNADE Laplace	RNADE Normal	RNADE Deep Normal	RNADE Deep Laplace	MONDE Const Cov	MONDE Param Cov	MONDE AR	PUMONDE	MDN	True LL
Sin Normal	0.115	0.118	**0.155**	0.130	0.136				0.134	0.176
Sin T	-0.200	-0.193	**-0.194**	-0.205	**-0.178**				-0.317	-0.163
Inv Sin Normal	0.186	0.212	**0.253**	0.226	0.174				0.227	
Inv Sin T	-0.083	-0.109	-0.132	**-0.063**	-0.089				-0.199	
MV Nonlinear	-6.196	-6.067	-5.695	-5.281	-5.095	-5.074	-5.135	**-5.033**	-5.247	-4.973



(a) sin normal (b) sin t

(c) inv sin normal (d) inv sin t

Figure S5: Generated Univariate Response Data.

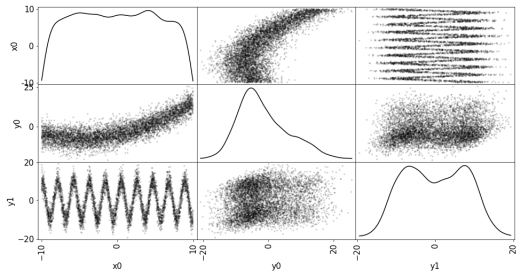
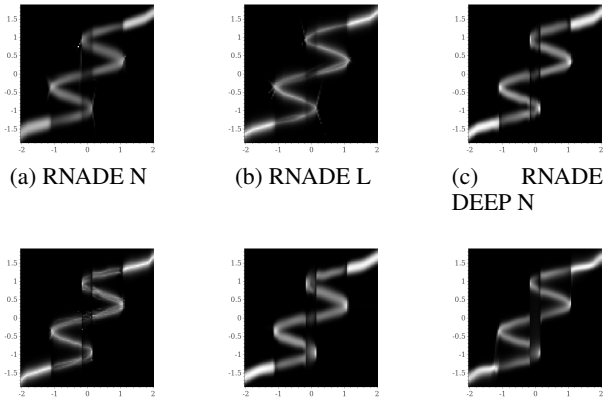


Figure S6: Generated Multivariate Response Data.



(a) RNADE N (b) RNADE L (c) RNADE DEEP N

(d) RNADE DEEP L (e) MDN (f) MONDE

Figure S7: Density Heatmap - Inv Sin Normal.

MDN models have difficulty in encoding the probabilistic structure, but our models can capture it well, which is also confirmed by the test log-likelihood evaluations shown in Table 2.

S4.4 SMALL UCI DATASETS

The results from the experiments using smaller UCI datasets (Dua and Graff, 2019) are shown in Table 3. The UCI datasets are preprocessed in the same way as done by Uria et al. (2013) i.e., we removed categorical variables and variables which have absolute value of Pearson coefficient correlation with other variable larger than 0.98. We use each UCI dataset in two separate experiments. Firstly, by assuming the two last columns as response variables (ordering as defined by the documentation of the data), while the remaining columns constitute the covariates. These are the datasets with suffix "2D". Secondly, we perform experiments by using all columns as response variables, hence assuming the covariate vector is empty. These datasets are given the suffix "unsupervised".

MONDE models achieve the best results among all model trained in these datasets.

We report that when we trained the models on datasets composed solely of continuous variables where some columns have a considerably small number of unique values (but still specified as real type variables by the dataset documentation), the RNADE models tend to "overfit" by placing very narrow Gaussians at particular points (the test likelihood is high, but this is an artefact of treating essentially discrete data as if it had a density). This was especially a problem for RNADE model, when one of such problematic variables was the first variable in the autoregressive expansion for the joint probability. In this case, because train, validation and test data contained a lot of points with the same values, RNADE could create overfitted first unconditional densities. This is not a real-

Table 3: Mean Loglikelihoods - Small UCI Datasets.

	RNADE Laplace	RNADE Normal	RNADE Deep Normal	RNADE Deep Laplace	MONDE Const Cov	MONDE Param Cov	MONDE AR	PUMONDE	MDN
UCI Redwine 2D	-2.543	-2.506	-3.310	-2.343	-2.672	-2.462	**-1.795**	**-1.997**	-2.250
UCI Redwine Unsupervised	-6.574	-6.496	-8.244	-8.297	-2.992	**-1.879**	-8.077		-8.676
UCI Whitewine 2D	-1.958	-1.956	-1.957	-1.968	-1.910	**-1.891**	**-1.915**	**-1.901**	-1.940
UCI Parkinsons 2D	-1.406	-1.323	-1.424	-2.910	-4.032	-4.766	**-1.134**	-1.254	-1.368
UCI Parkinsons Unsupervised	0.999	0.304	-3.265	-3.214	1.328	-3.654	**3.055**		-0.870

istic training procedure, since the data here is discrete for all practical purposes, resulting on unbounded test "densities" being easily achieved depending on the minimal scale allowed for a Gaussian mixture component in the training procedure. In Figure S9, we show an example of the $f(y_1)$ density estimated by the RNADE model using a mixture of Laplace distributions, and the density of the same variable estimated by the autoregressive MONDE model. The data is the UCI whitewine dataset (used as an unsupervised case i.e. all variables treated as response variables with covariate set being empty). We checked that MONDE models were not impacted by this spurious "continuous" variables and fitted smooth distributions. In our final experiment runs, we imposed the rule to remove a column if it has below 5% of unique values, compared to the number of samples. Only when we removed such columns we were able to train the RNADE models to the satisfactory level of generalization.

S4.5 FINANCIAL DATASETS

The results from experiments with financial datasets are shown in Table 4. We use two different sources of the financial data.

The first source is the the Yahoo service[1] from which we downloaded exchange traded funds dataset. We obtain two time series of daily close prices between dates: 03.01.2011 and 14.04.2015. The "ETF 1D" are daily returns of the SPY financial instrument. The response and covariate variables in this dataset are consecutive returns in the time series accordingly. The "ETF 2D" are daily returns of the SPY and DIA symbols. The returns are aggregated into the final dataset using the same transformation as in the fist univariate case, but this time both instrument returns are combined together so response and covariate vectors have two components.

The second source is the FXCM repository[2] from which we downloaded the foreign exchange tick data. We downloaded prices for the following currency pairs: EURUSD, GBPUSD, USDJPY, USDCHF, USDCAD, NZDUSD, NZDJPY, GBPJPY for the period between 05.01.2015 and 30.01.2015. Each currency pair dataset contains top of the book tick level bid and offer prices. We computed the returns of the mid point prices for these time series. Then we subsumpled each time series using a 1 minute interval and aligned all of them into one data frame. This table of aligned currency pairs' returns was used to create three datasets. "FX EUR predicted" contains the EURUSD returns as the response variable and ten previous historical returns from all currency pairs as covariates i.e. if we have EURUSD return at time t, we collected for this response covariates as returns for all instruments at times $t-1, t-2, \ldots, t-10$. The "FX EUR and GBP predicted" dataset was constructed as the previous one but with two response variables i.e. EURUSD and GBPUSD returns and the covariates constructed from all four previous historical returns plus hour of day variable. The "FX all assets predicted" dataset contains all contemporaneous currency pairs returns as response and two autoregressive returns of all currency pairs as covariates. We see from the results in Table 4 that there is an improvement in using our approach in an autoregressive representation and other versions of our model are also comparable with recently proposed solutions.

S4.5.1 Classification

In Section 4.2 we used the following foreign exchange instruments to construct the experiment dataset: AUDCAD, AUDJPY, AUDNZD, EURCHF, NZDCAD, NZDJPY, NZDUSD, USDCHF, USDJPY, EURUSD, GBPUSD and USDCAD.

S4.5.2 Bivariate likelihood

In Section 4.5 we used the following foreign exchange instruments to construct the experiment dataset: AUDCAD, AUDJPY, AUDNZD, AUDCHF, EURAUD, EURCHF, NZDCAD, CADCHF, EURJPY, NZDJPY, GBPNZD, GBPJPY, NZDUSD, USDCHF, GBPCHF, USDJPY, EURUSD, GBPUSD, EURGBP, USDCAD and NZDCHF.

[1] https://github.com/ranaroussi/yfinance
[2] https://github.com/fxcm/MarketData

Table 4: Mean Loglikelihoods - Financial Datasets.

	RNADE Laplace	RNADE Normal	RNADE Deep Normal	RNADE Deep Laplace	MONDE Const Cov	MONDE Param Cov	MONDE AR	PUMONDE	MDN
ETF 1D	**-1.416**	-1.495	**-1.422**	**-1.408**	**-1.383**			**-1.398**	
ETF 2D	-1.501	**-1.472**	-1.857	-1.588	**-1.426**	**-1.466**	**-1.401**	-1.599	**-1.441**
FX EUR Predicted	**-1.054**	-1.074	-1.093	-1.272	-1.081			-1.185	
FX EUR GBP Predicted	-2.070	-2.072	-2.268	**-2.024**	-2.292	-2.162	-2.074	-2.048	-2.130
FX ALL Predicted	-2.940	-2.985	-3.479	-3.741	-4.853	-8.107	**-2.838**		-5.604

S4.6 TAIL DEPENDENCE

To compute tail dependence, we use empirical functions suggested by (Charpentier, 2012) and (Venter and Instrat, 2001):

$$\hat{\lambda}_L(u) = \frac{\sum_{k=1}^{n} \mathbf{1}(Y_{i,k} \leq \hat{F}_i^{-1}(u), Y_{j,k} \leq \hat{F}_j^{-1}(u))}{\sum_{k=1}^{n} \mathbf{1}(Y_{i,k} \leq \hat{F}_i^{-1}(u))} \tag{S1}$$

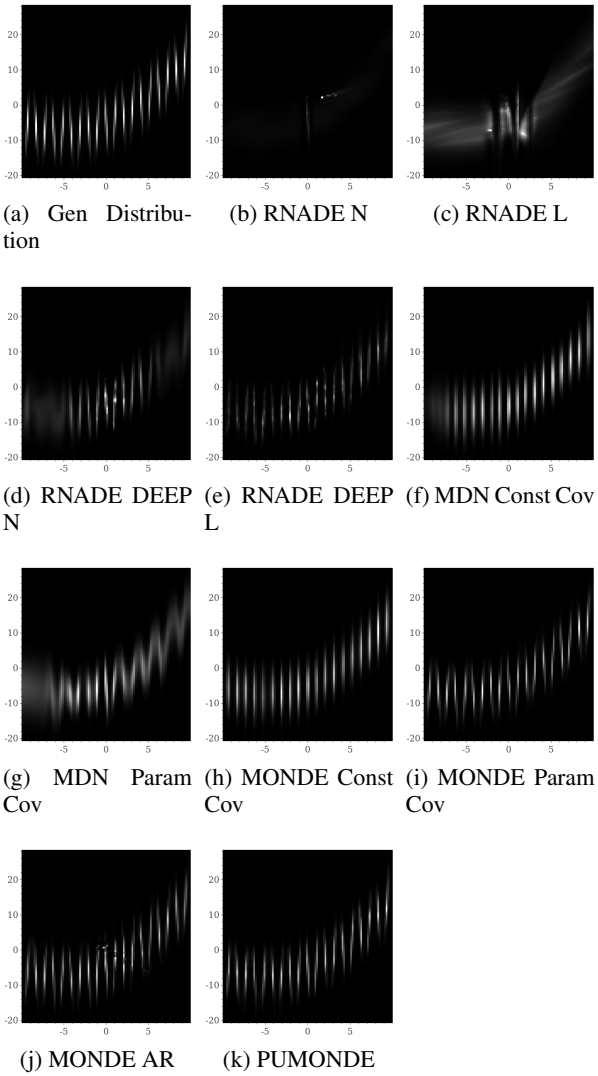
$$\hat{\lambda}_R(u) = \frac{\sum_{k=1}^{n} \mathbf{1}(Y_{i,k} > \hat{F}_i^{-1}(u), Y_{j,k} > \hat{F}_j^{-1}(u))}{\sum_{k=1}^{n} \mathbf{1}(Y_{i,k} > \hat{F}_i^{-1}(u))}, \tag{S2}$$

for the models where it is possible to directly sample new data (like MAF and MDN), and for datasets used for training generated from the mixture model. We plug-in the distribution transformation $\hat{F}_i(y_{i,k})$ computed directly from sampled data using the rank function.

To compute tail dependence indices for models which we cannot sample new data easily but we can compute marginal distribution functions (like PUMONDE, MONDE Copula) we apply the following process: 1) Estimate the marginal quantile functions numerically conditioning it on the mean of one of the mixtures: $\hat{F}_i^{-1}(\cdot|mean(\mathbf{x}_c))$; 2) Generate vector $\mathbf{u}$ as a grid of points equidistantly spaced between $(0, 1)$ (we use the same grid $\mathbf{u}$ used for computing tail indices for models that are easy to sample from); 3) Compute $\hat{F}_k^{-1}(\mathbf{u}|mean(\mathbf{x}_c))$ for $k = i, j$ 4) Compute $\hat{F}_{ij}(\hat{F}_i^{-1}(\mathbf{u}|mean(\mathbf{x}_c), \hat{F}_j^{-1}(\mathbf{u}|mean(\mathbf{x}_c))|mean(\mathbf{x}_c))$ directly from the model and substitute it into the $\lambda_L(u)$ and $\lambda_R(u)$ equations from Section 4.3.

S5 SOURCE CODE

Source code is provided in https://github.com/pawelc/NeuralLikelihoods.



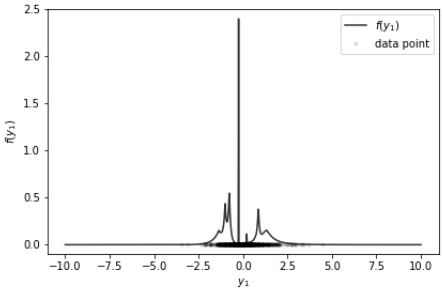
(a) Gen Distribution (b) RNADE N (c) RNADE L

(d) RNADE DEEP N (e) RNADE DEEP L (f) MDN Const Cov

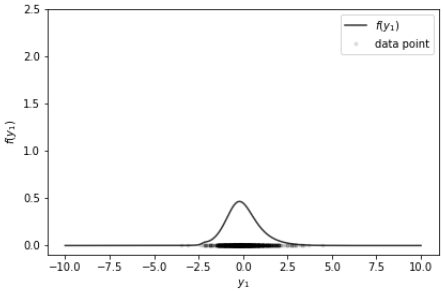
(g) MDN Param Cov (h) MONDE Const Cov (i) MONDE Param Cov

(j) MONDE AR (k) PUMONDE

Figure S8: Density Heatmap ($f(y_0, y_1 = 0 \mid x)$) - MV Nonlinear.



(a) RNADE



(b) MONDE

Figure S9: Densities estimated by the RNADE and MONDE AR models for the first variable in AR ordering when the corresponding variable had a small number of values. The data points are also shown on the x axis as black dots.

Table 5: Hyper parameters search space

EXPERIMENT	MODEL	SEARCH SPACE	DATA SET/BEST PARAMETERS
Synthetic Data, Small UCI Data, Financial Data	RNADE Normal	hidden units $\in (20, 200)$ number of components in mixture $\in (1, 100)$ learning rate $\in (10^{-4}, 10^{-2})$ batch size $= 200$	Sin Normal (30;93;0.0044) Sin T (90;7;0.0017) Inv Sin Normal (69;7;0.0022) Inv Sin T (69;7;0.0022) MV Nonlinear (194;100;0.0089) UCI Redwine 2D (170;50;0.001) UCI Redwine Unsupervised (164;10;0.0011) UCI Whitewine 2D (100;24;0.0012) UCI Whitewine Unsupervised (56;30;0.0002) UCI Parkinsons 2D (104;11;0.0064) UCI Parkinson Unsupervised (200;24;0.0006) ETF 1D (69;7;0.0022) ETF 2D (164;1;0.0041) FX EUR Predicted (176;83;0.0046) FX EUR GBP Predicted (189;78;0.0027) FX All predicted (69;7;0.0022)
Synthetic Data, Small UCI Data, Financial Data	RNADE Laplace	hidden units $\in (20, 200)$ number of components in mixture $\in (1, 100)$ learning rate $\in (10^{-4}, 10^{-2})$ batch size $= 200$	Sin Normal (100;24;0.0012) Sin T (56;30;0.0002) Inv Sin Normal (172;32;0.0011) Inv Sin T (200;67;0.0046) MV Nonlinear (90;67;0.0074) UCI Redwine 2D (20;100;0.0065) UCI Redwine Unsupervised (69;7;0.0022) UCI Whitewine 2D (69;7;0.0022) UCI Whitewine Unsupervised (100;24;0.0012) UCI Parkinsons 2D (90;67;0.0074) UCI Parkinson Unsupervised (138;25;0.0021) ETF 1D (200;1;0.01) ETF 2D (200;1;0.01) FX EUR Predicted (47;5;0.0017) FX EUR GBP Predicted (146;7;0.0011) FX All predicted (69;7;0.0022)
Synthetic Data, Small UCI Data, Financial Data	MONDE Const Cov	number of layers for x transformation $\in (0, 3)$ number of hidden units per layer for x transformation $\in (10, 200)$ number of layers y transformation $\in (1, 5)$ number of hidden units per layer for y transformation $\in (10, 200)$ learning rate $\in (10^{-4}, 10^{-2}$ batch size $= 200$	Sin Normal (1;131;4;10;0.0016) Sin T (1;112;2;110;0.0008) Inv Sin Normal (0;0;2;126;0.0004) Inv Sin T (1;112;2;110;0.0008) MV Nonlinear (3;113;1;89;0.0002) UCI Redwine 2D (0;0;4;10;0.0001) UCI Redwine Unsupervised (0;0;2;65;0.0002) UCI Whitewine 2D (0;0;3;28;0.0064) UCI Whitewine Unsupervised (0;0;3;200;0.0001) UCI Parkinsons 2D (1;112;2;110;0.0008) UCI Parkinson Unsupervised (0;0;1;10;0.0005) ETF 1D (3;200;1;200;0.0001) ETF 2D (0;0;1;200;0.0003) FX EUR Predicted (0;0;1;10;0.0001) FX EUR GBP Predicted (2;10;4;10;0.01) FX All predicted (0;0;5;200;0.0039)
Synthetic Data, Small UCI Data, Financial Data	MONDE Param Cov	number of layers for x transformation $\in (0, 3)$ number of hidden units per layer for x transformation $\in (10, 200)$ number of layers y transformation $\in (1, 5)$ number of hidden units per layer for y transformation $\in (10, 200)$ number of layers for x covariance transformation $\in (1, 3)$ number of hidden units per layer for x covariance transformation $\in (10, 200)$ learning rate $\in (10^{-4}, 10^{-2}$ batch size $= 200$	MV Nonlinear (3;200;4;116;0;0;0.0003) UCI Redwine 2D (0;0;2;200;0;0;0.01) UCI Redwine Unsupervised (0;0;2;65;0;0;0.0002) UCI Whitewine 2D (2;17;2;35;2;79;0.0068) UCI Whitewine Unsupervised (0;0;4;69;0;0;0.0001) UCI Parkinsons 2D (1;121;1;184;0;0;0.0066) UCI Parkinson Unsupervised (0;0;4;70;0;0;0.0011) ETF 2D (1;54;3;127;?;?;0.0081) FX EUR GBP Predicted (0;0;5;10;?;?;0.0001) FX All predicted (1;68;3;191;?;?;0.01)
Synthetic Data, Small UCI Data, Financial Data	MDN	number of hidden layers $\in (1, 6)$ number of hidden units per layer $\in (20, 200)$ number of components in mixture $\in (1, 100)$ learning rate $\in (10^{-4}, 10^{-2}$ batch size $= 200$	Sin Normal (6;20;1;0.0001) Sin T (4;74;45;0.0003) Inv Sin Normal (2;82;95;0.0034) Inv Sin T (1;141;60;0.0022) MV Nonlinear (3;61;54;0.0067) UCI Redwine 2D (6;20;100;0.0002) UCI Redwine Unsupervised (6;188;14;0.0099) UCI Whitewine 2D (4;74;45;0.0003) UCI Whitewine Unsupervised (6;20;100;0.01) UCI Parkinsons 2D (1;104;11;0.0064) UCI Parkinson Unsupervised (6;94;98;0.0026) ETF 1D (2;114;9;0.0068) ETF 2D (2;114;9;0.0068) FX EUR Predicted (1;24;99;0.0001) FX EUR GBP Predicted (6;172;32;0.0011) FX All predicted (6;181;59;0.0011)

(*table continues on the next page*)

Table 5: Hyper parameters search space

EXPERIMENT	MODEL	SEARCH SPACE	DATA SET/BEST PARAMETERS
Synthetic Data, Small UCI Data, Financial Data	MONDE AR	number of hidden layers for x transformation $\in (0, 3)$ number of hidden units per x transformation layer $\in (10, 200)$ number of hidden layers for y transformation $\in (1, 5)$ number of hidden units per y transformation layer $\in (10, 200)$ learning rate $\in (10^{-4}, 10^{-2}$ batch size $= 200$	MV Nonlinear (1;22;1;180;0.0001) UCI Redwine 2D (0;0;3;28;0.0064) UCI Redwine Unsupervised (0;0;1;37;0.0038) UCI Whitewine 2D (0;0;2;159;0.0007) UCI Whitewine Unsupervised (0;0;2;10;0.0036) UCI Parkinsons 2D (1;121;1;184;0.0066) UCI Parkinson Unsupervised (1;130;1;199;0.0064) ETF 1D ETF 2D (0;0;1;176;0.01) FX EUR GBP Predicted (0;0;5;10;0.01) FX All predicted (0;0;3;28;0.0064)
Synthetic Data, Small UCI Data, Financial Data	PUMONDE	number of hidden layers for x transformation $\in (1, 3)$ number of hidden units per x transformation layer $\in (10, 200)$ number of hidden layers for y transformation $\in (1, 3)$ number of hidden units per y transformation layer $\in (10, 200)$ number of hidden layers for xy transformation $\in (1, 3)$ number of hidden units per xy transformation layer $\in (10, 200)$ learning rate $\in (10^{-4}, 10^{-3}$ batch size $= 200$	MV Nonlinear (0;0;3;67;2;118;0.0002) UCI Redwine 2D (1;37;3;88;1;129;0.0007) UCI Redwine Unsupervised UCI Whitewine 2D (2;17;2;35;2;79;0.0068) UCI Parkinsons 2D (2;117;2;35;2;79;0.0068) ETF 2D (0;0;3;67;2;118;0.0002) FX EUR GBP Predicted (0;0;3;67;2;118;0.0002)
Density Estimation	MONDE MADE	number of hidden layers $\in \{8, 10\}$ number of blocks $\in \{40, 60\}$ start batch size $= 128$ batch size increments $= 3$ learning rate $= 0.001$	Power (10;40)
Density Estimation	MONDE MADE	number of hidden layers $\in \{8, 10\}$ number of blocks $\in \{60, 80\}$ start batch size $= 128$ batch size increments $= 5$ learning rate $= 0.001$	Gas (10;80) Hepmass (8;60)
Density Estimation	MONDE MADE	number of hidden layers $\in \{3, 5, 7, 8\}$ number of blocks $\in \{10, 50, 80\}$ start batch size $= 128$ batch size increments $= 5$ batch size increase patience threshold $= = 20$ learning rate $= 0.001$	Miniboone (5;10)
Density Estimation	MONDE MADE	number of hidden layers $\in \{3, 5, 7\}$ number of blocks $\in \{10, 30, 40\}$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold $= = 20$ learning rate $= 0.001$	Bsds300 (3;10)
Classification	MONDE Const Cov	number of layers for x transformation $\in \{2, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers y transformation $\in \{2, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ covariance learning rate $= 0.05$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold $= = 20$ learning rate $= 0.001$	Classification (FX) (2;100;4;100)
Classification	MONDE Param Cov	number of layers for x transformation $\in \{2, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers y transformation $\in \{2, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of layers used for covariance parametrization $\in \{2, 4\}$ number of hidden units per layer in covariance parametrization $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold $= = 20$ learning rate $= 0.001$	Classification (FX) (2;100;4;50;2;100)
Classification	PUMONDE	number of layers for x transformation $\in \{3, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers for y transformation $\in \{3, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of layers for x,y transformation $\in \{3, 4\}$ number of hidden units per layer for x,y transformation $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold $= = 20$ learning rate $= 0.001$	Classification (FX) (3;50;4;50;4;50)

(*table continues on the next page*)

Table 5: Hyper parameters search space

EXPERIMENT	MODEL	SEARCH SPACE	DATA SET/BEST PARAMETERS
Classification	NN Classifier	number of layers $\in \{2, 3, 5\}$ number of hidden units per layer$\in \{50, 100\}$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold = $= 20$ learning rate $= 0.001$	Classification (FX) (3;100)
Classification	Random Forest	number of estimators $\in \{10, 50, 100\}$ maximum tree depth $\in \{5, 10, 20\}$	Classification (FX) (100;20)
Classification	Xgb Classifier	subsample $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0\}$	Classification (FX) (0.3)
Tail Dependence Mutual Information	MONDE Const Cov	number of layers for x transformation $\in \{3, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers y transformation $\in \{3, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ covariance learning rate $= 0.05$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold = $= 20$ learning rate $= 0.001$	Mixture Process (3;100;3;50)
Tail Dependence Mutual Information	MONDE Param Cov	number of layers for x transformation $\in \{3, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers y transformation $\in \{3, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of layers used for covariance parametrization $\in \{3, 4\}$ number of hidden units per layer in covariance parametrization $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold = $= 20$ learning rate $= 0.001$	Mixture Process (3;50;3;50;4;50)
Tail Dependence Mutual Information	PUMONDE	number of layers for x transformation $\in \{3, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers for y transformation $\in \{3, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of layers for x,y transformation $\in \{3, 4\}$ number of hidden units per layer for x,y transformation $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold = $= 20$ learning rate $= 0.001$	Mixture Process (3;100;3;100;4;100)
Tail Dependence Mutual Information	MAF	number of bijectors $\in \{2, 4, 5, 8\}$ number of layers in each bijector $\in \{1, 2, 4\}$ number of hidden units per layer in bijector $\in \{64, 128, 512\}$ number of hidden units for covariate transformation $\in \{16, 32, 64\}$ batch normalization $\in \{True, False\}$ learning rate $\in \{1e^{-3}, 1e^{-4}\}$ batch size $= 128$	Mixture Process (5;1;512;16;True;0.001)
Tail Dependence Mutual Information	MDN	number of hidden layers $\in (2, 3, 5)$ number of hidden units per layer $\in (128, 512)$ number of components in mixture $\in (2, 10, 50, 100)$ learning rate $\in \{1e^{-3}, 1e^{-4}\}$ batch size $= 128$	Mixture Process (5;128;100;0.001)
Bivariate Likelihood	MONDE Const Cov	number of layers for x transformation $\in \{2, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers y transformation $\in \{2, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ covariance learning rate $= 0.05$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold = $= 20$ learning rate $= 0.001$	FX (2;50;2;50)
Bivariate Likelihood	MONDE Param Cov	number of layers for x transformation $\in \{2, 4\}$ number of hidden units per layer for x transformation $\in \{50, 100\}$ number of layers y transformation $\in \{2, 4\}$ number of hidden units per layer for y transformation $\in \{50, 100\}$ number of layers used for covariance parametrization $\in \{2, 4\}$ number of hidden units per layer in covariance parametrization $\in \{50, 100\}$ number of hidden units per layer in y transformation used for x,y transformation $= 30$ start batch size $= 128$ batch size increments $= 3$ batch size increase patience threshold = $= 20$ learning rate $= 0.001$	FX (2;50;4;100;2;100)

(*table continues on the next page*)

Table 5: Hyper parameters search space

EXPERIMENT	MODEL	SEARCH SPACE	DATA SET/BEST PARAMETERS
Bivariate Likelihood	PUMONDE	number of layers for x transformation $\in \{2, 3\}$ number of hidden units per layer for x transformation $\in \{128, 256\}$ number of layers for y transformation $\in \{2, 3\}$ number of hidden units per layer for y transformation $\in \{64, 128\}$ number of layers for x,y transformation $\in \{2, 3\}$ number of hidden units per layer for x,y transformation $\in \{64, 128\}$ number of hidden units per layer in y transformation used for x,y transformation $= 16$ batch size $= 128$ learning rate $= 0.001$	FX (2;128;3;128;3;64)
Bivariate Likelihood	MDN	number of hidden layers $\in (2, 3, 5)$ number of hidden units per layer $\in (128, 512)$ number of components in mixture $\in (2, 10, 50, 100)$ learning rate $\in \{1e^{-3}, 1e^{-4}\}$ batch size $= 128$	FX (2;512;2;0.0001)

References

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

A. Charpentier. Copulas and tail dependence, part 1. https://freakonometrics.hypotheses.org/2435, 2012. Accessed: 2019-12-28.

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

K. Diederik and B. Jimmy. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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.

D. Dua and C. Graff. UCI machine learning repository, 2019. URL http://archive.ics.uci.edu/ml.

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

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.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

B. Lang. Monotonic multi-layer perceptron networks as universal approximators. In *ICANN*, 2005.

H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *AISTATS*, 2011.

S. Liew, M. Khalil-Hani, and R. Bakhteri. Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems. *Neurocomputing*, 216, 2016.

A. Likas. Probability density estimation using artificial neural networks. *Computer Physics Communications*, 135(2), 2001.

M. Magdon-Ismail, , and A. Atiya. Density estimation and random variate generation using multilayer networks. *IEEE Trans. Neural Networks*, 13(3), 2002.

A. Minin, M. Velikova, B. Lang, and H. Daniels. Comparison of universal approximators incorporating partial monotonicity by structure. *Neural Networks*, 23(4), 2010.

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/.

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

F. Rapisarda, D. Brigo, and F. Mercurio. Parameterizing correlations: a geometric interpretation. *IMA Journal of Management Mathematics*, 18(1), 2007.

J. Sill. Monotonic networks. In *NIPS*, 1997.

J. Sill and Y. Abu-Mostafa. Monotonicity hints. In *NIPS*, 1996.

S. Smith, P-J. Kindermans, C. Ying, and V. Le Quoc. Don't decay the learning rate, increase the batch size. In *ICLR*, 2018.

E. Trentin. Soft-constrained nonparametric density estimation with artificial neural networks. In *ANNPR*, 2016.

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.

G. Venter and G. Instrat. Tails of copula. In *ASTIN*, 2001.

S. Wang. A neural network method of density estimation for univariate unimodal data. *NCAA*, 2(3), 1994.

S. Zhang. From CDF to PDF — A density estimation method for high dimensional data. *arXiv:1804.05316*, 2018.