Unconditionally Calibrated Priors for Beta Mixture Density Networks

Alix Lhéritier
Amadeus, Sophia Antipolis, France

Maurizio Filippone
Statistics Program, KAUST, Saudi Arabia

Abstract

Mixture Density Networks (MDNs) allow to model arbitrarily complex mappings between inputs and mixture densities, enabling flexible conditional density estimation, at the risk of severe overfitting. A Bayesian approach can alleviate this problem by specifying a prior over the parameters of the neural network. However, these priors can be difficult to specify due to the lack of interpretability. We propose a novel neural network construction for conditional mixture densities that allows one to specify the prior in the predictive distribution domain. The construction is based on mapping the targets to the unit hypercube via a diffeomorphism, enabling the use of mixtures of Beta distributions. We prove that the prior predictive distributions are calibrated in the sense that they are equal to the unconditional density function defined by the diffeomorphism. Contrary to Bayesian Gaussian MDNs, which exhibit tied functional and distributional complexity, we show that our construction allows to decouple them. We propose an extension allowing to model correlations in the covariates via Gaussian copulas, potentially reducing the necessary number of mixture components. Our experiments show competitive performance on standard benchmarks with respect to the state of the art.

1 INTRODUCTION

Conditional density estimation addresses the general problem of estimating a probability density function conditioned on input data. Gaussian Mixture Density Networks (GMDN) (Bishop, 1994) leverage the power of neural networks and allow one to model arbitrarily complex mappings between inputs and Gaussian mixture densities, enabling flexible conditional density estimation depending on the number of components. However, GMDN are prone to severe overfitting, and previous approaches adopt Bayesian inference to avoid it by imposing an isotropic Gaussian prior over the parameters of the GMDN and employ variational inference to approximate the posterior (Papamakarios and Murray, 2016). However, it is difficult to interpret the effect of this prior on the representation capabilities of the resulting model, and this poses a challenge in selecting sensible values for its parameters.

Choosing a good prior is of paramount importance for Bayesian Deep Learning in general. In Fortuin (2022), different approaches to define priors are surveyed. Some are defined in the weight space, like matrix-valued Gaussians (Louizos and Welling, 2016), hierarchical hyperpriors (Graves, 2011; Wu et al., 2019) and the horseshoe prior (Ghosh et al., 2018), and others are defined in the function space, like Gaussian Process priors (Flam-Shepherd et al., 2017; Tran et al., 2022), which allow encoding functional prior knowledge through the choice of kernel and mean functions.

In Trippe and Turner (2018), they use normalizing flows to represent conditional distributions and priors that encode assumptions on continuity and smoothness. However, as noted by the authors, the similarity notion used is based on closeness in the parameter space rather than the distribution space, making it dependent on the neural network architecture. In the same work, the authors speculate that Bayesian GMDNs fail to optimally capture the trade-offs between distributional and functional complexity due to tied prior variances for all weights and biases.

In this work, we consider flexible models based on neural networks, but we take a different approach to GMDNs by focusing directly on the conditional density of interest. In particular, we use a given (or learned) unconditional cumulative distribution function to map the targets to $[0, 1]$, allowing us to map the prior to the space of predictive distributions with support on $[0, 1]$. Due to this mapping, we employ mixtures of

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

Beta distributions to satisfy the domain constraint and achieve flexibility. The resulting model is a neural network that parameterizes the mixing coefficients and parameters of the Beta components. The innovation lies in the parameterization of the conditional density and in the fact that we treat the resulting model in a Bayesian way, whereby we employ Markov chain Monte Carlo sampling techniques to sample from the posterior distribution over neural network parameters, yielding predictive distributions over conditional densities.

In order to improve the applicability of the proposed model, we consider an extension which allows for the modeling of multiple target variables through Gaussian copulas. In this extension, we consider neural networks parameterizing a Lewandowski-Kurowicka-Joe (LKJ) distribution over correlation matrices which we use as the basis to construct copulas. To the best of our knowledge, this is the first Bayesian approach considering neural networks for multivariate conditional density estimation through Gaussian copulas.

In summary, the contributions of this paper are: i) we propose a novel neural network construction for conditional mixture densities that allows for a sensible specification of the prior in the predictive distribution domain; ii) we derive a theorem guaranteeing that the prior matches the given unconditional distribution; iii) we provide an analysis showing how our construction allows to independently control functional and distributional complexity; iv) we provide a novel multivariate extension based on Gaussian copulas, for which we carry out a Bayesian treatment.

2 BACKGROUND

Mixture Density Networks. Mixture Density Networks (MDNs) were proposed in Bishop (1994) to tackle conditional density estimation problems, combining the flexibility of neural networks and mixture models. Let $\mathbf{x} \in \mathbb{R}^D$ represent an input vector. For simplicity, let us consider a one-hidden-layer neural network with H hidden nodes with an activation function $\rho(\cdot)$ and O outputs. The k-th output is defined as:

$$f_k(\mathbf{x}) = \mathbf{w}_{2k}^\top \mathbf{h} + b_{2k}\,, \qquad \mathbf{h}(\mathbf{x}) = \rho(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\,,$$

where $\mathbf{h}(\mathbf{x}) \in \mathbb{R}^H$ is a vector of the outputs of the hidden units, $\mathbf{W}_1 \in \mathbb{R}^{H\times D}$ and $\mathbf{w}_{2k} \in \mathbb{R}^H$ are network weights, and $\mathbf{b}_1 \in \mathbb{R}^H$ and $b_{2k} \in \mathbb{R}$ are the biases. Let $\theta \equiv \{\mathbf{W}_1, \mathbf{b}_1\} \cup \{\mathbf{w}_{2k}, b_{2k}\}_{k=1\ldots O}$.

The idea of MDNs is to parameterize the N components of a mixture through the $O = 3N$ outputs of the neural network. For example, in Gaussian MDNs, one can use a subset of the outputs $\mathbf{f} \equiv \{f_k(\mathbf{x})\}_{k=1..O}$ to parameterize the mixing coefficients, another subset of $\mathbf{f}$ for the means of the mixture components, and the remaining outputs for their variances.

Bayesian neural networks. MDNs are prone to overfitting, due to their flexibility. One way around this is to carry out a Bayesian treatment of the neural network. For this, we need a likelihood, which we already have by considering standard MDNs, and a prior over the weights and biases of the networks. In this work, we consider a product prior on the weights and biases with zero mean and variances $\sigma^2_{\mathbf{W}_1}, \sigma^2_{\mathbf{w}_2}, \sigma^2_{\mathbf{b}_1}$, and $\sigma^2_{b_2}$. In order to characterize the posterior over model parameters, there exist several approximations, and in this work we resort to Markov chain Monte Carlo sampling techniques, as discussed in the experiments.

Limiting behavior of wide Bayesian neural networks. In this work, we rely on the limit of infinitely wide neural networks, as we will describe in Section 3.2. This is based on a normalizing operation that depends on the distribution of $\mathbf{h}(\mathbf{x})$ that we analyze next. Given an input value $\mathbf{x}$, let $h_j(\mathbf{x})$ denote the j-th component of $\mathbf{h}(\mathbf{x})$ and $v(\mathbf{x}) \equiv \mathbb{E}_{\mathbf{W}_1,\mathbf{b}_1}\left[h_j(\mathbf{x})^2\right]$, which is the same for all j. Then, for large H, by the Central Limit Theorem, the prior distribution of $f_k(\mathbf{x})$ is approximately Gaussian (Neal, 1996) with zero mean and variance:

$$V(\mathbf{x}) \equiv \sigma^2_{b_2} + H\sigma^2_{\mathbf{w}_2} v(\mathbf{x})\,. \tag{1}$$

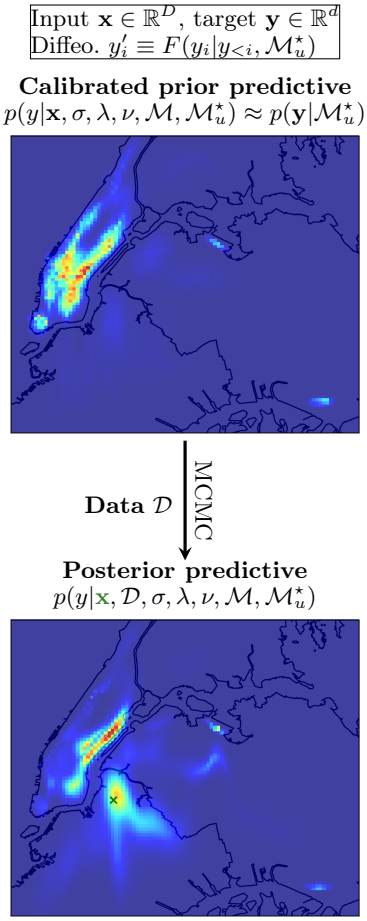
For example, when $\sigma^2_{\mathbf{W}_1} = \sigma^2_{\mathbf{w}_2} = \sigma^2_{\mathbf{b}_1} = \sigma^2_{b_2} = \sigma^2$ and $g(z) = \max(0, z)$ (ReLu activation), we get

$$v(\mathbf{x}) \equiv \mathbb{E}_{\mathbf{W}_1,\mathbf{b}_1}\left[h_j(\mathbf{x})^2\right] = \frac{1}{2}\mathrm{Var}\left((\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)_j\right) \tag{2}$$

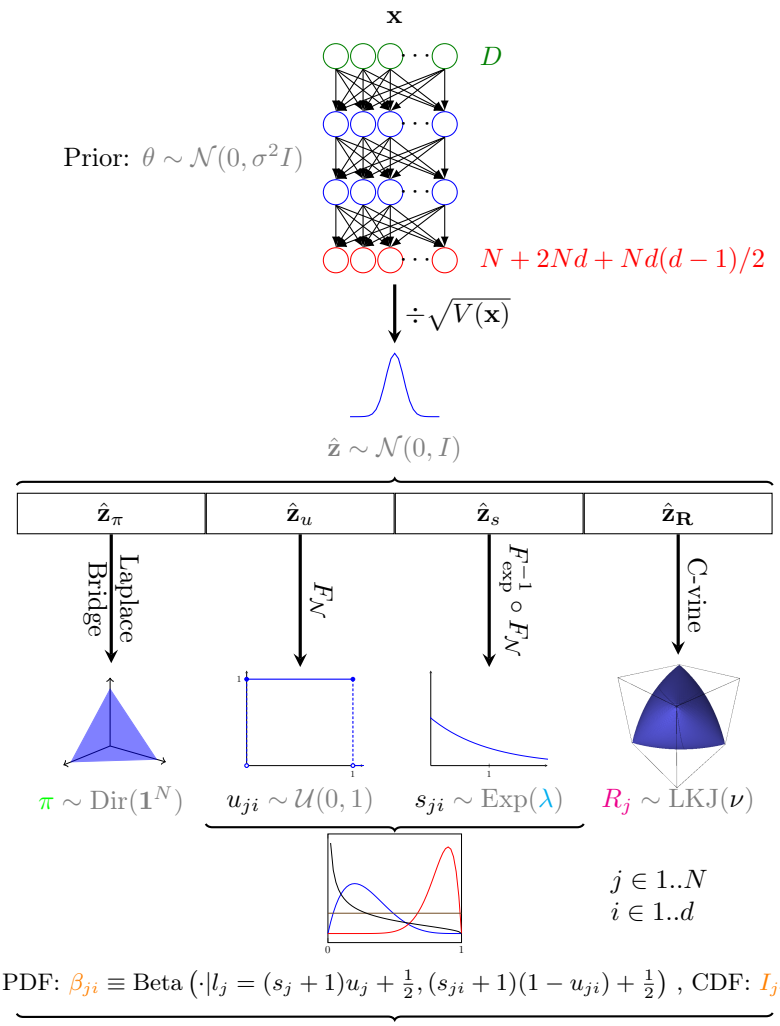
$$= \frac{\sigma^2(1 + \|\mathbf{x}\|^2)}{2}\,. \tag{3}$$

When scaling the standard deviation as $\sigma_{\mathbf{w}_2} = \omega H^{-1/2}$, for some fixed ω, and letting $H \to \infty$, sampled neural networks represent functions that are drawn from a Gaussian Process (GP) with some kernel (or covariance) function $k(\cdot,\cdot)$ (Neal, 1996). Our work is compatible with any finite neural network whose asymptotical output variance can be analytically derived. For deeper networks, other architectures and activations, Cho and Saul (2009); Lee et al. (2018); Matthews et al. (2018); Han et al. (2022) provide closed-form formulas for the covariance of the equivalent GP. Note that the Neural Tangent parameterization (Jacot et al., 2018; Sohl-Dickstein et al., 2020) is also compatible with our work, although the neural tangent kernel itself is not needed.[1]

[1] In the infinite-width limit, a neural network undergoing training by gradient descent evolves like a linear model under kernel gradient descent(Jacot et al., 2018). In our case, we do not resort to typical gradient descent training but to variants of MCMC to sample from the posterior.



(a) Process overview. Taxi Drop-off density conditioned on pick-up location $\times$ (shown) and time features (not shown).



Likelihood:
$p(\mathbf{y}|\mathbf{x}, \theta, \lambda, \nu, \mathcal{M}, \mathcal{M}^\star_u) = p(\mathbf{y}|\mathcal{M}^\star_u) \sum_j^N \pi_j c^{\text{Gauss}}_{R_j}(I_{j1}(y'_1), \ldots, I_{jd}(y'_d)) \prod_i^d \beta_{ji}(y'_i)$

(b) Construction overview. Distributions under the prior are shown in grey.

Figure 1

3 METHODS

Our conditional likelihood is based on a mixture of Beta distributions, thus, with support on $[0, 1]$. A given unconditional cumulative density function of the targets (or an estimate of it) is used as a diffeomorphism to squeeze the targets into $[0, 1]$ (Sec. 3.1). As depicted in Fig. 1b, the conditional likelihood is obtained by first processing the inputs through a neural network whose parameters are given a Bayesian treatment by assigning them a prior. The subsequent steps (described in Sec. 3.2,3.33.4) transform the raw outputs of the network to convert them into the parameters of the Beta mixture such that an important calibration property is obtained: the prior predictive distributions match the unconditional distribution. This desired property is stated in Eq. 4 and 8 and the result is stated as Theorem 1 in Sec. 4.1. We call our construction a Beta Mixture Density Network (BMDN). As illustrated in Fig. 1a, the rest of the process consists in, given some data $\mathcal{D} \equiv \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, estimating the posterior distribution of the parameters of the neural networks using some standard Monte Carlo Markov Chain (MCMC) procedure or some stochastic gradient based version for large datasets, using as objective the log likelihood added to the log prior of the parameters of the network.

3.1 Unconditional constraint: target squeezing and Beta mixtures

Our construction is based on an unconditional distribution that is given or learned. Let $\mathcal{M}^\star_u$ denote the

given unconditional model with density $p(y|\mathcal{M}_u^\star)$ and cumulative density function $F(y|\mathcal{M}_u^\star)$. Then, the prior $p(\theta)$ and our neural network model $\mathcal{M}$, should satisfy the following marginal constraint:

$$E_{p(\mathbf{x})p(\theta)}p(y|\mathbf{x},\theta,\mathcal{M}) = p(y|\mathcal{M}_u^\star) \text{ a.e.}\,. \tag{4}$$

Since we impose a prior on θ that does not depend on $\mathbf{x}$, we shall impose the following stronger constraint

$$E_{p(\theta)}p(y|\mathbf{x},\theta,\mathcal{M}) = p(y|\mathcal{M}_u^\star) \quad \text{a.e.} \quad \forall \mathbf{x}\,, \tag{5}$$

implying Eq. 4. Let us apply the (unconditional) Probability Integral Transform on the target variable, i.e., $y' \equiv F(y|\mathcal{M}_u^\star)$ and $y' \sim \mathcal{U}(0,1)$. Note that the change of variable formula implies:

$$p(y|\mathbf{x},\theta,\mathcal{M},\mathcal{M}_u^\star) = p(y'|\mathbf{x},\theta,\mathcal{M})p(y|\mathcal{M}_u^\star)\,, \tag{6}$$

since $p(y|\mathcal{M}_u^\star) = \frac{d}{dy}F(y|\mathcal{M}_u^\star)$. Then, the marginal constraint of Eq. 5 can be expressed as

$$E_{p(\theta)}p(y'|\mathbf{x},\theta,\mathcal{M})p(y|\mathcal{M}_u^\star) = p(y|\mathcal{M}_u^\star) \text{ a.e. } \forall \mathbf{x}\,, \tag{7}$$

which is satisfied if

$$E_{p(\theta)}p(y'|\mathbf{x},\theta,\mathcal{M}) = 1 \quad \text{a.e.} \quad \forall \mathbf{x}\,. \tag{8}$$

In Section 4.1, we will show that Eq. 8 can be enforced by Beta mixtures. As shown in (Diaconis and Ylvisaker, 1985), any continuous density on $[0,1]$ can be approximated arbitrarily well—if a sufficient number of components N is used—by Beta mixtures of the following form:

$$p(y'|\mathbf{a},\mathbf{b},\boldsymbol{\pi}) \equiv \sum_{j=1}^{N} \pi_j \beta(y'|a_j,b_j) \tag{9}$$

$$\text{s.t.} \quad a_j \in \mathbb{N}^+, b_j \in \mathbb{N}^+\,, \quad \sum_{j=1}^{N}\pi_j = 1\,, \tag{10}$$

where β is the Beta density:

$$\beta(y'|a,b) \equiv \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} y'^{a-1}(1-y')^{b-1}\,. \tag{11}$$

As we will see later, in practice, the parameters a_j, b_j will be relaxed to take real values in $[1/2,\infty)$.

3.2 Normalization and splitting

In order to parameterize such mixtures conditioned on $\mathbf{x}$, the parameters $\mathbf{a},\mathbf{b},\boldsymbol{\pi}$ will be computed from the $O = 3N$ outputs of some neural network for which the variance $V(\mathbf{x})$ can be computed, in the infinite-width case, in closed form (see, e.g., Han et al. (2022)). Note that this construction is applicable to a wide spectrum of finite-size architectures since, as shown in Matthews et al. (2018), the convergence to the limit is fast. In order to make the output distribution independent of $\mathbf{x}$, we divide the outputs by their (infinite-width limit) variance from Eq. 1

$$\hat{f}_k(\mathbf{x}) \equiv \frac{f_k(\mathbf{x})}{\sqrt{V(\mathbf{x})}} \sim \mathcal{N}(0,1)\,. \tag{12}$$

Then, $\hat{\mathbf{f}}(\mathbf{x})$ is split in three parts, one corresponding to each vector parameter of the mixture, and further transformed to obtain the desired prior distributions for the parameters of the mixture, as we discuss next.

3.3 Weights of the components

Let $\hat{\boldsymbol{z}}_\pi \equiv \left(\hat{f}_1(\mathbf{x}),\ldots,\hat{f}_N(\mathbf{x})\right)$ be a multivariate Normal variable that will be transformed to obtain a vector of weights. We put a Dirichlet prior with parameters $\alpha_j = 1$ on the weights, since it is a uniform one. In order to transform $\hat{\mathbf{z}}_\pi$ into an approximate N-dimensional Dirichlet variable, we resort to the Laplace bridge (MacKay, 1998). Consider the N-dimensional $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$ defined as the softmax of $\mathbf{z} \in \mathbb{R}^N$:

$$\pi_j(\mathbf{z}) = \frac{\exp(z_j)}{\sum_{l=1}^{N}\exp(z_l)}\,. \tag{13}$$

Then, the Dirichlet density can be expressed in the softmax basis

$$\text{Dir}_{\mathbf{z}}(\boldsymbol{\pi}(\mathbf{z}) \mid \boldsymbol{\alpha}) := \frac{\Gamma\left(\sum_{k=1}^{N}\alpha_k\right)}{\prod_{k=1}^{N}\Gamma(\alpha_k)} \prod_{k=1}^{N}\pi_k(\mathbf{z})^{\alpha_k}\,. \tag{14}$$

As shown in Hennig et al. (2012), this density can be accurately approximated by a Gaussian through a Laplace approximation with mean $\boldsymbol{\mu} \in \mathbb{R}^N$ and symmetric positive definite covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{N\times N}$, given by

$$\mu_k = \log\alpha_k - \frac{1}{N}\sum_{l=1}^{N}\log\alpha_l\,,$$

$$\Sigma_{k\ell} = \delta_{k\ell}\frac{1}{\alpha_k} - \frac{1}{N}\left[\frac{1}{\alpha_k} + \frac{1}{\alpha_\ell} - \frac{1}{N}\sum_{u=1}^{N}\frac{1}{\alpha_u}\right]\,,$$

where δ is Kronecker delta. For the non-informative $\alpha_k = 1$, it simplifies to $\mu_k = 0$ and $\Sigma_{k\ell} = \delta_{k\ell} - \frac{1}{N}$. Therefore, it suffices to transform the variable $\hat{\mathbf{z}}_\pi$ into variables distributed as $\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$ by applying the linear transformation $\tilde{\boldsymbol{z}} \equiv \mathbf{A}\hat{\boldsymbol{z}}_\pi$, with $\mathbf{A}$ s.t. $\mathbf{A}\mathbf{A}^\top = \boldsymbol{\Sigma}$.

3.4 Scale and location of the components

Let us consider the reparameterization of the Beta distribution proposed in Robert and Rousseau (2002)[2]:

$$a_j = s_j u_j + 1\,, \quad b_j = s_j(1-u_j) + 1\,, \tag{15}$$

[2]We add one to satisfy Eq. 10.

where $u_j \in [0,1]$ is the location and $s_j > 0$ is the scale, which can be rewritten as

$$a_j = l_j + 1\,, \quad b_j = s_j - l_j + 1\,. \tag{16}$$

with $l_j = s_j u_j$. However, in order to achieve the desired calibration, we shall consider the parameterization of Eq. 16 with $l_j = u_j(s_j+1) - 1/2$ instead (see Thm. 1 and Appendix B.

We use the remaining $2N$ outputs to obtain the desired scale and location parameters. Let $F_{\mathcal{N}}$ be the Normal CDF. The uniformly distributed location is obtained as follows

$$u_j \equiv F_{\mathcal{N}}(\hat{f}_{N+j}(\mathbf{x}))\,. \tag{17}$$

For the scale, we choose an exponential distribution to encourage spreading the mixture components under the prior, which is obtained as

$$u'_j \equiv F_{\mathcal{N}}(\hat{f}_{2N+j}(x))\,, \quad s_j \equiv -\frac{1}{\lambda}\log(1-u'_j)\,, \tag{18}$$

as $Q(p) = -\frac{1}{\lambda}\log(1-p)$ is the quantile function of the exponential distribution with parameter λ.

4 PROPERTIES OF THE PRIOR

4.1 Unconditional calibration

We now prove the desired calibration of Eq. 8, under the assumption of rounded Beta parameters.

Theorem 1. *Consider the priors and the transformations defined before, and rounded Beta parameters* $\bar{a} = \bar{l} + 1, \bar{b} = \bar{s} - \bar{l} + 1$ *where* $\bar{l} = \lfloor u(\bar{s}+1) - 1/2 \rceil$ *and* $\bar{s} = \lfloor s \rceil$*. Then, in the infinite width-limit, we have Eq. 8 satisfied, i.e.:*

$$E_{p(\theta)} p(y'|\mathbf{x}, \theta, \mathcal{M}) = p(y') = 1 \quad \forall \boldsymbol{x}\,. \tag{19}$$

Proof. Note that, under the infinite-width limit assumptions, the variables $\bar{\mathbf{a}}, \bar{\mathbf{b}}, \boldsymbol{\pi}$ are independent. Then,

$$E_{p(\theta)} p(y'|\mathbf{x}, \theta, \mathcal{M}) = E_{p(\bar{\mathbf{a}}, \bar{\mathbf{b}}, \boldsymbol{\pi})} p(y'|\bar{\mathbf{a}}, \bar{\mathbf{b}}, \boldsymbol{\pi}) = \tag{20}$$

$$E_{p(\bar{\mathbf{a}}, \bar{\mathbf{b}}, \boldsymbol{\pi})} \sum_{j=1}^{N} \pi_j \beta(y'|\bar{a}_j, \bar{b}_j) \tag{21}$$

$$= E_{\boldsymbol{\pi}} \sum_{j=1}^{N} \pi_j E_{p(\bar{a}_j, \bar{b}_j)} \beta(y'|\bar{a}_j, \bar{b}_j) \tag{22}$$

$$= E_{\boldsymbol{\pi}} \sum_{j=1}^{N} \pi_j E_{p(\bar{s}_j)} E_{p(\bar{l}_j|\bar{s}_j)} \beta(y'|\bar{l}_j + 1, \bar{s}_j - \bar{l} + 1)\,.$$

Since we assume rounded locations and scales, we have that $\bar{s} \in \mathbb{Z}_0^+$, $\bar{l} \in 0, \ldots, \bar{s}$ and $p(\bar{l}|\bar{s})$ is a discrete uniform distribution. Then,

$$E_{p(\bar{l}|\bar{s})} \beta(y'|\bar{l}+1, \bar{s} - \bar{l} + 1) \tag{23}$$

$$= \frac{1}{\bar{s}+1} \sum_{\bar{l}=0}^{\bar{s}} \frac{\Gamma(\bar{l}+1+\bar{s}-\bar{l}+1)}{\Gamma(\bar{l}+1)\Gamma(\bar{s}-\bar{l}+1)} y'^{\bar{l}} (1-y')^{\bar{s}-\bar{l}} \tag{24}$$

$$= \frac{1}{\bar{s}+1} \sum_{\bar{l}=0}^{\bar{s}} \frac{(\bar{s}+1)!}{\bar{l}!(\bar{s}-\bar{l})!} y'^{\bar{l}} (1-y')^{\bar{s}-\bar{l}} \tag{25}$$

$$= \sum_{\bar{l}=0}^{\bar{s}} \binom{\bar{s}}{\bar{l}} y'^{\bar{l}} (1-y')^{\bar{s}-\bar{l}} = (y' + (1-y'))^{\bar{s}} = 1\,. \square$$

In practice, to facilitate differentiation (e.g., required by stochastic gradient-based samplers), we use s and $l = u(s+1) - 1/2$ instead of $\bar{s}$ and $\bar{l}$, respectively. Thus, the expectation of Eq. 24 turns into

$$E_{p(l)} \beta(y'|l+1, s-l+1) \tag{26}$$

$$= \frac{1}{s+1} \int_{-1/2}^{s+1/2} \frac{\Gamma(l+1+s-l+1)}{\Gamma(l+1)\Gamma(s-l+1)} y'^{l} (1-y')^{s-l} dl$$

$$= \int_{-1/2}^{s+1/2} \frac{\Gamma(s+1)}{\Gamma(l+1)\Gamma(s-l+1)} y'^{l} (1-y')^{s-l} dl\,. \tag{27}$$

Given the intricate form of the integral, we do not prove the limit for $s \to \infty$, but instead we rely on a numerical simulation to show that indeed this expression quickly converges to $p(y') = 1$ as $s \to \infty$ (see Appendix B).

4.2 Functional and distributional complexity

The following analysis aims at showing how our construction allows us to decouple functional and distributional complexity. For simplicity, we do not take into account the effect of the diffeomorphism $F(\cdot|\mathcal{M}_u^\star)$.

As noted in Trippe and Turner (2018), GMDNs suffer from tied functional and distributional complexity. Roughly speaking, we can define functional complexity as the ability of a function class $\mathcal{F}$ to map a particular $\mathbf{x}$ to any particular $\mathbf{z}$. For example, in the case of polynomials, higher complexity is achieved with higher order. In the case of the class of functions defined by Bayesian Neural Networks, higher functional complexity is achieved by larger networks (universality) and larger variance of the prior over the network parameters, allowing for higher model flexibility. On the other hand, distributional complexity, in a mixture setup, depends on the number of components and how concentrated they are. Since there is no standard definition for distributional complexity, we resort instead to the differential entropy, which characterizes how confined to a small volume a random variable is (Cover, 1999, Thm. 8.2.3) (higher entropy corresponds to a more dispersed variable). Note that for a mixture of Diracs, the differential entropy is $-\infty$. As shown in Appendix

B.2, the negative differential entropy of a Beta mixture defined by a neural network-based function f is $-H(p(y|\mathbf{x}, f)) = O(\lambda^{-1} \log \lambda^{-1})$ and, thus, λ^{-1} allows to control the distributional complexity.

In Trippe and Turner (2018), priors encoding assumptions on continuity and smoothness of the conditional distributions are used. However, the similarity notion used is based on closeness in the parameter space rather than the distribution space. Instead, we want to analyze the the prior sensitivity of the output distributions with respect to the input, with the distribution space perspective. Therefore, we resort to the input-parameterized Fisher Information Matrix (FIM)

$$I_{Y|f}(\mathbf{x}) = E_{p(y|\mathbf{x},f)} \left[\nabla_{\mathbf{x}} \log p(y|\mathbf{x}, f) \nabla_{\mathbf{x}} \log p(y|\mathbf{x}, f)^{\intercal} \right].$$

The quantity $\sqrt{|I_{Y|f}(\mathbf{x})|}d\mathbf{x}$ is the volume measure in the Riemannian geometry. In other words, it converts a small displacement $d\mathbf{x}$ at $\mathbf{x}$ to a volume element in model space $dm_{\mathbf{x}} = \sqrt{|I_{Y|f}(\mathbf{x})|}d\mathbf{x}$ (see, e.g., Ly et al. (2017)). Expected prior sensitivity in the relationship between $\mathbf{x}$ and conditional distributions can be measured by $E_f \left[\sqrt{|I_{Y|f}(\mathbf{x})|} \right]$ for each $\mathbf{x}$. Therefore, this quantity depends both on functional and distributional complexity. As shown in Appendix B.3, this quantity depends on λ^{-1} as $O\left(\lambda^{-D/2}\right)$, on the normalized output of the neural network $\hat{f}(\mathbf{x})$ and on the partial derivatives $\frac{\partial \hat{f}_k}{\partial x_i}$.

We analyze the partial derivative, under the prior, in the infinite-width limit of a neural network with Gaussian prior $\mathcal{N}(0, \sigma^2/N_l)$ on each parameter (where N_l is the number of nodes of the previous layer), yielding an equivalent Gaussian Process $\mathcal{GP}$ from which a function f is sampled. Consider the operator acting on functions in $\mathbb{R}^D$ and returning functions in $\mathbb{R}^D$: $T : f(\cdot) \to f(\cdot)/\sqrt{V(\cdot)}$. Note that this operator is linear since for a scalar a and two functions f, g we have:

$$T(af + g)(\mathbf{x}) = (af + g)(\mathbf{x})/\sqrt{V(\mathbf{x})} \tag{28}$$

$$= af(\mathbf{x})/\sqrt{V(\mathbf{x})} + g(\mathbf{x})/\sqrt{V(\mathbf{x})}\ . \tag{29}$$

Since $f \sim \mathcal{GP}(\mu, k)$ and GPs are closed under linear operators Rasmussen and Williams (2006), Tf can be seen as a sample from another GP with mean $T(\mu)$ and kernel function $k = T_1 T_2 k$, where T_i denotes the application of T to the function of the i-th argument. In our case, we get $\hat{f} \sim \mathcal{GP}\left(0, k(\mathbf{x}, \mathbf{x}')/\left(\sqrt{V(\mathbf{x})}\sqrt{V(\mathbf{x}')}\right)\right)$. Next, we consider two activation functions and their equivalent kernels, representing two different examples in terms functional complexity.

In the case of a single hidden-layer ReLU network with prior $\mathcal{N}(0, \sigma^2/N_l)$ where N_l is the number of nodes of the previous layer, we get, in the infinite-width limit, the equivalent GP (Cho and Saul, 2009):

$$f \sim \mathcal{GP}(0, k) \text{ s.t. } \theta = \cos^{-1}\left(\frac{\mathbf{x}^{\intercal}\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}\right) \text{ and} \tag{30}$$

$$k(\mathbf{x}, \mathbf{x}') = \frac{\sigma^2 \|\mathbf{x}\|\|\mathbf{x}'\|}{2\pi} \left(\sin|\theta| + (\pi - |\theta|)\cos\theta\right)\ . \tag{31}$$

Note that this GP is not stationary and the kernel depends on σ^2. After applying the transformation T with $V(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) = \frac{\sigma^2\|\mathbf{x}\|^2}{2\pi}$, we obtain a GP with kernel function $k_V(\mathbf{x}, \mathbf{x}') = \sin|\theta| + (\pi - |\theta|)\cos\theta$, which is independent of σ^2 and magnitude-invariant (since it depends only on the angle θ between the inputs). For scalar inputs, $\theta = \cos^{-1}(\text{sign}(x)\text{sign}(x'))$ and, as the differentiation operator is also linear, we have that $\frac{d}{dx}\hat{f}\,(x)$ can be seen as a sample from a GP with zero mean and kernel function obtained by differentiating k_V with respect to each argument, which yields a constant kernel function equal to zero everywhere except at 0 (where it is infinite, in the distribution sense).

In the case of a single hidden-layer network with the sine activation function $\rho(t) = \sqrt{2}\sin(t + \pi/4)$and a prior $\mathcal{N}(0, \sigma^2)$, we get an equivalent GP with the RBF kernel $k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\frac{\sigma^2}{2D}\|\mathbf{x} - \mathbf{x}'\|^2)$, where D is the input dimension (Rahimi and Recht, 2007), and therefore with $V(\mathbf{x}) = \sigma^2$. For scalar inputs, it yields a derivative GP with kernel $\frac{\partial}{\partial x \partial x'} k_V\,(x, x') = \left(\sigma^2 - \sigma^4\,(x - x')^2\right)\exp\left(-\sigma^2\frac{(x-x')^2}{2}\right)$, which still depends on σ^2. Other activation functions with their equivalent GP kernels can be found in Han et al. (2022).

We empirically evaluate the entropy and the square root determinant of the input-parameterized FIM for a neural network with an input layer of dimension $D = 3$, one hidden-layer of 128 nodes, $N = 3$ components, with ReLU and sinusoidal activations. We vary the parameter σ of the Gaussian prior and λ for BMDN. We compute Monte Carlo estimates by averaging the log density of samples y from the mixtures for the differential entropy, and the score products for the FIM, over samples of the inputs $\mathbf{x}$ from $\mathcal{N}(0, 1)$ and samples of the weights $\mathbf{w}$ from the prior. The score function is obtained via automatic differentiation. In Fig. 2 and 3, we see, for GMDN, that the differential entropy decreases while the average square-root of the FIM increases with σ, confirming the tied functional and distributional complexity. For BMDN, both the negative entropy and the Fisher sensitivity increase with λ^{-1} and the former is invariant with respect to σ, as expected. In the case of ReLU, we see that the Fisher information is approximately invariant to σ as anticipated by the previous analysis on the infinite-width case. On the other hand, for the RBF case, the Fisher information increases with σ, showing that the functional complexity increases. These considerations

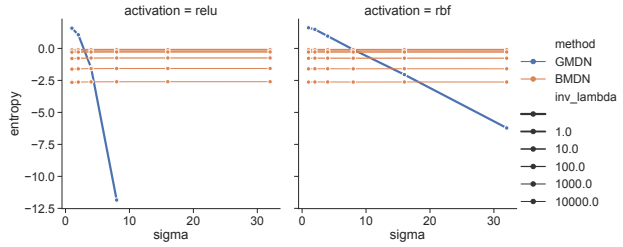


Figure 2: Average differential entropy.

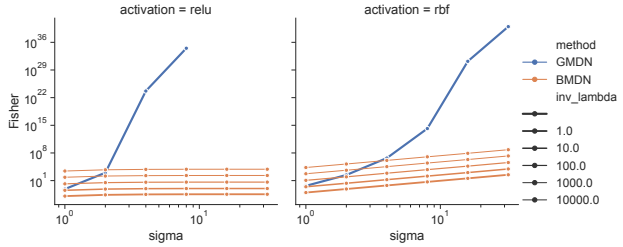


Figure 3: Average square-root determinant of the input-parameterized Fisher information Matrix.

make BMDN easier to work with, for flexibility reasons and also for numerical stability.

5 MULTIVARIATE EXTENSION VIA GAUSSIAN COPULAS

We now consider the multivariate case where the target variable $\mathbf{y} \in \mathbb{R}^d$. Let $F_i^\star(y_i) \equiv F_i(y_i|\mathcal{M}_u^\star)$ denote the unconditional CDF of the i-th component of the target variable and $\mathbf{y}' \equiv (F_1^\star(y_1), \dots, F_d^\star(y_d))$. Copulas allow to separate the univariate marginals (Beta distributions in our case) from all other dependence factors specified by a correlation matrix. Then, the neural network can be extended to the multivariate case in the following way.

The first N outputs, correspond as before to the weights π_j of the N components. The next $2Nd$ outputs allow to obtain the parameters a, b of each marginal of each component of the mixture using the same construction as in the univariate case. Let $F_{a_{ji},b_{ji}}$ denote the Beta CDF for the i-th target variable of the j-th mixture component and $f_{a_{ji},b_{ji}}$ the corresponding density.

The last $Nd(d-1)/2$ outputs correspond to N correlation matrices allowing to correlate the marginal Beta distributions of each component via a Copula construction. We place an LKJ distribution (Lewandowski et al., 2009) with parameter ν on the correlation matrices R. $\nu = 1$ defines a uniform distribution over all correlation matrices. In order to transform the normal variables $\mathbf{z}$ into LKJ ones, we use the C-vine sampling approach that uses Beta samples to generate correlation matrices in the Cholesky space, which is desirable for numerical stability (see Appendix C for more details). In order to obtain the Beta samples from $\mathbf{z}$ in a numerically stable way, we resort to the Laplace bridge described before.

The j^{th} component of the multivariate mixture density is

$$h_j(y'_1, \dots, y'_d) = c^G_{\mathbf{R}}\left(F_{a_{j1},b_{j1}}(y'_1), \dots, F_{a_{jd},b_{jd}}(y'_d)\right) \cdot f_{a_{j1},b_{j1}}(y'_1) \dots f_{a_{jd},b_{jd}}(y'_d)\,, \quad (32)$$

where $c^G_{\mathbf{R}}$ is the Gaussian copula, i.e.:

$$c^G_{\mathbf{R}}(\mathbf{v}) \equiv \frac{1}{\sqrt{\det \mathbf{R}}} \exp\left(-\frac{1}{2}\left(\mathbf{u}^\intercal\left(\mathbf{R}^{-1} - \mathbf{I}\right)\mathbf{u}\right)\right)\,, \quad (33)$$

where $\mathbf{u} = \left(\Phi^{-1}(v_1), \dots, \Phi^{-1}(v_d)\right)^\intercal$ and Φ^{-1} is the inverse cumulative distribution function of the standard normal. Finally, the mixture is obtained as

$$p(\mathbf{y}|\mathbf{x}, \theta, \mathcal{M}, \mathcal{M}_u^\star) = \sum_j \pi_j h_j(y'_1, \dots, y'_d) \prod_i p_i(y_i|\mathcal{M}_u^\star)\,. \quad (34)$$

We can also consider a more general diffeomorphism $\mathbf{g}$ mapping $\mathbf{y}$ to $\mathbf{y}'$, e.g., a normalizing flow (Papamakarios et al., 2021), yielding the more general equation involving the Jacobian operator J: $p(\mathbf{y}|\mathbf{x}, \theta, \mathcal{M}) = \sum_j \pi_j h_j(y'_1, \dots, y'_d)|J\mathbf{g}(\mathbf{y})|$. In our experiments, we consider the following diffeomorphism. Assuming the joint unconditional distribution of $\mathbf{y}$ is given or learned, we squeeze the i-th component of $\mathbf{y}$ using the conditional cdf $F(y_i|\mathbf{y}_{<i})$ (conditioned on the covariates with lower index). This defines a diffeomorphism with lower triangular Jacobian with determinant equal to the unconditional multivariate joint density of $\mathbf{y}$ (see (Papamakarios et al., 2021, Section 2.2)). For a Gaussian Mixture, the conditional cdf can be easily obtained in closed form (see (Bishop, 2006, Chap. 2.3.1-2.3.2)). The resulting $\mathbf{y}'$ will have uniform marginals as before and will, in addition, be jointly uniform. In this case, the resulting mixture reads as:

$$p(\mathbf{y}|\mathbf{x}, \theta, \mathcal{M}, \mathcal{M}_u^\star) = \sum_j \pi_j h_j(y'_1, \dots, y'_d) p(\mathbf{y}|\mathcal{M}_u^\star)\,. \quad (35)$$

Note that, by Sklar's theorem (Sklar, 1959), $h_j(y'_1, \dots, y'_d)$ defines a d-dimensional density function with marginal densities $f_{a_{ji},b_{ji}}(y'_i)$. Under the prior distributions considered before, by Thm.1, $f_{a_{ji},b_{ji}}$ are uniform as desired. Additionally, $E_{p(\theta)}h_j(y'_1, \dots, y'_d)$ converges to the uniform distribution on the unit hypercube as $\nu \to \infty$ (see Appendix A) and, thus, we get the multivariate calibration $E_{p(\theta)}p(\mathbf{y}|\mathbf{x}, \theta, \mathcal{M}, \mathcal{M}_u^\star) = p(\mathbf{y}|\mathcal{M}_u^\star)$.

In terms of computational complexity order per forward pass with respect to D and d, our method is equivalent

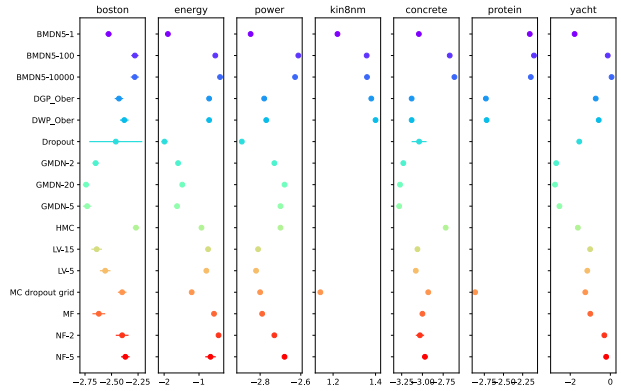


Figure 4: Results on UCI datasets in terms of log likelihood of the test set. Error bar represents std err.

to GMDN with full covariance matrix (Kruse, 2020), since the transformations allowing to obtain π, u_{ji} and s_{ji} are $O(1)$, the C-vine algorithm to obtain $\mathbf{R}_j$ is $O(d^2)$ (Lewandowski et al., 2009) and the algorithm to compute $C^G_{\mathbf{R}_j}$ is also $O(d^2)$ (see Appendix C). Note that the Cholesky decomposition required to obtain the matrix $\mathbf{A}$ to compute π scales with the cube of the number of components, but it can be precomputed. Also note that building the unconditional model is a preprocessing step that does not add any burden on each step, since the transformed labels y' can be precomputed; when taking the logarithm of Eq. 34, the unconditional term does not depend on the parameters of the network and, thus, can be omitted until the final evaluation of the likelihood. For further details on the implementation, see Appendix C.

6 EXPERIMENTS

We implemented BMDN in JAX (Bradbury et al., 2018) and the Neural Tangent python library (Novak et al., 2020), which allows to compute the equivalent kernel for a wide range of layers (e.g. attention) and activation functions. The code is available at `https://github.com/alherit/calibrated-betamix`. The following experiments focus on the ReLU and erf activation functions. For results on the RBF sinusoidal activation, see Appendix D.

6.1 UCI benchmark

Data We evaluate our method on standard UCI data with univariate targets, using the same splits as Gal and Ghahramani (2016). Inputs are standardized.

Contenders: ($*$ denotes the level of complexity)

- GMDN-*: Bayesian GMDNs with VI
- LV-*: neural networks with latent variable inputs, fitted with mean field VI (Depeweg et al., 2016).
- NF-*: normalizing flow based method (Trippe and Turner, 2018)
- MF and HMC: Bayesian neural networks models with homoscedastic Gaussian likelihoods using respectively mean field VI (MF) and Hamiltonian Monte Carlo from
- DGP and DWP: Deep GP and Deep Wishart Processes (Ober, 2024)
- MC dropout grid: Monte Carlo dropout results reported in `https://github.com/yaringal/DropoutUncertaintyExps`.

For GMDN-*, LV-*, NF-*, HMC and MF, we show the results reported in Trippe and Turner (2018).

Hyperparameters Our architecture matches that of GMDN-* (1 hidden layer with 50 nodes and $3N$ outputs) except for the final transformations and the activation function (erf in our case). Prior parameters: $\sigma^2_{\mathbf{W}_1} = \sigma^2_{\mathbf{w}_2} = \sigma^2_{\mathbf{b}_1} = \sigma^2_{b_2} = 1$, $\alpha_k = 1$ and $\lambda^{-1} : 1, 100, 10000$. We use SGLD-Adam from the SGMCMCJax library (Coullon et al., 2021) (see Appendix D for details).

Unconditional model. The unconditional model is a Gaussian mixture fitted marginally with respect to $\mathbf{x}$ with scikit-learn (Pedregosa et al., 2011) using the same number of components as the Beta MDN.

Prior calibration Fig. 11 in Appendix D.4 shows the predictive distributions sampled from the prior and their average that closely matches the given unconditional distribution for BMDN-λ^{-1}, in contrast to the predictive distributions sampled from GMDN using the same σ, which naturally lack calibration.

Results Figure 4 shows the results. Our best results are obtained with BMDN5-1000 (i.e $N = 5$ and $\lambda^{-1} = 10^4$), which outperforms or is comparable to best contenders. A typical run for the largest dataset takes around 15min on an NVIDIA Tesla T4 GPU.

6.2 Multivariate NYC Taxi dataset

We evaluate our multivariate extension on the taxi dataset[3] preprocessed as in Rothfuss et al. (2019), where the goal is to predict the drop-off location coordinates given the pick-up location and time features.

[3] https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Table 1: Results on NYC Taxi dataset (mean & stdev)

method	log likelihood
BMDN50-100 COPULA	5.07±0.01
LSCDE	4.85±0.02
CKDE R.O.T.	4.87±0.02
CKDE CV-ML	5.27±0.06
NKDE R.O.T	4.34±0.04
NKDE CV-ML	4.93±0.08

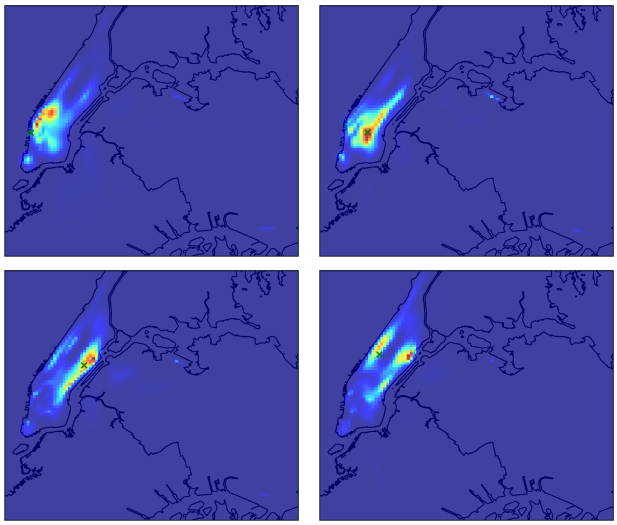


Figure 5: Copula mixture based posterior predictive distributions obtained on the taxi dataset.

Contenders We compare against the results reported in Rothfuss et al. (2019), excluding the noise regularized methods to keep the comparison fair since they affect the training set.

Hyperparameters We performed an extensive hyperparameter tuning using Bayesian optimization (number of runs=512) on a validation set over λ, σ, N and architecture choices (see Appendix D for details).

Unconditional model We use Eq. 35 with an unconditional Multivariate Gaussian mixture fitted with scikit-learn (Pedregosa et al., 2011) with tuned N.

Advantage of the copula Fig. 13 in Appendix D.5 shows the advantage in expressiveness of the copula structure as the number of components N increases, which allows to capture more complex patterns.

Results Table 1 shows that our result is competitive with the state-of-the-art methods. Fig. 5 shows some posterior predictive distributions. A typical run took around 20 min on an NVIDIA Tesla A100 GPU.

6.3 Synthetic experiment

In Appendix D.6, we consider synthetic data sampled from a uniform distribution whose support is defined by the image of the letter β. We show that the RBF sinusoidal activation allows for independent control of the functional and distributional complexities as shown in Sec. 4.2, in contrast to GMDN where they are tied (Trippe and Turner, 2018).

7 CONCLUSIONS

In this paper, we proposed a novel approach to conditional density estimation with Bayesian neural networks. Our approach is characterized by a novel parameterization of the conditional density, which yields a sensible specification of the conditional density one assumes *a priori*. The Bayesian treatment of this class of models yields uncertainties in the form of distributions over conditional densities induced by the posterior over model parameters. We also studied an extension suitable for multivariate conditional estimation through Gaussian copulas; we are not aware of other works proposing Bayesian neural networks for Gaussian copulas. Through classic UCI benchmarks and a multivariate data set on taxi pickup and dropoff locations, we showed that the results obtained by the proposed approach are competitive with the state of the art.

We conclude with remarks on the importance of model selection, which would entail tuning λ and σ^2. Model selection for Bayesian neural networks is not straightforward, but one could rely on alternatives based on approximate Bayesian inference. Another aspect worth mentioning is that, while the novel parameterization based on mixtures of Beta distributions has the advantages we advocate in this paper, MCMC sampling can sometimes be unstable and extra care needs to be taken when tuning MCMC hyper-parameters. Although the unconditional transformation is key to allow the desired calibration, in some cases, it could make the conditional learning problem harder. This can be compensated by adding more components to the mixture, since a Beta mixture can approximate arbitrarily well any distribution (Diaconis and Ylvisaker, 1985).

Acknowledgments

Thanks to Eoin Thomas, Rodrigo Acuña-Agost and Nicolas Bondoux for their insightful comments and to the anonymous reviewers whose suggestions have greatly improved this manuscript. This work has been partially supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

References

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. A. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.

C. M. Bishop. Mixture density networks. *Technical Report*, 1994.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

A. Cabezas, A. Corenflos, J. Lao, and R. Louf. Blackjax: Composable Bayesian inference in JAX, 2024.

Y. Cho and L. Saul. Kernel methods for deep learning. *Advances in neural information processing systems*, 22, 2009.

M. Cohen. The fisher information and convexity (corresp.). *IEEE Transactions on Information Theory*, 14(4):591–592, 1968.

J. Coullon and C. Nemeth. Sgmcmcjax: a lightweight jax library for stochastic gradient markov chain monte carlo algorithms. *Journal of Open Source Software*, 7(72):4113, 2022.

J. Coullon, L. South, and C. Nemeth. Stochastic gradient mcmc with multi-armed bandit tuning. *arXiv preprint arXiv:2105.13059*, 2021.

T. M. Cover. *Elements of information theory*. John Wiley & Sons, 1999.

S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

P. Diaconis and D. Ylvisaker. Quantifying prior opinion (with discussions), bayesian statist. 2, 1985.

A. Eskenazis and L. Gavalakis. Gaussian mixtures: convexity properties and clt rates for the entropy and fisher information. In *2024 IEEE International Symposium on Information Theory (ISIT)*, pages 3594–3599. IEEE, 2024.

D. Flam-Shepherd, J. Requeima, and D. Duvenaud. Mapping gaussian process priors to bayesian neural networks. In *NIPS Bayesian deep learning workshop*, volume 3, 2017.

V. Fortuin. Priors in bayesian deep learning: A review. *International Statistical Review*, 90(3):563–591, 2022.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

S. Ghosh, J. Yao, and F. Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. In *International Conference on Machine Learning*, pages 1744–1753. PMLR, 2018.

A. Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

I. Han, A. Zandieh, J. Lee, R. Novak, L. Xiao, and A. Karbasi. Fast neural kernel embeddings for general activations. *Advances in neural information processing systems*, 35:35657–35671, 2022.

P. Hennig, D. Stern, R. Herbrich, and T. Graepel. Kernel topic models. In *Artificial intelligence and statistics*, pages 511–519. PMLR, 2012.

M. Hobbhahn, A. Kristiadi, and P. Hennig. Fast predictive uncertainty for classification with bayesian deep networks. In *Uncertainty in Artificial Intelligence*, pages 822–832. PMLR, 2022.

A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

S. Kim, Q. Song, and F. Liang. Stochastic gradient langevin dynamics algorithms with adaptive drifts. *arXiv preprint arXiv:2009.09535*, 2020.

J. Kruse. Technical report: Training mixture density networks with full covariance matrices. *arXiv preprint arXiv:2003.05739*, 2020.

J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

D. Lewandowski, D. Kurowicka, and H. Joe. Generating random correlation matrices based on vines and extended onion method. *Journal of multivariate analysis*, 100(9):1989–2001, 2009.

C.-K. Li and B.-S. Tam. A note on extreme correlation matrices. *SIAM Journal on Matrix Analysis and Applications*, 15(3):903–908, 1994.

C. Louizos and M. Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International conference on machine learning*, pages 1708–1716. PMLR, 2016.

A. Ly, M. Marsman, J. Verhagen, R. P. Grasman, and E.-J. Wagenmakers. A tutorial on fisher information. *Journal of Mathematical Psychology*, 80:40–55, 2017.

D. J. MacKay. Choice of basis for laplace approximation. *Machine learning*, 33:77–86, 1998.

A. G. d. G. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018.

R. M. Neal. *Priors for Infinite Networks*, pages 29–53. Springer New York, New York, NY, 1996. ISBN 978-1-4612-0745-0.

R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

S. W. Ober. Towards improved variational inference for deep bayesian models. *arXiv preprint arXiv:2401.12418*, 2024.

G. Papamakarios and I. Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

D. Phan, N. Pradhan, and M. Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

C. P. Robert and J. Rousseau. A mixture approach to bayesian goodness of fit. *Technical report, Cahiers du CEREMADE, Universite Paris Dauphine.*, 2002.

J. Rothfuss, F. Ferreira, S. Boehm, S. Walther, M. Ulrich, T. Asfour, and A. Krause. Noise regularization for conditional density estimation. *arXiv preprint arXiv:1907.08982*, 2019.

M. Siotani. Some applications of loewner's ordering on symmetric matrices. *Annals of the Institute of Statistical Mathematics*, 19:245–259, 1967.

M. Sklar. Fonctions de répartition à n dimensions et leurs marges. In *Annales de l'ISUP*, volume 8, pages 229–231, 1959.

J. Sohl-Dickstein, R. Novak, S. S. Schoenholz, and J. Lee. On the infinite width limit of neural networks with a standard parameterization. *arXiv preprint arXiv:2001.07301*, 2020.

B.-H. Tran, S. Rossi, D. Milios, and M. Filippone. All you need is a good functional prior for bayesian deep learning. *The Journal of Machine Learning Research*, 23(1):3210–3265, 2022.

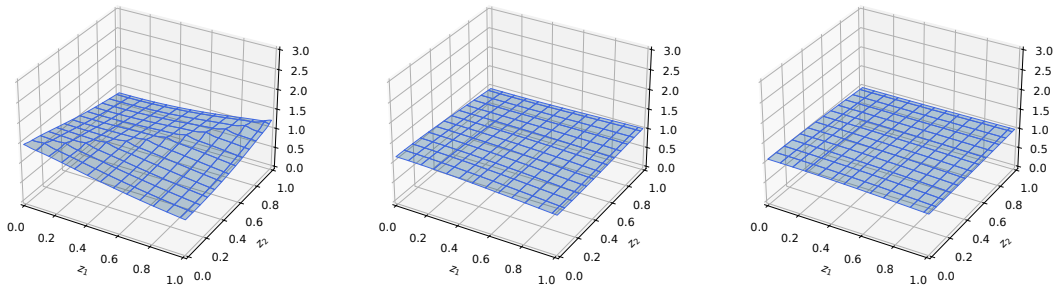
B. L. Trippe and R. E. Turner. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernández-Lobato, and A. L. Gaunt. Deterministic variational inference for robust bayesian neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes, the infinite-width limit is clearly stated when assumed]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Not Applicable.]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes, the paper was accepted and we released the code.]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes, the infinite-width limit is clearly stated when assumed and assumptions on discreteness of location and scale are stated in Thm.1]
   (b) Complete proofs of all theoretical results. [Yes, for space constraints, some details are provided in the Appendix.]
   (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes, the paper was accepted and we released the code.]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, details are provided in the Appendix]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes, public datasets are used]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]



(a) $\nu = 1$ (b) $\nu = 2$ (c) $\nu = 3$

Figure 6: Simulation of $\mathbb{E}_{\mathbf{R}}\, c^G_{\mathbf{R}}(\mathbf{z})$ for $\mathbf{R} \sim \text{LKJ}(\nu)$, $d = 2$, $\nu \in \{1, 2, 3\}$ and $\mathbf{z} \in [0, 1]^d$.

A MULTIVARIATE CALIBRATION

Theorem 2. *Under the conditions of Thm. 1 and* $\mathbf{R} \sim \text{LKJ}(\nu)$,

$$\lim_{\nu \to \infty} \mathbb{E}_{\bar{\mathbf{a}}, \bar{\mathbf{b}}, \mathbf{R}}\, h_j(y'_1, \dots, y'_d) = 1 \quad \mathbb{P}_{\mathbf{y}}\text{-}a.e\,. \tag{36}$$

where $\mathbb{P}_{\mathbf{y}}$ *is the probability measure of the target variable* $\mathbf{y}$.

Proof. Let us consider $\mathbf{y}' \in (0, 1)^d$ and define $X_\nu \equiv f_{\bar{a}_{j1}, \bar{b}_{j1}}(y'_1) \dots f_{\bar{a}_{jd}, \bar{b}_{jd}}(y'_d) c^G_{\mathbf{R}} \left(F_{\bar{a}_{j1}, \bar{b}_{j1}}(y'_1), \dots, F_{\bar{a}_{jd}, \bar{b}_{jd}}(y'_d) \right)$. In order to apply the Dominated Convergence Theorem (DCT), consider any arbitrary discrete sequence $\nu_n \to \infty$. If the limit holds for any discrete sequence, then it holds in the continuous case.

First note that $|X_\nu|$ is almost surely bounded by a quantity that does not depend on ν. Since the set of all correlation matrices is compact (see, e.g. , Li and Tam (1994)), there exists $\bar{\mathbf{R}}$ such that

$$|X_\nu| \leq f_{\bar{a}_{j1}, \bar{b}_{j1}}(y'_1) \dots f_{\bar{a}_{jd}, \bar{b}_{jd}}(y'_d) c^G_{\bar{\mathbf{R}}} \left(F_{\bar{a}_{j1}, \bar{b}_{j1}}(y'_1), \dots, F_{\bar{a}_{jd}, \bar{b}_{jd}}(y'_d) \right) \equiv \bar{X} \tag{37}$$

with $\mathbb{E}_{\bar{\mathbf{a}}, \bar{\mathbf{b}}}\, \bar{X} < \infty$ since $\mathbf{y}' \in (0, 1)^d$. Since the off-marginal elements of $\frac{\mathbf{R}_\nu + 1}{2}$ have a Beta distribution with parameters $\alpha = \beta = \nu - 1 + d/2$ (Lewandowski et al., 2009), $\mathbf{R}_\nu$ converges in probability to $\mathbf{I}$. Therefore, it also holds that $\mathbf{R}_{\nu_n} \to \mathbf{I}$ in probability. By the Continuous Mapping Theorem, X_{ν_n} converges in probability to $f_{\bar{a}_{j1}, \bar{b}_{j1}}(y'_1) \dots f_{\bar{a}_{jd}, \bar{b}_{jd}}(y'_d)$ since $c^G_{\mathbf{I}}$ is uniform. Then, by the DCT (which holds also when the convergence is in probability),

$$\lim_{n \to \infty} \mathbb{E}_{\bar{\mathbf{a}}, \bar{\mathbf{b}}, \mathbf{R}}\, X_{\nu_n} = \mathbb{E}_{\bar{\mathbf{a}}, \bar{\mathbf{b}}, \mathbf{R}}\, f_{\bar{a}_{j1}, \bar{b}_{j1}}(y'_1) \dots f_{\bar{a}_{jd}, \bar{b}_{jd}}(y'_d) = 1 \tag{38}$$

where the second equality holds by Thm. 1. The claim follows by noting that $\mathbb{P}_{\mathbf{y}} \left[\mathbf{y}' \in (0, 1)^d \right] = 1$. □

We simulate $\mathbb{E}_{\mathbf{R}}\, c^G_{\mathbf{R}}(\mathbf{z})$ for $\mathbf{R} \sim \text{LKJ}(\nu)$, $d = 2$, $\nu \in \{1, 2, 3\}$ and $\mathbf{z} \in [0, 1]^d$, and observe, in Figure 6, that it quickly converges to the uniform distribution.

B DETAILS OF THE RESULTS

B.1 Numerical simulation of the integral of Eq. 26

In practice, to facilitate differentiation (e.g., required by stochastic gradient-based samplers), we use s and $l = u(s + 1) - 1/2$ instead of $\bar{s}$ and $\bar{l}$, respectively. Thus, the expectation of Eq. 24 turns into

$$E_{p(l)} \beta(y' | l + 1, s - l + 1) = \frac{1}{s + 1} \int_{-1/2}^{s+1/2} \frac{\Gamma(l + 1 + s - l + 1)}{\Gamma(l + 1)\Gamma(s - l + 1)} y'^l (1 - y')^{s-l} dl \tag{39}$$

$$= \int_{-1/2}^{s+1/2} \frac{\Gamma(s + 1)}{\Gamma(l + 1)\Gamma(s - l + 1)} y'^l (1 - y')^{s-l} dl\,. \tag{40}$$

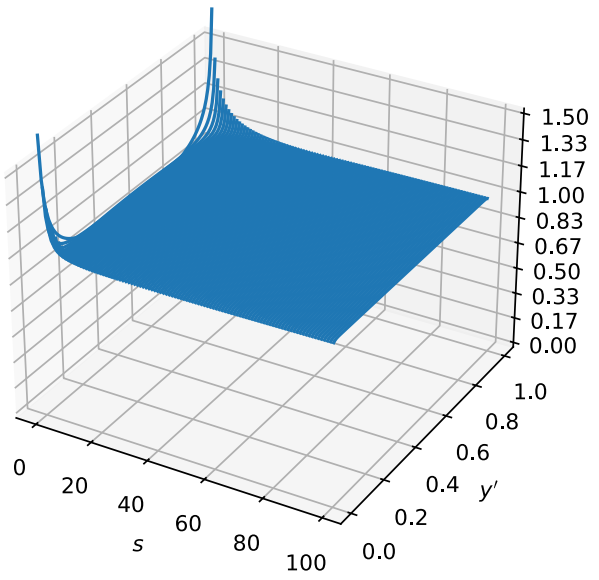


Figure 7: Convergence to uniform distribution of $E_{p(l)}\beta(y'|l+1, s-l+1)$ with continuous s and $l = u(s+1) - 1/2$ as scale s grows.

Fig. 7 shows the integral quickly converges to a uniform distribution as $s \to \infty$.

Notice that when using the more straightforward continuous parameterization s and $l = us$ instead of $\bar{s}$ and $\bar{l}$, we get

$$E_{p(l)}\beta(y'|l+1, s-l+1) = \frac{1}{s+1}\int_0^s \frac{\Gamma(l+1+s-l+1)}{\Gamma(l+1)\Gamma(s-l+1)} y'^l(1-y')^{s-l}dl \tag{41}$$

$$= \int_0^s \frac{\Gamma(s+1)}{\Gamma(l+1)\Gamma(s-l+1)} y'^l(1-y')^{s-l}dl\,, \tag{42}$$

which yields a much slower convergence as shown in Fig. 8.

B.2 Negative Differential Entropy

For notation simplicity, let us consider the mixture of Beta distributions parameterized as

$$p(y|x, f) = \sum_{i=1}^{N} \pi_i(x)\beta(y|\lambda^{-1}s_i(x)u_i(x), \lambda^{-1}s_i(x)(1-u_i(x)))\,. \tag{43}$$

Proposition 1.

$$-H(p(y|x, f)) = O(\lambda^{-1}\log\lambda^{-1})\,. \tag{44}$$

Proof. Let $L \equiv \lambda^{-1}$. By the concavity of the entropy, we have that

$$-H(p(y|x, f)) \equiv E_{p(y|x,f)}\log p(y|x, f) \leq \sum_{i=1}^{N} \pi_i H(\beta(y|Ls_iu_i, Ls_i(1-u_i))) \tag{45}$$

where we omitted the dependence on x to avoid clutter. The differential entropy of a single Beta is given by

$$H(\beta(y|Lsu, Ls(1-u))) = -(Lsu-1)(-\psi_0(Ls) + \psi_0(Lsu)) \tag{46}$$

$$+ (Ls(u-1)+1)(-\psi_0(Ls) + \psi_0(Ls(1-u))) \tag{47}$$

$$- \log\Gamma(Ls) + \log\Gamma(Lsu) + \log\Gamma(Ls(1-u)) \tag{48}$$

where ψ_0 is the digamma function. By noting that $\log\Gamma(z) = O(z\log z)$ and $\psi_0(z) = O(\log z)$, the result follows. □

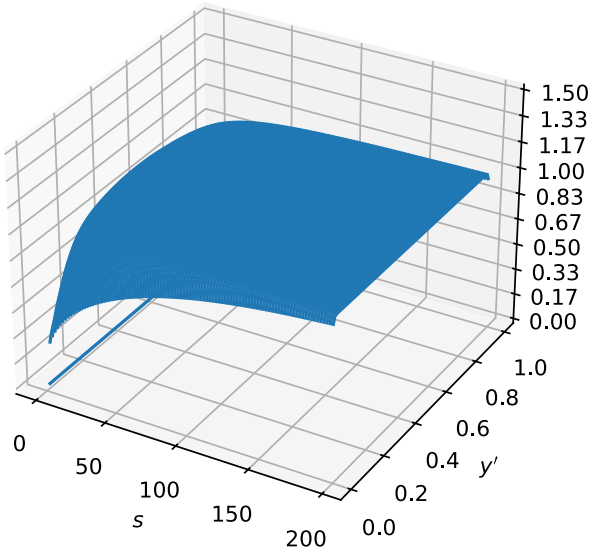


Figure 8: Convergence to uniform distribution of $E_{p(l)}\beta(y'|l+1, s-l+1)$ with continuous s and $l = us$ as scale s grows.

B.3 Fisher Information

For notation simplicity, let us consider the Beta mixture parameterization of Eq. 43. We have the following result.

Proposition 2.

$$\sqrt{|I_{Y|f}(x)|} = O\left(\lambda^{-D/2}\right) . \tag{49}$$

Proof. Let $L \equiv \lambda^{-1}$ and ψ_i denote the i-th polygamma function. Let us first consider the case of a scalar input x ($D = 1$). By the convexity of the Fisher Information (Cohen, 1968), for any $0 < \pi < 1$ and two densities p_1 and p_2, $I(\pi p_1 + (1-\pi)p_2) < \pi I(p_1) + (1-\pi)I(p_2)$. This allows us to obtain an upper bound on the Fisher Information of a Beta Mixture based on the Fisher information of a single Beta distribution which can be obtained by, first, differentiating two times with respect to x:

$$\begin{aligned}
&\frac{d}{dx^2}\beta(y|Ls(x)u(x), Ls(x)(1-u(x))) = \\
&\qquad -L^2\psi_1(Ls(x))\left(\frac{d}{dx}s(x)\right)^2 - L\psi_0(Ls(x))\frac{d^2}{dx^2}s(x) + \\
&\left(-L\left(u(x)-1\right)\frac{d}{dx}s(x) - Ls(x)\frac{d}{dx}u(x)\right)^2\psi_1(-L\left(u(x)-1\right)s(x)) + \\
&\left(Ls(x)\frac{d}{dx}u(x) + Lu(x)\frac{d}{dx}s(x)\right)^2\psi_1(Ls(x)u(x)) + \\
&\left(-L\left(u(x)-1\right)\frac{d^2}{dx^2}s(x) - Ls(x)\frac{d^2}{dx^2}u(x) - 2L\frac{d}{dx}s(x)\frac{d}{dx}u(x)\right)\psi_0(-L\left(u(x)-1\right)s(x)) + \\
&\left(L\left(u(x)-1\right)\frac{d^2}{dx^2}s(x) + Ls(x)\frac{d^2}{dx^2}u(x) + 2L\frac{d}{dx}s(x)\frac{d}{dx}u(x)\right)\log\left(1-y\right) - \\
&\left(Ls(x)\frac{d^2}{dx^2}u(x) + Lu(x)\frac{d^2}{dx^2}s(x) + 2L\frac{d}{dx}s(x)\frac{d}{dx}u(x)\right)\log\left(y\right) + \\
&\qquad\left(Ls(x)\frac{d^2}{dx^2}u(x) + Lu(x)\frac{d^2}{dx^2}s(x) + 2L\frac{d}{dx}s(x)\frac{d}{dx}u(x)\right)\psi_0(Ls(x)u(x))
\end{aligned} \tag{50}$$

and then taking the negative expectation with respect to y distributed as the single Beta distribution, which is obtained by replacing in the previous equation $\log y$ and $\log(1-y)$ by their expectations, i.e., $\psi_0(Ls(x)u(x)) - \psi_0(Ls(x))$ and $\psi_0(Ls(x)(1-u(x))) - \psi_0(Ls(x))$, and rearranging terms, we get

$$
\begin{aligned}
- E_{y\sim\beta(y|Ls(x)u(x),Ls(x)(1-u(x))}\left[\frac{d}{dx^2}\beta(y|Ls(x)u(x),Ls(x)(1-u(x)))\right] = \\
L^2\left(s^2(x)\psi_1(L\,(1-u(x))\,s(x))\left(\frac{d}{dx}u(x)\right)^2 + s^2(x)\psi_1(Ls(x)u(x))\left(\frac{d}{dx}u(x)\right)^2 + \right. \\
2s(x)u(x)\psi_1(L\,(1-u(x))\,s(x))\frac{d}{dx}s(x)\frac{d}{dx}u(x) + 2s(x)u(x)\psi_1(Ls(x)u(x))\frac{d}{dx}s(x)\frac{d}{dx}u(x) - \\
2s(x)\psi_1(L\,(1-u(x))\,s(x))\frac{d}{dx}s(x)\frac{d}{dx}u(x) + u^2(x)\psi_1(L\,(1-u(x))\,s(x))\left(\frac{d}{dx}s(x)\right)^2 + \\
u^2(x)\psi_1(Ls(x)u(x))\left(\frac{d}{dx}s(x)\right)^2 - 2u(x)\psi_1(L\,(1-u(x))\,s(x))\left(\frac{d}{dx}s(x)\right)^2 - \\
\left.\psi_1(Ls(x))\left(\frac{d}{dx}s(x)\right)^2 + \psi_1(L\,(1-u(x))\,s(x))\left(\frac{d}{dx}s(x)\right)^2\right) . \quad (51)
\end{aligned}
$$

Finally, for the univariate case, since $\psi_1(z) = O(1/z)$ for $z > 0$, the Fisher information of a single Beta and, thus, $|I_{Y|f}(x)|$ are $O(L)$.

For the multivariate case $D > 1$, Eskenazis and Gavalakis (2024, Prop. 9) extends the convexity of the Fisher information to the matrix case, i.e. $I(\pi p_1 + (1-\pi)p_2) \preceq \pi I(p_1) + (1-\pi)I(p_2)$, where $\preceq$ denotes Loewner order, which implies, by Siotani (1967, Lemma 2.2), $|I(\pi p_1 + (1-\pi)p_2)| \leq |\pi I(p_1) + (1-\pi)I(p_2)|$. Then, it suffices to calculate the elements of the Fisher information matrix (FIM) of a single Beta by taking, in turn, the partial derivative with respect to each variable of the corresponding pair. This results in an expression analogous to Eq. 50, i.e. a sum with some terms including a factor $\log y$ or $\log(1-y)$, which are then replaced by the expectation to obtain:

$$
\begin{aligned}
- E_{y\sim\beta(y|Ls(\mathbf{x})u(\mathbf{x}),Ls(\mathbf{x})(1-u(\mathbf{x}))}\left[\frac{\partial}{\partial x_i \partial x_j}Beta(y|Ls(\mathbf{x})u(\mathbf{x}),Ls(\mathbf{x})(1-u(\mathbf{x})))\right] = \\
L^2\left(s^2(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_i}u(\mathbf{x})\frac{\partial}{\partial x_j}u(\mathbf{x}) + s^2(\mathbf{x})\psi_1(Ls(\mathbf{x})u(\mathbf{x}))\frac{\partial}{\partial x_i}u(\mathbf{x})\frac{\partial}{\partial x_j}u(\mathbf{x})\right. \\
+ s(\mathbf{x})u(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}u(\mathbf{x}) + s(\mathbf{x})u(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_j}s(\mathbf{x})\frac{\partial}{\partial x_i}u(\mathbf{x}) \\
+ s(\mathbf{x})u(\mathbf{x})\psi_1(Ls(\mathbf{x})u(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}u(\mathbf{x}) + s(\mathbf{x})u(\mathbf{x})\psi_1(Ls(\mathbf{x})u(\mathbf{x}))\frac{\partial}{\partial x_j}s(\mathbf{x})\frac{\partial}{\partial x_i}u(\mathbf{x}) \\
- s(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}u(\mathbf{x}) - s(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_j}s(\mathbf{x})\frac{\partial}{\partial x_i}u(\mathbf{x}) \\
+ u^2(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}s(\mathbf{x}) + u^2(\mathbf{x})\psi_1(Ls(\mathbf{x})u(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}s(\mathbf{x}) \\
- 2u(\mathbf{x})\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}s(\mathbf{x}) - \psi_1(Ls(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}s(\mathbf{x}) \\
\left.+\psi_1(L\,(1-u(\mathbf{x}))\,s(\mathbf{x}))\frac{\partial}{\partial x_i}s(\mathbf{x})\frac{\partial}{\partial x_j}s(\mathbf{x})\right) . \quad (52)
\end{aligned}
$$

Therefore, each element of the Beta FIM is $O(L)$. Then, Leibniz formula implies that the determinant of the desired FIM is $O(L^D)$, from which the result follows. □

C IMPLEMENTATION OF THE COPULA

In order to compute the log likelihood in the multivariate case, we need the log of copula density that is

$$\log c^G_{\mathbf{R}}(\mathbf{v}) = -\frac{1}{2}\left(\log\det\mathbf{R} + \mathbf{u}^\intercal\left(\mathbf{R}^{-1} - \mathbf{I}\right)\mathbf{u}\right) \tag{53}$$

where $\mathbf{u} = \left(\Phi^{-1}(v_1), \ldots, \Phi^{-1}(v_d)\right)^\intercal$ and Φ^{-1} is the inverse cumulative distribution function of the standard normal. Given the Cholesky decomposition $\mathbf{R} = \mathbf{L}\mathbf{L}^\intercal$, we have

$$\log\det\mathbf{R} = 2\sum_i \log L_{ii} \tag{54}$$

and

$$\mathbf{u}^\intercal\left(\mathbf{R}^{-1} - \mathbf{I}\right)\mathbf{u} = \|\mathbf{L}^{-1}\mathbf{u}\|^2 - \mathbf{u}^\intercal\mathbf{u}\,. \tag{55}$$

Note that the inverse of the lower triangular matrix $\mathbf{L}$ is numerically stabler and faster to compute than the one of $\mathbf{R}$.

For numerical stability, the vector $\mathbf{u} = \left(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_d)\right)^\intercal$ is computed with R's qnorm5 algorithm that we translated to Python and JAX, whose arguments are given in log form. These are obtained from the log CDF of the Beta distribution, where the integral is approximated using the rectangle method and then the logsumexp trick is used.

To generate the matrix $\mathbf{R}$ directly in the Cholesky space we use the C-vine method that uses Beta samples that we obtain from the $\hat{\mathbf{z}}$ normal samples using Laplace's bridge (Hobbhahn et al., 2022). Our implementation is based on Numpyro's (Phan et al., 2019; Bingham et al., 2019) class LKJCholesky, which is based on the algorithm of Lewandowski et al. (2009, Sec. 2.4).

```
from numpyro.distributions.util import (matrix_to_tril_vec,
                                        signed_stick_breaking_tril)

def to_LKJCholesky_cvine(normal_sample, dimension, concentration=1.):

    Dm1 = dimension - 1
    marginal_concentration = concentration + 0.5 * (dimension - 2)
    offset = 0.5 * jnp.arange(Dm1)

    offset_tril = matrix_to_tril_vec(jnp.broadcast_to(offset, (Dm1, Dm1)))
    beta_concentration = (
        jnp.expand_dims(marginal_concentration, axis=-1) - offset_tril
    )

    mu = beta_t_logit_mu(beta_concentration,beta_concentration)
    sigma = jnp.sqrt(beta_t_logit_var(beta_concentration,beta_concentration))
    beta_sample = logistic_transform(normal_sample[:3] * sigma + mu)

    partial_correlation = 2 * beta_sample - 1
    return signed_stick_breaking_tril(partial_correlation)

def beta_t_logit_mu(a,b):
    return(jnp.log(a/b))

def beta_t_logit_var(a,b):
    return((a+b)/(a*b))

def logistic_transform(logit):
    return(1 / (1 + jnp.exp(-logit)))
```

D EXPERIMENTS

D.1 Hyperparameters for UCI experiments

Our architecture matches that of GMDN-* (1 hidden layer with 50 nodes and $3N$ outputs) except for the final transformations and the activation function (erf in our case). Prior parameters: $\sigma^2_{\mathbf{W}_1} = \sigma^2_{\mathbf{w}_2} = \sigma^2_{\mathbf{b}_1} = \sigma^2_{b_2} = 1$, $\alpha_k = 1$ and $\lambda^{-1} \in \{1, 100, 10000\}$. We used SGLD-Adam from the SGMCMCJax library (Coullon et al., 2021; Coullon and Nemeth, 2022; Coullon et al., 2021; Kim et al., 2020) with the following parameters:

- batch size: 256
- step size: 0.003
- number of chains: 4
- warm-up iterations per chain: 1e5
- thinning (keep every): 100
- number of samples used for testing: 1000

D.2 Hyperparameters for NYC Taxi dataset

Hyperparameters We performed an extensive hyperparameter tuning using Bayesian optimization (number of runs=512) on a validation set over the following values (the best value found is shown in bold):

- $\lambda^{-1} \in [1, 10000]$: **960.01**
- $\sigma \in [0.001, 10]$: **9.51**
- SGLD step size $\in [0.0001, 0.01]$: **0.0038**
- $N \in [1, 100]$: **83**
- number of components of the unconditional gaussian mixture $\in [1, 100]$: **74**
- activation $\in$ [ReLU,erf]: **ReLU**
- hidden layers $\in [1, 3]$: **1**
- number of nodes $\in [2, 512]$: **339**

For this experiment, we used the Blackjax's library (Cabezas et al., 2024) for the SGLD algorithm (Welling and Teh, 2011).

The others hyperparameters were:

- batch size: 80
- number of chains: 4
- warm-up iterations per chain: 50000
- thinning (keep every): 10
- number of samples used for testing: 400
- concentration parameter for the LKJ distribution $\nu = 1$
- number of rectangles for the Beta log CDF approximation: 500

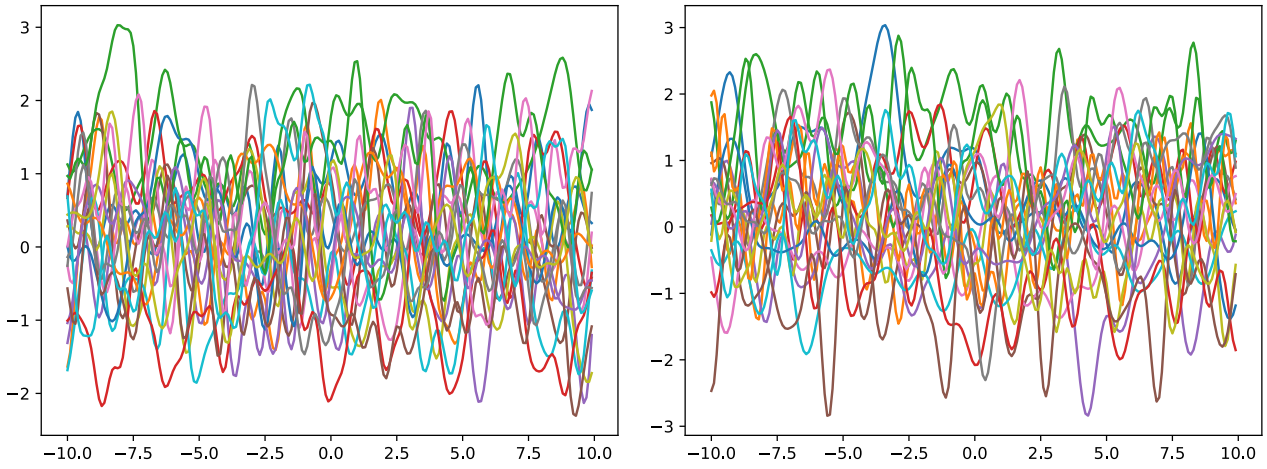
D.3 Experiments with the RBF kernel on UCI datasets

In this experiment, we used the neural tangent parameterization given by the Neural Tangents library with the "standard" parameterization. We fixed $\lambda^{-1} = 10000$ and vary $\sigma \in [1, 2, 4, 8]$ to study how the functional complexity affects the results.

Fig. 9 shows samples of $\hat{f}$ for the different σ values, illustrating the different levels of functional complexity considered. Fig. 10 shows how increasing the functional complexity affects the results. In particular, the energy, power and protein datasets benefit from an increased functional complexity, outperforming all the contenders.

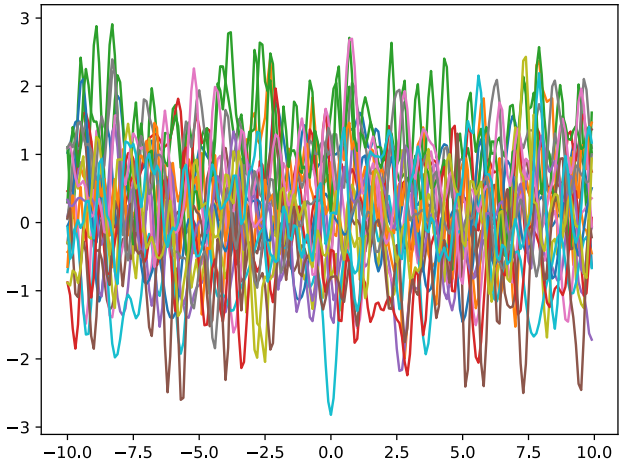
D.4 UCI benchmark : prior calibration

Figure 11 shows the predictive distributions sampled from the prior (in red) and their average (in blue) that closely matches the given unconditional distribution (in green) for BMDN-λ^{-1} (with erf activation). Notice that,

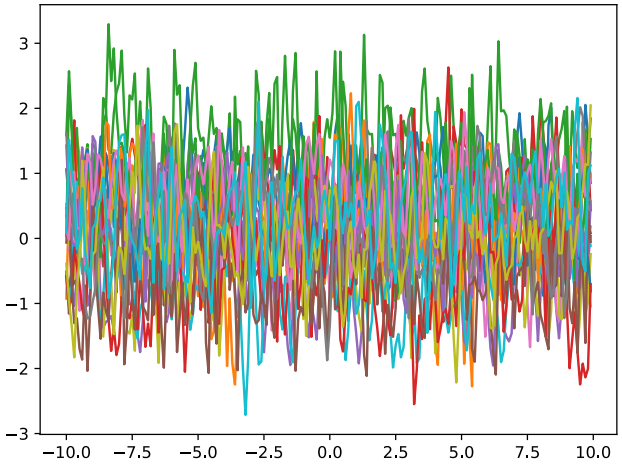


(a) $\sigma = 1$

(b) $\sigma = 2$



(c) $\sigma = 4$



(d) $\sigma = 8$

Figure 9: $\hat{f}$ samples.

as increases λ^{-1}, the sampled predictive distributions become more concentrated but their average remains close to the given unconditional distribution. For comparison, we also show the predictive distributions sampled from GMDN using the same σ, which naturally lack calibration and can be far from the unconditional distribution.

D.5 Multivariate NYC Taxi dataset: advantage of the copula

Figure 13 illustrates the advantage in expressiveness of the copula structure as the number of components N increases. We use the same hyperparameters in all the cases (2 hidden layers with 32 nodes, ReLU activation, $\sigma = 1$, $\lambda^{-1} = 1000$) and, to reduce as much as possible the effect of the diffeomorphism, we use only one Gaussian component for the unconditional. In comparison to the case of an identity correlation matrix with the same number of components N (denoted with the prefix "nocopula" in the figure), we observe that the copula structure allows to capture more complex patterns (e.g. it is able to represent the shape of Central Park where no taxi drop-offs occur), resulting in a better test log likelihood.

D.6 Synthetic experiment : independent control of functional and distributional complexity

In Figure 12, we consider a synthetic dataset, sampled from a uniform distribution whose support is defined by the image of the letter β. On each plot, the abscissa corresponds the conditioning variable x and the ordinate to the target y. In the top left corner, there is a plot of the $n = 500$ samples. We use the RBF sinusoidal

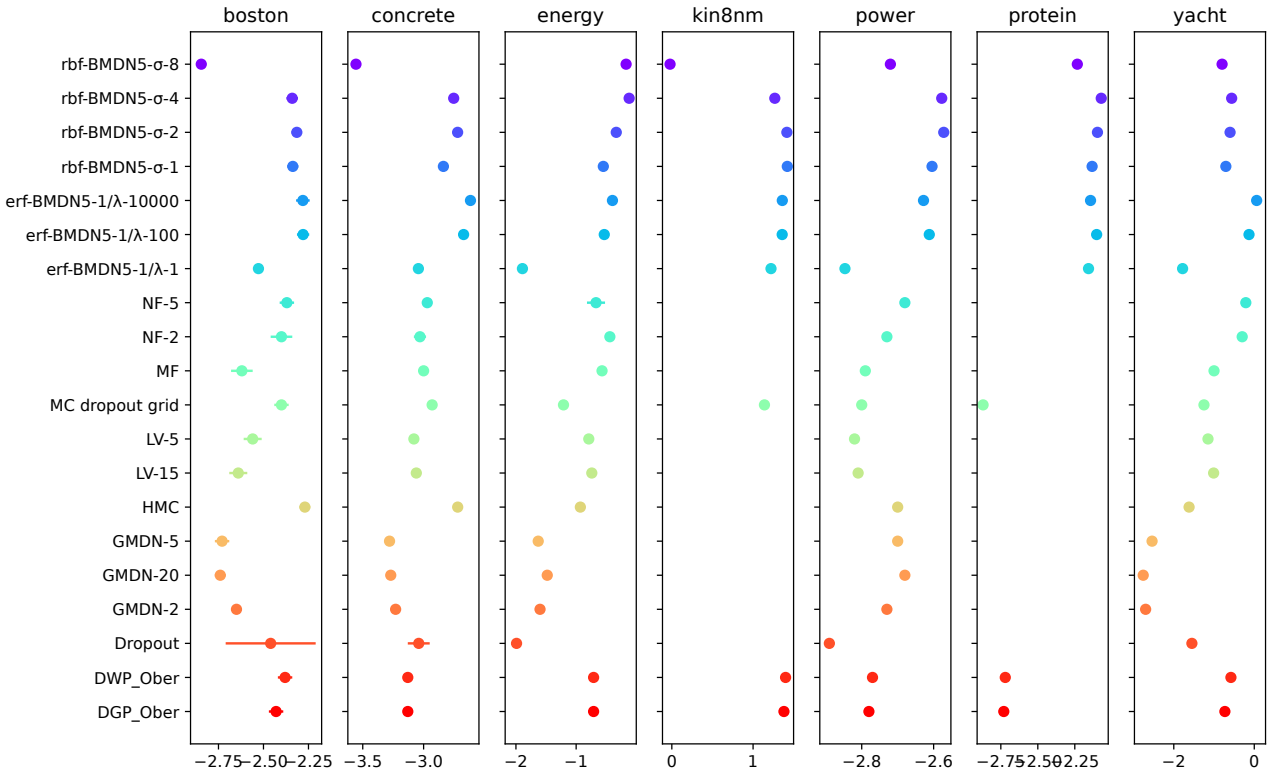


Figure 10: Results on UCI datasets in terms of log likelihood of the test set. Error bar represents std err.

activation, which allows independent control of the functional and distributional complexities as shown in Section 4.2. When going down the rows, σ increases and the prior curves exhibit quicker variations (higher functional complexity). When going to the right across the columns, λ^{-1} increases and the prior curves become "thinner", i.e. distributions are more concentrated (higher distributional complexity). In the resulting posteriors, we observe that this independent control allows to represent the small loop in the center of the letter β (see, e.g., the case of $\sigma = 1$ and $\lambda^{-1} = 1e4$). We compare to Gaussian MDN (shown in the first column), which exhibits tied distributional and functional complexities (as noted in Trippe and Turner (2018)), i.e., increasing σ favors both higher distributional and functional complexity (curves become both thinner and wilder). In all the cases, NUTS with windows adaptation (from the blackjax library) is used for sampling and all the experiments used the same hyperparameters except those that vary (specified in the figure).

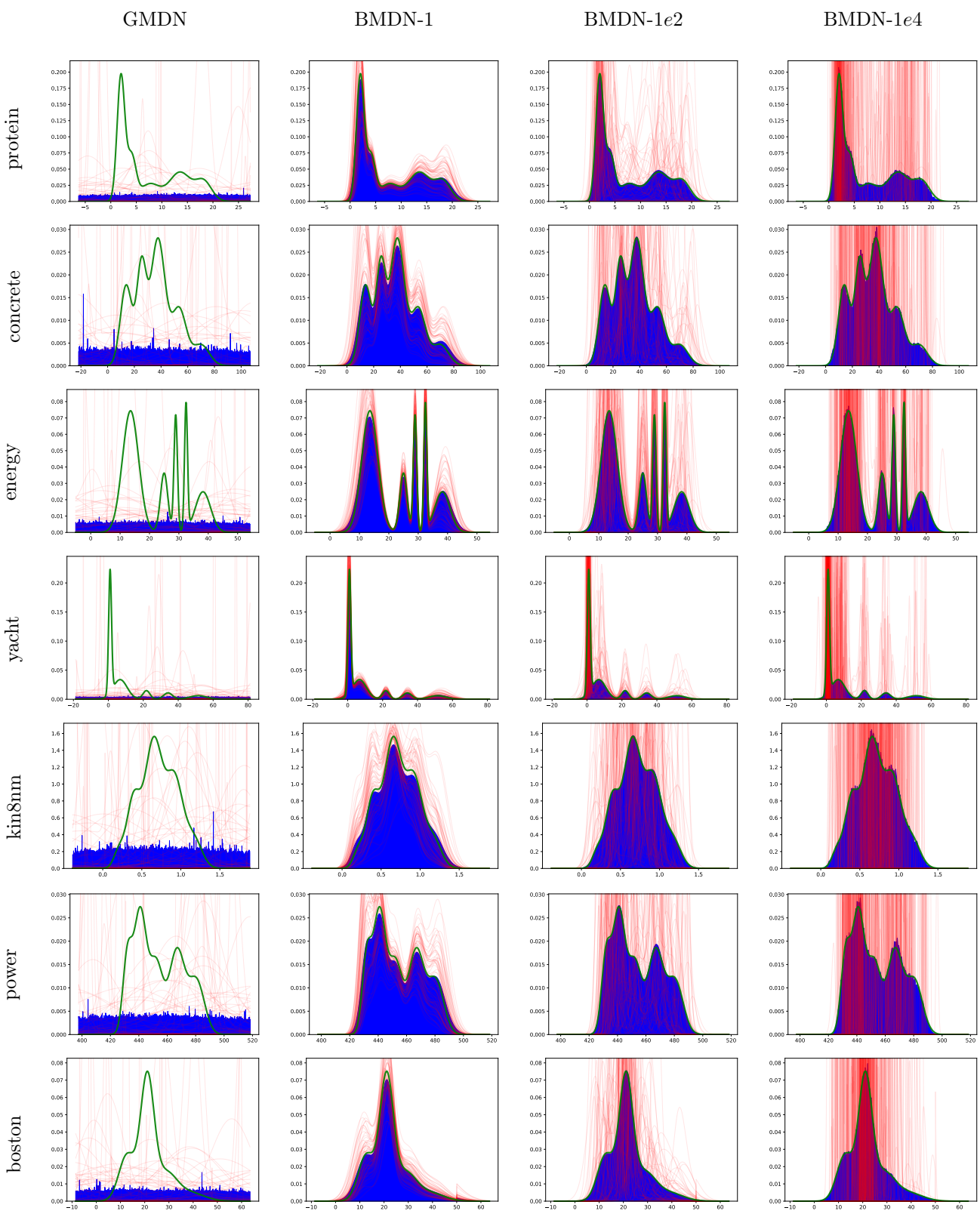


Figure 11: Sampled prior predictive distributions (red) and their average (blue) for sampled input values from the training set, compared to the given unconditional distribution (green), for the experiment of Sec 6.1. Both the unconditional and the conditional distributions are modeled with 5 components.

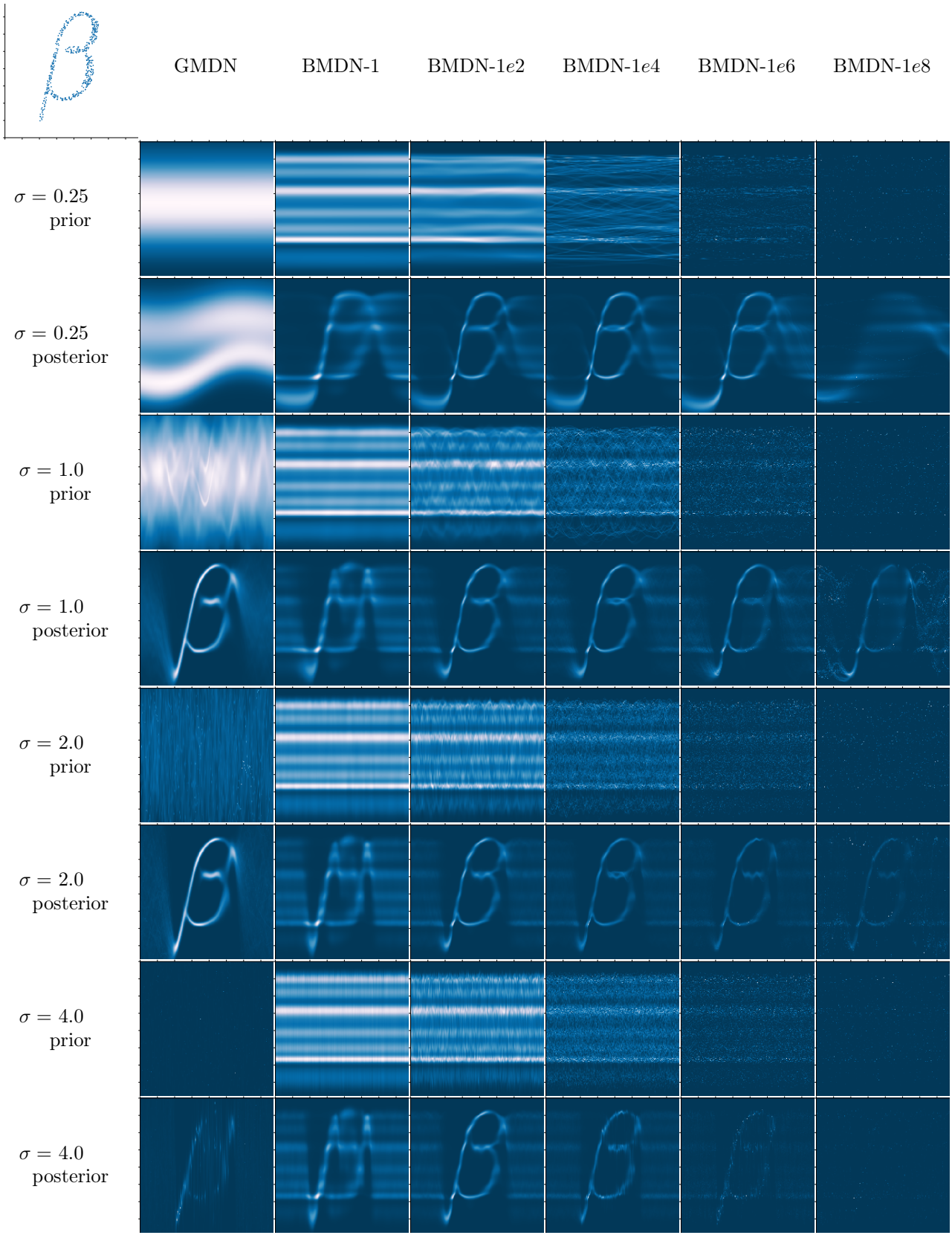


Figure 12: Synthetic experiment showing independent control of functional and distributional complexity.

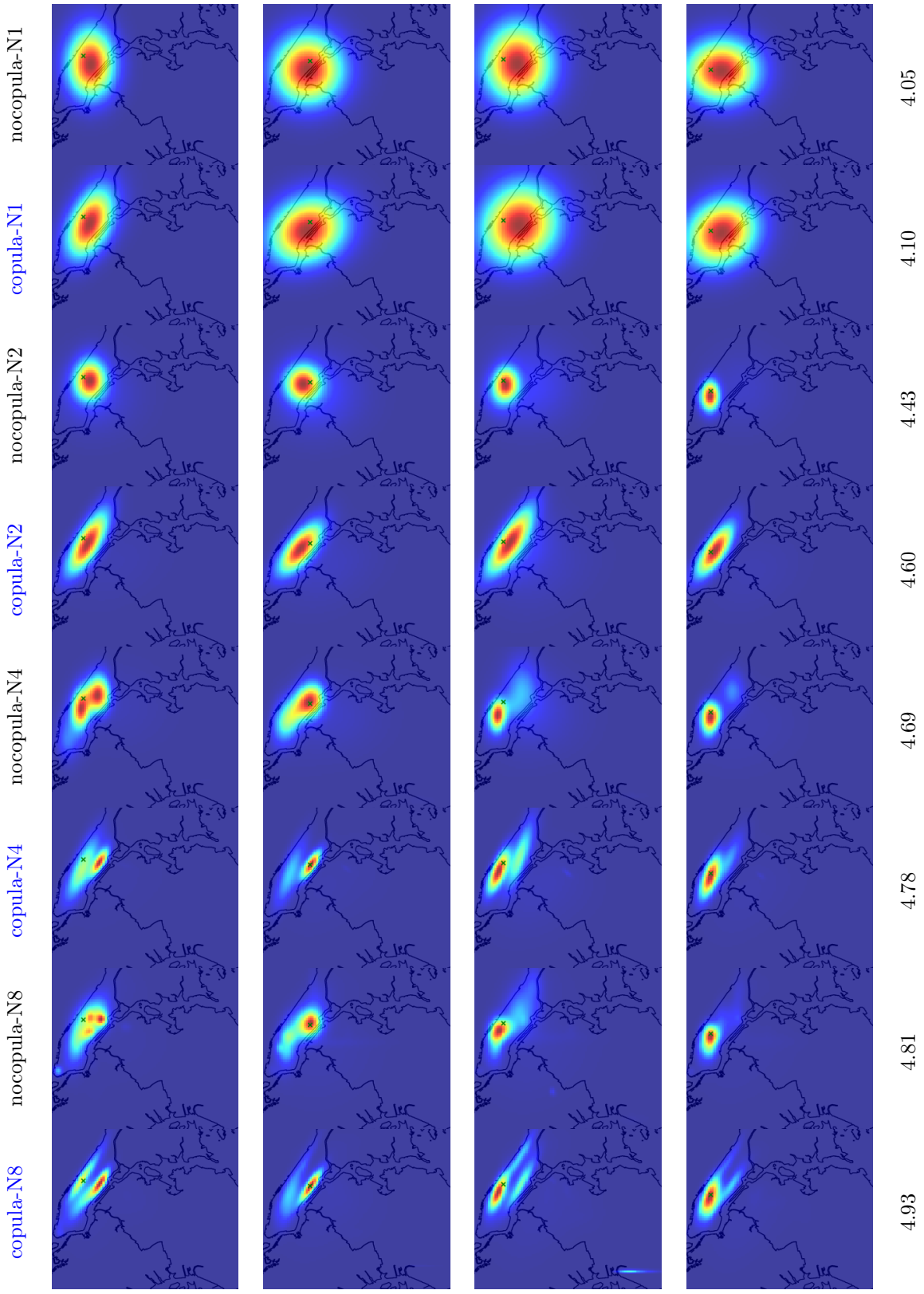


Figure 13: Predictions given by BMDN on test examples of the Taxi dataset (each column corresponds to a different conditioning $\mathbf{x}$, for different number of components N, with and without copula correlation. Test set log likelihood is displayed on the right.