Eur. Phys. J. C (2021) 81:662
https://doi.org/10.1140/epjc/s10052-021-09469-y

THE EUROPEAN PHYSICAL JOURNAL C

Special Article - Tools for Experiment and Theory

Mixture density network estimation of continuous variable maximum likelihood using discrete training samples

Charles Burton[1,a], **Spencer Stubbs**[1,2,b], **Peter Onyisi**[1,c]

[1] Department of Physics, University of Texas, Austin, TX, USA
[2] *Present address*: Physics Department, Rutgers University, New Brunswick, NJ, USA

Received: 26 March 2021 / Accepted: 20 July 2021 / Published online: 28 July 2021

Abstract Mixture density networks (MDNs) can be used to generate posterior density functions of model parameters $\boldsymbol{\theta}$ given a set of observables $\mathbf{x}$. In some applications, training data are available only for discrete values of a continuous parameter $\boldsymbol{\theta}$. In such situations, a number of performance-limiting issues arise which can result in biased estimates. We demonstrate the usage of MDNs for parameter estimation, discuss the origins of the biases, and propose a corrective method for each issue.

1 Introduction

A frequent goal in the analysis of particle physics data is the estimation of an underlying physical parameter from observed data, in situations where the parameter is not itself observable, but its value alters the distribution of observables. One typical approach is to use maximum likelihood estimation (MLE) to extract values of the underlying parameters and their statistical uncertainties from experimental distributions in the observed data. In order to do this, a statistical model $p\left(\mathbf{x}|\boldsymbol{\theta}\right)$ of the observable(s), as a function of the underlying parameters, must be available. These are frequently available only from Monte Carlo simulation, not from analytic predictions. In typical usage, the value of a parameter is varied across a range of possible values; the derived models (determined from histograms or other kernel density estimators, or approximated with analytic functions) are then compared to the distributions in the observed data to estimate the parameter.

A number of methods to perform this type of inference have been discussed in the literature. See, for example, Refs. [1–6]. Some of these also use machine-learning approaches, and many support the use of multiple observables in order to improve statistical power.

If one has a complete statistical model $p\left(\mathbf{x}|\boldsymbol{\theta}\right)$ for the observables available for any given value of the parameter, the MLE can be computed. Unfortunately, this is usually difficult to determine analytically, especially if there are multiple observables with correlations, detector effects, or other complications. An alternative procedure is to directly approximate the likelihood function of the parameter, $\mathcal{L}\left(\boldsymbol{\theta}|\mathbf{x}\right)$.

Mixture density networks [7] solve a closely-related task. They are used to approximate a posterior density function $p\left(\boldsymbol{\theta}|\mathbf{x}\right)$ of parameters $\boldsymbol{\theta}$ from input features $\mathbf{x}$ as a sum of basis probability density functions (PDFs). More specifically, the neural network predicts the coefficients/parameters of the posterior density. With Bayes' theorem, we will relate the posterior density, which is output by the network, to the desired parameter likelihood function. Notably, because of the flexibility of network structure, MDNs permit the straightforward use of multidimensional observables, as well as approximating the posterior density function with any desired set of basis PDFs.

When training MDNs, one typically assumes that all input parameter values are equally likely to be sampled in the training dataset, and that the parameter is continuous. In essence, this is equivalent to specifying a flat prior for the application of Bayes' theorem that translates the posterior density that the MDN learns into the likelihood function.

However, such datasets may not be readily available for various reasons: one may share Monte Carlo samples with analyses using other estimation techniques which use histograms at specific parameter values to build up templates, or there may be computational difficulties with changing the parameter values in the Monte Carlo generator for every event. For example, in a top quark mass measurement, Monte Carlo event generators do not efficiently support the case of simulating events along a continuum of possible top mass values. Rather, events are generated where the top mass has

[a] e-mail: burton@utexas.edu (corresponding author)
[b] e-mail: f.spencer.stubbs@gmail.com
[c] e-mail: ponyisi@utexas.edu

been set to one value on a grid of possible parameter values. In this work, we discuss issues which arise when using training samples with discrete parameter values, rather than a continuous parameter distribution, to estimate parameters with MDNs.

A related issue arises when a restricted range of parameter values is used in the training, in which the trained network is reluctant to predict values near the boundaries of the range due to the relative lack of training events in that vicinity. This occurs even when training with a continuous parameter distribution and affects tasks other than likelihood estimation, such as simple regression. Since this issue will appear in any practical application of an MDN to estimate a physical parameter, we will discuss it.

The aim of the paper is to demonstrate the construction of MDNs for estimating continuous parameters from datasets populated only with discrete parameter values, and to discuss pitfalls that can occur in the training. This paper is structured as follows. The basic concept of mixture density networks is introduced and the application to likelihood estimation is discussed. Potential biases in the network training are explained, and procedures to mitigate them are proposed, in the context of specific examples. The performance of trained MDNs is demonstrated. Finally, limitations of the technique and avenues for future improvement are discussed.

2 Mixture density networks as likelihood approximators

A mixture density network is a neural network where the output is a set of parameters of a *function*, defined to be a properly normalized PDF. The outputs of the MDN can be, for example, the mean, width, and relative contributions of a (fixed) number of Gaussian distributions. But, generally speaking, the goal of MDN usage is to obtain an estimate of a posterior density function $p(\boldsymbol{\theta}|\mathbf{x})$ of a data set that includes parameters $\boldsymbol{\theta}$ and observations $\mathbf{x}$.

The MDN represents the target posterior density with a weighted sum of n generic basis PDF functions $\mathcal{B}_n$,

$$\tilde{p}(\boldsymbol{\theta}|\mathbf{x};\mathbf{Z}) = \sum_{i=1}^{n} c_i(\mathbf{x}) \cdot \mathcal{B}_i(\boldsymbol{\theta}; \mathbf{z}_i(\mathbf{x})), \tag{1}$$

where $c_i(\mathbf{x})$ and $\mathbf{z}_i(\mathbf{x})$ are the $\mathbf{x}$-dependent coefficients and parameters of the basis functions, which are predicted by the network, and $\mathbf{Z} = \{c_i\} \cup \{\mathbf{z}_i\}$ is shorthand to represent all of the coefficients of the learned model. Typically, the conditions $c_i \in [0, 1]$ and $\sum_i c_i = 1$ are imposed, for example through the softmax function. In principle, the basis functions can be any set of basis PDFs. A useful choice for many applications is a mixture of n Gaussian functions. (Since we are estimating a posterior density function of a continuous parameter, we might expect a minimum in the negative logarithm of this function. The minimum would be quadratic to leading order, making Gaussians a natural choice for PDF basis functions.) For that choice, the neural network's output is a set of $3n - 1$ independent coefficients $\mathbf{Z}(\mathbf{x}) = \{c_i(\mathbf{x}), \mu_i(\mathbf{x}), \sigma_i(\mathbf{x})\}$ (with one softmax normalization condition).

To train the network, in each epoch, the output function of the network $\tilde{p}(\boldsymbol{\theta}|\mathbf{x};\mathbf{Z})$ is evaluated for each point in the training data set $\left\{\left(\boldsymbol{\theta}_j, \mathbf{x}_j\right)\right\}$. The cost function,

$$\mathcal{C}(\mathbf{Z}) = -\log\left[\prod_{j=0}^{m} \tilde{p}\left(\boldsymbol{\theta}_j|\mathbf{x}_j;\mathbf{Z}\left(\mathbf{x}_j\right)\right)\right], \tag{2}$$

is the negative logarithm of the product of these values. The error is backpropagated through the network in the standard way, and the cost is minimized to determine the ideal coefficients,

$$\hat{\mathbf{Z}} = \arg\max_{\mathbf{Z}}\{\mathcal{C}(\mathbf{Z})\}. \tag{3}$$

From the physics standpoint of parameter estimation with discretized data, we are not actually interested in using the network to model the *true* posterior density, as one might typically do in MDN applications. (With discrete training samples, the true posterior of the training data set involves delta functions at the various values of $\boldsymbol{\theta}$ where each template lies.) Instead, we convert $\tilde{p}\left(\boldsymbol{\theta}|\mathbf{x};\hat{\mathbf{Z}}\right)$ created by the MDN into an estimate of the likelihood function $\mathcal{L}(\boldsymbol{\theta}|\mathbf{x})$. Using Bayes' theorem,

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = \tilde{p}\left(\mathbf{x}|\boldsymbol{\theta};\hat{\mathbf{Z}}\right) = \frac{\tilde{p}\left(\boldsymbol{\theta}|\mathbf{x};\hat{\mathbf{Z}}\right)p(\mathbf{x})}{p(\boldsymbol{\theta})}. \tag{4}$$

Notably, in the mindset of estimating some parameters $\boldsymbol{\theta}$, as long we ensure a flat prior $p(\boldsymbol{\theta})$, the likelihood that we seek and the posterior density which is output by the trained MDN differ only by an irrelevant multiplicative factor – the prior $p(\mathbf{x})$, which is determined by the entire training set, and does not depend on $\boldsymbol{\theta}$.

In order for the MDN to effectively *interpolate*, it is important that there not be too much freedom in the MDN output. For example, suppose there were an equal number of training templates and Gaussian components in $\tilde{p}(\boldsymbol{\theta}|\mathbf{x})$. The network could then essentially collapse the Gaussian functions to delta functions at each template $\boldsymbol{\theta}$ value and reduce the cost function without limit. As long as the observed values $\mathbf{x}$ can reasonably be produced by multiple values of $\boldsymbol{\theta}$, and the number of basis functions in the MDN is kept reasonably low, the MDN will naturally be forced to interpolate between parameter points, as desired for estimating $\mathcal{L}(\boldsymbol{\theta}|\mathbf{x})$.

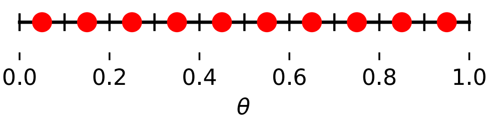

Fig. 1 For some applications, training data are only available at discrete values of θ (e.g. at each of the red markers). For every example discussed here, the training samples consist of 10 template data sets with parameter θ equally spaced between 0 and 1

3 Density of parameter points

The application of Bayes' theorem in Eq. (4) involves $p(\boldsymbol{\theta})$, and simplifies if $p(\boldsymbol{\theta})$ can be assumed to be flat. To ensure this condition, the locations of the templates in parameter space should be chosen to ensure an equal density of training points across the entire range.

This first requires that the templates must be equally distributed in the parameter space. Otherwise, the density of training points would be non-constant, implying a non-flat $p(\boldsymbol{\theta})$. This would bias the network.

Secondly, this necessitates that the parameter range extend slightly outside of the range of the templates in parameter space. For example, suppose we have one parameter θ and the parameter range is selected as $\theta \in [0, 1]$, and 10 templates are to be used. To ensure equal and unbiased coverage of the parameter range, they should be placed at $\theta = 0.05, 0.15, \ldots, 0.95$, as shown in Fig. 1. If the extra gaps (from 0 to 0.05 and from 0.95 to 1) are not included at the edges (for example, if eleven templates were used, at $\theta = 0, 0.1, \ldots, 1$), then the density of training data is higher at the extremal values of θ, again creating a non-flat $p(\theta)$ and a bias in the training.

Essentially, the templates' parameter values should lie at the centers of equal-width histogram bins that extend from the lowest to highest values of θ. Note that, in Fig. 1, each θ bin has an equal amount of training data. This condition can be generalized to multidimensional parameter spaces.

4 PDF normalization

Another issue arises with respect to the normalization of the MDN output. The MDN's output posterior $\tilde{p}(\boldsymbol{\theta}|\mathbf{x}; \mathbf{Z})$ might not be constructed with any knowledge of the range of parameters $\boldsymbol{\theta}$ in the training set. For example, Gaussian basis functions have infinite support and therefore will always predict a non-zero posterior density outside the range of the training data. Proper training of an MDN requires that the output posterior density be properly normalized across the selected range of $\boldsymbol{\theta}$ for MDN training to work properly. If this is not done, parameter values near the edges of the range will be penalized because the posterior predicts parameter values to occur outside of the range, and these are never encountered in the training data.

For a one-dimensional parameter $\theta \in [\theta_{\min}, \theta_{\max}]$, one must require that

$$\int_{\theta_{\min}}^{\theta_{\max}} p(\theta|\mathbf{x})\, d\theta = 1.$$

This constraint can be achieved by dividing MDN's predicted posterior density by the integral of the posterior density over the parameter range. Since, during training, the posterior density is never evaluated outside of this range, this results in the proper normalization.

Practically speaking, it is easier to apply this renormalization procedure for each function $\mathcal{B}_i$ than to do it on the sum. This has the benefit that the condition $\sum_i c_i = 1$ is still valid. In the one-dimensional parameter case, if the cumulative distribution function (CDF) is available,

$$\mathcal{B}_i(\theta; \mathbf{z}_i) = \frac{\mathrm{PDF}(\theta; \mathbf{z}_i)}{\mathrm{CDF}(\theta_{\max}; \mathbf{z}_i) - \mathrm{CDF}(\theta_{\min}; \mathbf{z}_i)}.$$

This effect is not specific to training with discrete parameter choices, and will generally occur in regions where observed data could be compatible with parameters outside the training range.

5 Edge bias from training on templates

We now discuss a bias which arises from the discreteness of the input parameter values. As discussed in the previous sections, in order to achieve a flat $p(\boldsymbol{\theta})$, we need to consider the input range of parameters $\boldsymbol{\theta}$ to be broader than just the range where training data are located. However the cost function is only evaluated on the training data, and so the optimizer can "cheat" by overpredicting values of $\boldsymbol{\theta}$ that are compatible with a broad range of observations, while underpredicting extremal values of $\boldsymbol{\theta}$ that are not represented in the training data. The symptom of this is that the probability density $\tilde{p}(\mathbf{x}|\boldsymbol{\theta})$ implied by the MDN output posterior $\tilde{p}(\boldsymbol{\theta}|\mathbf{x})$,

$$\tilde{p}(\mathbf{x}|\boldsymbol{\theta}) = \frac{\tilde{p}(\boldsymbol{\theta}|\mathbf{x})\, p(\mathbf{x})}{p(\boldsymbol{\theta})},$$

when integrated over data $\mathbf{x}$, does not integrate to one for all values of $\boldsymbol{\theta}$ (as one would expect for a properly normalized density function). Rather, it is smaller than one at extremal values of $\boldsymbol{\theta}$ and greater than one in the interior of the range.

The size of this effect depends on how much the templates overlap. Figure 2 shows examples of distributions which will demonstrate negligible and extreme bias.

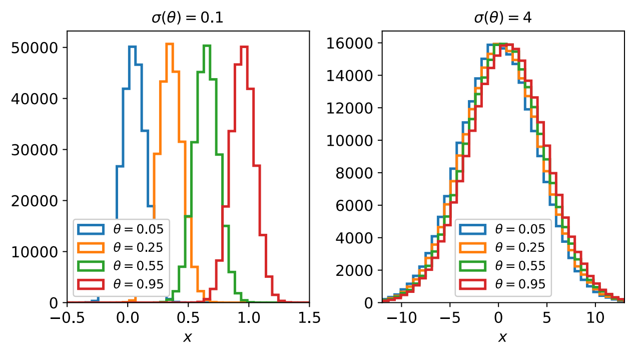

Fig. 2 Examples of templates (training data for discrete parameter values) of an observable x for different values of a parameter θ. When the observable templates are distinguishable (left), the edge bias (see text) is negligible. When the templates are harder to distinguish over the range of θ (right), the correction for the edge bias is critical

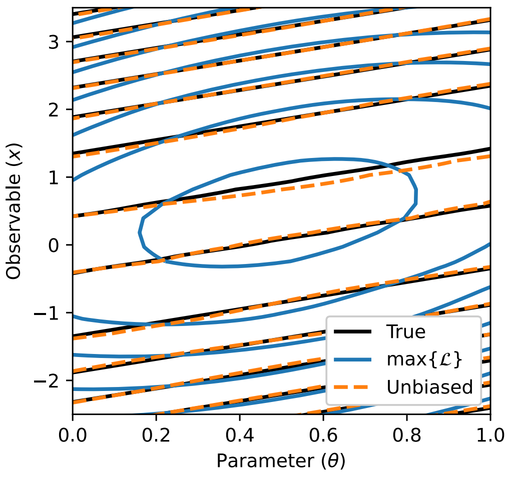

Fig. 3 Joint PDFs for the example described in Sect. 5.1. The "true" joint PDF is shown in black. Fitting to the training dataset with discrete parameter values results in the blue PDF, which is different from the desired form. Adding additional constraints to the fit requiring $\int p(x|\theta)\, dx$ to be 1 at $\theta = 0$ and $\theta = 1$ yields the orange dashed line PDF instead, which is a good approximation of the true distribution

5.1 Demonstration with functional fit

It should be emphasized that the observed edge bias is *not* unique to the mixture density network method. Rather, it is simply a result of minimizing the cost function Eq. (2) composed of the posterior density $p\left(\boldsymbol{\theta}|\mathbf{x}\right)$ for a finite number of templates. To illustrate this, we will show the existence and correction of the bias in a simple functional fit.

Consider a statistical model which, given some parameter value $\theta \in [0, 1]$, produces a Gaussian distribution of x, a univariate variable: this could correspond to a "true" value θ and an "observed" value x which is subject to resolution effects. For this example, we choose the distribution

$$p(x|\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x-\theta}{\sigma}\right)^2\right), \tag{5}$$

with $\sigma = 4$. We choose 10 equally-spaced values of θ between 0 and 1 at which to sample the model to generate training points. Since the width of each template σ is much broader than the total range of θ, the 10 templates are not very distinguishable from one another. A few of the templates are shown on the right-hand side of Fig. 2.

Next, we attempt to reconstruct the joint probability distribution $p(x, \theta)$ by performing an unbinned maximum likelihood fit of a three-parameter function to the generated data,

$$f\left(x, \theta; \mu_m, \mu_b, \sigma\right) = \mathcal{A} \exp\left(-\frac{1}{2}\left(\frac{x-(\mu_m\theta+\mu_b)}{\sigma}\right)^2\right),$$

where $\mathcal{A}$ is a normalization factor. The best-fit values of the function parameters are not the ones used to generate the data.

To understand the source of the problem, it is helpful to consider the joint probability density, which is visualized in Fig. 3. We know the "true" value we want to reproduce, which is the product of the distribution $p(x|\theta)$ in Eq. (5) with a flat prior $p(\theta) = 1$ on the range $\theta \in [0, 1]$. We can then overlay $\hat{f}(x, \theta)$ given by the best-fit values of μ_m, μ_b, and σ. We see that the overall shapes of the two are similar, but $\hat{f}(x, \theta)$ is more concentrated at intermediate values of θ. Integrating along vertical lines, the value $\int \hat{f}(x|\theta = 0)\, dx$ is clearly less than $\int \hat{f}(x|\theta = 0.5)\, dx$, while the values $\int p(x|\theta)\, dx$ are constant for all θ. While maintaining the *total probability* in the joint PDF constant, the fit to f has recognized that a better value of the likelihood can be achieved by removing probability from the regions near $\theta = 0$ and $\theta = 1$ (where there are no data points to enter the likelihood computation) and placing that likelihood near $\theta = 0.5$, a parameter value that is consistent with almost all values of x. In other words, the function $\hat{f}$ is not consistent with the presumption of a flat prior $p(\theta)$.

Having seen this, we can now propose a solution: we require the fit to minimize the cost in Eq. (2) while also satisfying the following constraint:

$$\int_{-\infty}^{\infty} p\left(\theta_j, x\right) dx = 1\, \forall\, \theta_j \in [0, 1]\,. \tag{6}$$

This constraint prevents the optimizer from improving the cost function by implicitly modifying $p(\theta)$. In practice, the constraint is applied for a few values of θ, for example $\theta = 0, 0.5, 1$, as this was seen to be enough to correct the bias. In this case, the smoothness of the functional forms forced the constraint to apply everywhere. Applying this constraint, we redo the fit. The best-fit parameters obtained – the bottom row of Table 1 – are now consistent with the true values used

Table 1 The maximum likelihood estimators for the parameters of the function f, when determined using discrete parameter inputs, do not match the intended model parameters due to edge effects. Adding a constraint during likelihood optimization resolves this, and the results are consistent with the desired values

	μ_m	μ_b	σ
True values	1	0	4
$\max\{\mathcal{L}\}$	0.17	0.41	1.65
Unbiased	1.03	−0.02	3.99

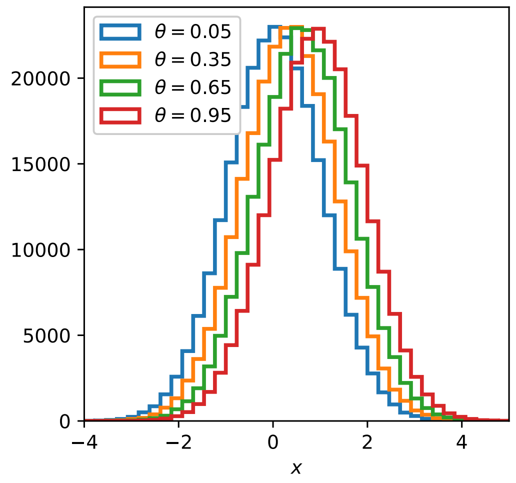

Fig. 4 Four of the ten template histograms used as training data for the single-input MDN of Sect. 5.2

to generate the data. Accordingly, the joint PDF of the constrained fit (in Fig. 3) shows no bias towards $\theta = 0.5$. Any remaining disagreement is attributable to statistical fluctuations in the estimation of $p(x)$ from the finite data set.

5.2 MDN with one observable

Now we will demonstrate how this correction is applied with a mixture density network. We construct a neural network using PyTorch [8] with a single input x, and outputs which are parameters of a function $\tilde{p}\,(\theta|x)$ which estimates the likelihood $\mathcal{L}\,(\theta|x)$. (See Sect. 2 for details.) There is a single hidden layer with an adjustable number of nodes. Because of the intentional simplicity of these data, the number of nodes in this hidden layer was generally kept between 2 and 5, and we model the posterior density with just a single Gaussian function. Therefore, the two output nodes of the network are just the mean and standard deviation of this function.[1]

A large number of x values were generated at each of 10 equally-spaced discrete values of θ from Gaussian distributions with $\mu = \theta$ and $\sigma = 1$ (see Fig. 4). The network is trained by minimizing the cost function described in Sect. 2. At each epoch, the cost is backpropagated through the network, and the Adam minimizer [9] is used to adjust the network parameters. We find that this produces biased results. Due to the construction of the training data, we expect that the MDN should predict that $\tilde{p}\,(\theta|x)$ is a truncated Gaussian (nonzero only on the range $\theta \in [0, 1]$) with $\mu = x$ and $\sigma = 1$. In Fig. 5, the MDN output from the naive training for input values of x are shown; we see that the network underestimates the width of the posterior density and biases the mean towards 0.5.

[1] To avoid ambiguity, it is common practice in MDN applications that the output node corresponding to the Gaussian's σ is actually $\log(\sigma)$, and the exponential of this node's value is used to calculate the loss. This action effectively forces $\sigma > 0$.

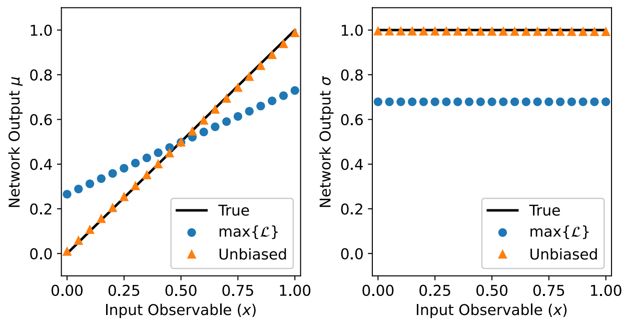

Fig. 5 Parameters of the posterior density functions predicted by the MDNs of Sect. 5.2 for various values of the input x. Without correction, the network will underestimate the width of the posterior density, and predict a maximum-likelihood θ value biased towards the middle of the sampled rang

5.2.1 1D network correction and validation

The cost function of the network is then modified by adding an extra term,

$$\mathcal{C} \rightarrow \mathcal{C} + \lambda \mathcal{S}, \tag{7}$$

where

$$\mathcal{S} = \text{std.}\left\{ \int_{-\infty}^{\infty} \tilde{p}\,(\theta_j, x)\, dx \right\}, \tag{8}$$

for some set of parameter values θ_j. With respect to Fig. 3, this extra term is the standard deviation of integrals of the joint probability $\tilde{p}\,(x, \theta)$ along vertical slices in θ. This term ensures that these integrals are all the same, thus ensuring that $p(x|\theta)$ remains constant for any value of θ, discouraging a bias towards intermediate values of θ. The joint probability,

$$\tilde{p}(\theta, x) = \tilde{p}(\theta|x)p(x),$$

is estimated from the product of the MDN output and the probability distribution over the training sample, which is estimated using histograms.

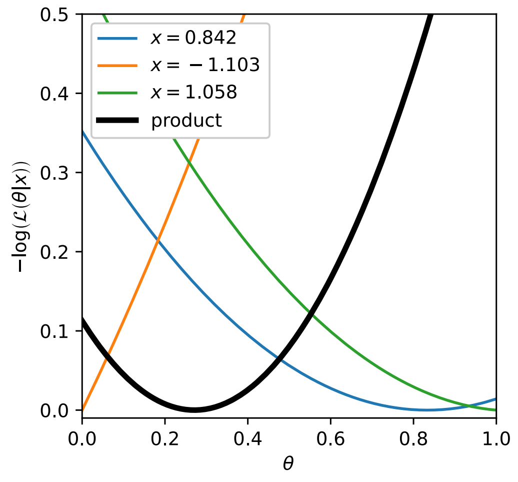

Fig. 6 For a fixed value of the parameter $\theta = 0.2$, three x_i values are generated. The posteriors $\tilde{p}_i(\theta|x_i)$ are obtained from the output of the trained MDN of Sect. 5.2. The maximum likelihood estimator θ_{ML} is extracted from the product of these posteriors according to Eq. (10) and shows a result statistically compatible with the expected value of 0.2. (For visualization, all functions are offset to have a minimum of 0)

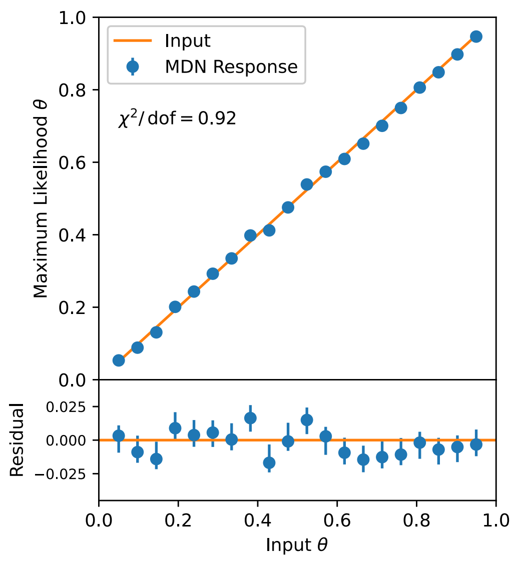

Fig. 7 Comparison of the true value of the parameter θ and the MLE estimate determined for a dataset generated at that θ over the range $\theta \in [0, 1]$. The errors shown are those obtained by applying Wilks' theorem to the posterior density. No bias is observed and the error estimates meet expectations

The new hyperparameter λ is tuned to balance the effectiveness of the additional term while avoiding the introduction of numerical instability into the minimization. In practice, its value during the training is initially set to zero, allowing the network to first loosely learn the posterior. Then, in a second step of the training, its value is increased so that the values of the two terms in Eq. 7 are approximately equal. With this new cost function, the outputs of the neural network – the coefficients of the Gaussian posterior density function – correctly match expectations, as seen in Fig. 5.

Next, we demonstrate the ability of the network to reconstruct the underlying parameter θ given a set of observations. Each measurement of the quantity x will be denoted x_i. For each x_i, the trained network outputs an estimate of the posterior function $\tilde{p}_i(\theta|x_i)$. To calculate the combined cost of an entire set of measurements, $\{x_i\}$, we take the negative logarithm of the product of the individual posteriors,

$$\mathcal{C}(\theta|\{x_i\}) = -\log\left[\prod_i \tilde{p}_i(\theta|x_i)\right]. \tag{9}$$

The maximum likelihood estimator is defined as

$$\theta_{\mathrm{ML}} = \underset{\theta\in[0,1]}{\arg\min}\{\mathcal{C}(\theta|\{x_i\})\}. \tag{10}$$

A likelihood-ratio test can be used to determine the statistical uncertainty of the estimator θ_{ML}. An example of the functions generated for this trained network is given in Fig. 6.

To test the accuracy and precision of the network, we choose 20 equally-spaced test θ_t values in the range [0, 1]. For each θ_t, we generate a set of 10,000 measurements $\{x_i\}$. The pseudodata is passed though the network for each θ_t, the posterior density functions calculated, and a θ_{ML} is found. The uncertainty in θ_{ML} is estimated using Wilks' theorem [10] (searching for values of θ that increase $\mathcal{C}$ by 0.5). If the network is accurate and unbiased, we should find statistical agreement between θ_t and θ_{ML}. Indeed, we found this to be true with $\chi^2/\mathrm{d.o.f.} = 0.92$. The results for the unbiased network are shown in Fig. 7.

5.3 Demonstration with 2D network

Next, we consider a more advanced usage of MDNs. We consider a system with two inputs, $\mathbf{x} = (x_1, x_2)$. The value of x_1 is sensitive to an underlying parameter θ, but only for small values of x_2; when x_2 is large, there is no sensitivity. This is chosen to demonstrate how this neural network technique can be an advantageous approach to parameter estimation: the MDN will automatically learn when observables offer meaningful information on parameters and when they do not.

5.3.1 2D toy model

The model used to generate 2D pseudodata comprises two components. The first is events with a small x_2 value. These events have an x_1 value which depends on the unknown parameter θ. In the toy model, this component is modeled by a Gaussian in x_1. The mean of this Gaussian varies with θ. In x_2, it is modeled by a Gaussian with a (lower) mean of 1.

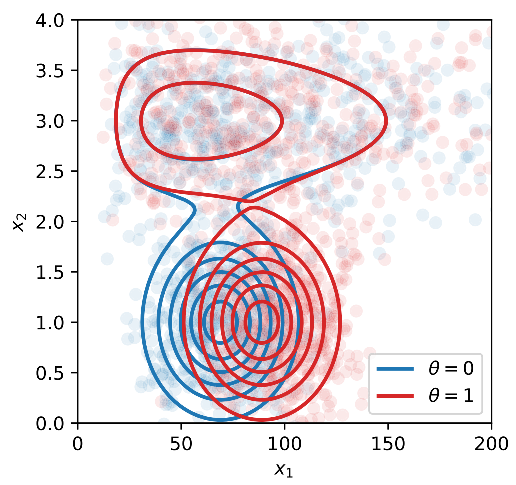

Fig. 8 Examples of PDFs (solid lines) and generated sample points (translucent circles) for different values of θ for the model of Sect. 5.3

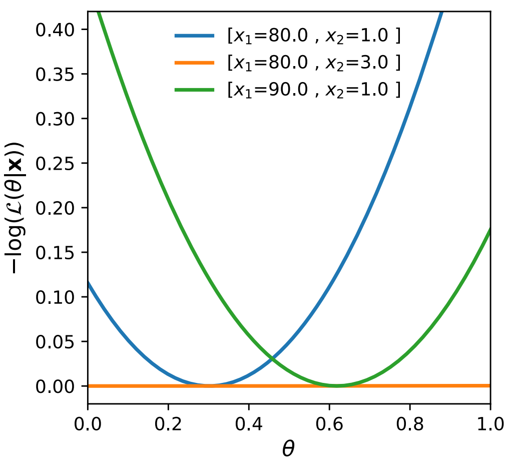

Fig. 9 Posterior densities predicted by the trained MDN for three possible observations in the model of Sect. 5.3. When x_2 is small (blue and green curves), the value of x_1 gives sensitivity to the model parameter θ. When x_2 is large (orange curve), there is very little sensitivity to θ and the posterior is flat. (For visualization, all functions are offset to have a minimum of 0)

In the other component, the value of x_2 is large, and the x_1 value is not related to the unknown parameter. This component is modeled with a Gamma distribution in x_1, and a Gaussian with a (higher) mean of 3 in x_2. The relative fraction of the two components is the final model parameter.

Written out, the 2D PDF takes the form

$$p\left(x_1, x_2|\theta\right) = f \cdot p_a\left(x_1, x_2|\theta\right) + (1-f) \cdot p_b\left(x_1, x_2\right),$$

where

$$p_a\left(x_1, x_2|\theta\right) = \mathcal{N}\left(x_1; \mu_{a1}\left(\theta\right), \sigma_{a1}\right) \cdot \mathcal{N}\left(x_2; \mu_{a2}, \sigma_{a2}\right)$$
$$p_b\left(x_1, x_2\right) = \Gamma\left(x_1; \mu_{b1}, \alpha, \beta\right) \cdot \mathcal{N}\left(x_2; \mu_{b2}, \sigma_{b2}\right).$$

This PDF is shown for the two extremal values of θ in Fig. 8.

5.3.2 2D network structure, correction, and validation

The network is built in the following manner. First, we note that there are two inputs. Accordingly, there will be two input nodes. Next, one should note from the contours of $p\left(x_1, x_2\right)$ in Fig. 8, for a sample $\mathbf{x}$ with large x_2, the value of x_1 has no predictive power of θ. Restated simply, the red and blue contours overlap entirely on the upper half of the plot. However, for small values of x_2, the x_1 variable can distinguish different values of θ; the red and blue contours are separated on the lower half of the plot. Two hidden layers are used. The output of the network is the coefficients of a mixture of Gaussian distributions. The number of Gaussians in the mixture is a hyperparameter which may be tuned to best match the particular structure of some data. In this example, the data are bimodal, so it is reasonable to expect two Gaussians to be expected. It was confirmed by trial-and-error that a mixture of two Gaussians was adequate to model these data, since it produced the correct linear response in the validation of the network. Therefore, for this network, there are six output nodes (with one normalization constraint) for a total of five independent values. The correction term presented in Eq. (7) is applied as well. To do this, $p(\mathbf{x})$ is estimated using a 2D histogram. It is then multiplied by the MDN output posterior $\tilde{p}\left(\theta|\mathbf{x}\right)$ to obtain the joint probability $\tilde{p}\left(\theta, \mathbf{x}\right)$. Finally, the integrals in Eq. (8) are calculated. The training data consist of 100,000 samples generated according to $p\left(x_1, x_2|\theta\right)$ at each of 10 equally-spaced discrete values of θ.

Figure 9 demonstrates that the trained network is able to learn that the sensitivity of individual observations to the value of the model parameter θ varies with x_2; the predicted posterior density is much flatter in regions where all parameter values give similar distributions.

To validate the network's performance, we generate test data sets $\{\mathbf{x}_i\}$ for various test parameter values θ_t. As in the one observable case of Sect. 5.2, each point $\mathbf{x}_i$ is passed through the network and a function $\tilde{p}_i\left(\theta|\mathbf{x}_i\right)$ is calculated. We then find the maximum likelihood estimator of θ. An accurate network should, within statistical uncertainty, reproduce the line $\theta_{\mathrm{ML}} = \theta_t$. Indeed, Fig. 10 shows that the network is accurate and unbiased, and that it provides a reasonable uncertainty estimate.

6 Method limitations

In the previous section, we have demonstrated necessary corrections for networks taking one or two values for each observation. In principle, this method could work with an arbitrarily high number of input observables. However, the current correction technique has a curse-of-dimensionality problem

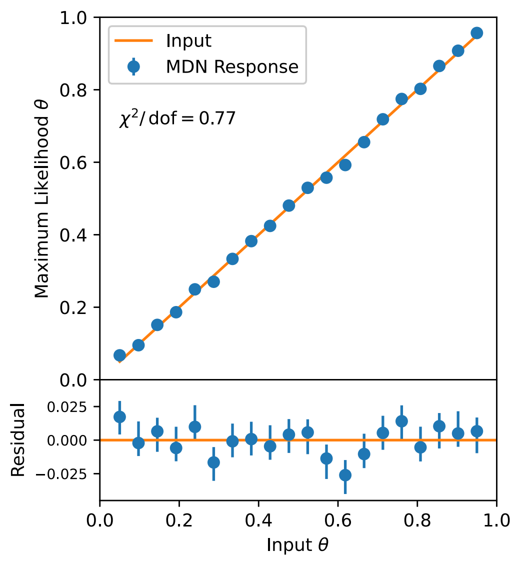

Fig. 10 Maximum likelihood estimator for parameter value θ_{ML}, and associated statistical uncertainties estimated using Wilks' theorem, for samples generated at various values of true θ. No bias is observed and the error estimation works well

in that computing the correction term $\mathcal{S}$ defined in Eq. (8) requires an estimate of $p(\mathbf{x})$ over the full training data set. When determined using histograms with m bins per axis, N observables will require an m^N-bin histogram to be reasonably well populated. Other techniques such as generic kernel density estimation [11] or k-nearest neighbors run into the same issues eventually, although at differing rates, and the best option should be explored in each specific situation. The dimensionality issue is alleviated somewhat by the fact that only one function ($p(\mathbf{x})$ over the entire training dataset) needs to be estimated, unlike methods which need to generate templates separately at each generated parameter point.

Another limitation concerns the spacing of training points. While we have demonstrated that MDNs can be trained with discrete templates of data and still interpolate properly to the full continuous parameter space, it has been necessary to use templates from parameters which are uniformly distributed, to enforce that $p(\boldsymbol{\theta})$ is flat. If the parameter values are not equally spaced, this would correspond to a non-flat $p(\boldsymbol{\theta})$. In principle, weights could be used during training to correct for this effect, but reconstructing $p(\boldsymbol{\theta})$ from the distribution of $\boldsymbol{\theta}$ is a density estimation problem and it is not clear if there is an optimal estimation method for the required $p(\boldsymbol{\theta})$ if there are only a limited number of $\boldsymbol{\theta}$ available. This is an area for future investigation.

7 Conclusions

Mixture density networks can output posterior density functions which robustly approximate likelihood functions and enable the estimation of parameters from data, even if the training data is only available at discrete values. This method permits one to proceed directly from (possibly multiple) observables to a likelihood function without having to perform the intermediate step of creating a statistical model for the observables, as would be required by many parameter estimation techniques. The MDN technique can be applied even with training data that provide a discrete set of parameter points, provided that the points are spaced evenly and certain corrections and constraints are applied during the training of the network.

An introductory tutorial has been implemented in a Jupyter Notebook and made public [12].

Acknowledgements This work was supported by the US Department of Energy, Office of Science, Office of High Energy Physics, under Award number DE-SC0007890.

Data Availability Statement This manuscript has no associated data or the data will not be deposited. [Authors' comment: There are no external data associated with the manuscript.]

References

1. J. Brehmer, K. Cranmer, G. Louppe, J. Pavez, Phys. Rev. D **98**(5), 052004 (2018). https://doi.org/10.1103/PhysRevD.98.052004
2. F. Flesher, K. Fraser, C. Hutchison, B. Ostdiek, M.D. Schwartz, Parameter inference from event ensembles and the top-quark mass (2020)
3. A. Andreassen, S.C. Hsu, B. Nachman, N. Suaysom, A. Suresh, Phys. Rev. D **103**(3), 036001 (2021). https://doi.org/10.1103/PhysRevD.103.036001
4. M. Baak, S. Gadatsch, R. Harrington, W. Verkerke, Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip. **771**, 39–48 (2015). https://doi.org/10.1016/j.nima.2014.10.033
5. K. Cranmer, G. Lewis, L. Moneta, A. Shibata, W. Verkerke, HistFactory: a tool for creating statistical models for use with RooFit and RooStats. Technical Report. CERN-OPEN-2012-016, New York University, New York (2012). https://cds.cern.ch/record/1456844
6. A.L. Read, Nucl. Instrum. Methods A **425**, 357 (1999). https://doi.org/10.1016/S0168-9002(98)01347-3
7. C.M. Bishop, Mixture density networks. Aston University Neural Computing Research Group Report NCRG/94/004 (1994)

8. A. Paszke et al., in *Advances in Neural Information Processing Systems*, vol. 32 (Curran Associates, Inc., Red Hook, 2019), pp. 8024–8035
9. D. Kingma, J. Ba, Adam: a method for stochastic optimization. Int Conf. Learn. Representations (2014)
10. S.S. Wilks, Ann. Math. Stat. **9**(1), 60 (1938). https://doi.org/10.1214/aoms/1177732360
11. E. Parzen, Ann. Math. Stat. **33**(3), 1065 (1962). https://doi.org/10.1214/aoms/1177704472
12. C.D. Burton, Mdn_likelihood_tutorial. https://doi.org/10.5281/zenodo.5061425