Multivariate Density Estimation by Neural Networks

Dewi E. W. Peerlings, Jan A. van den Brakel, Nalan Başt ürk, and Marco J. H. Puts

***Abstract*— We propose nonparametric methods to obtain the Probability Density Function (PDF) to assess the properties of the underlying data generating process (DGP) without imposing any assumptions on the DGP, using neural networks (NNs). The proposed NN has advantages compared to well-known parametric and nonparametric density estimators. Our approach builds on literature on cumulative distribution function (CDF) estimation using NN. We extend this literature by providing analytical derivatives of this obtained CDF. Our approach hence removes the numerical approximation error in differentiating the CDF output, leading to more accurate PDF estimates. The proposed solution applies to any NN model, i.e., for any number of hidden layers or hidden neurons in the multilayer perceptron (MLP) structure. The proposed solution applies the PDF estimation by NN to continuous distributions as well as discrete distributions. We also show that the proposed solution to obtain the PDF leads to good approximations when applied to correlated variables in a multivariate setting. We test the performance of our method in a large Monte Carlo simulation using various complex distributions. Subsequently, we apply our method to estimate the density of the number of vehicle counts per minute measured with road sensors for a time window of 24 h.**

***Index Terms*— Count data, Faà di Bruno (FDB) derivatives, Monte Carlo simulation, nonparametric density estimation, unsupervised learning.**

I. Introduction

PROBABILITY Density Function (PDF) estimation on low dimensions has been studied extensively, both parametrically and nonparametrically [1]. Parametric density estimation assumes a specific distribution for the data generating process (DGP), and the parameters of this distribution are then estimated, using, e.g., maximum likelihood. A conventional distribution used for this purpose is the normal distribution, but this distributional assumption is too restrictive for several DGPs [2]–[4]. To tackle this difficulty, nonparametric methods have been developed and applied to the PDF estimation. Nonparametric methods, such as kernel density estimator (KDE), do not impose distributional assumptions, but instead determine the distribution properties based on the data. Disadvantages of the KDE are the results' dependency on the bandwidth specification, particularly in a multivariate setting, together with large memory and high computational time required especially for large samples. In addition, the KDE method is fragmented, i.e., the combination of several kernel densities does not have a functional form [5].

Next to these two approaches, a recently developed third approach is the neural network (NN) approach. These methods attempt to obtain the best of both worlds: no assumptions are made *a priori* and their extension to multiple dimensions is theoretically straightforward, even though computational issues, such as time and memory, still need to be considered. The importance of nonparametric density estimation is also highlighted in [6]–[9], in the context of KDEs, self-organizing maps, and NN for forecasting with sequential data. We, on the other hand, focus on the NN approach that obtains a functional form of the PDF of the underlying DGP. Specifically, multilayer perceptrons (MLPs) are used, feeding forward from the input layer to the output layer [10, Ch. 4].

MLPs, and in general NNs, are supervised learning methods where the output needs to be labeled. However, when used for PDF estimation, they provide a solution to the unsupervised learning problem of describing the properties of the underlying data distribution [5], [11], [12]. To estimate the underlying DGP with MLPs, the data need to be labeled. The NN approach is nevertheless an unsupervised learning task as the characteristics of the underlying DGP are not known *a priori* and need to be estimated.

There are two distinct methods within the NN approach for density estimation. The first method, proposed in [5], directly estimates the PDF by NN. The second method is to estimate the cumulative distribution function (CDF) by NN and obtain the PDF by differentiating this MLP as in [12] and [13]. To build an MLP, training and test sets with labeled input variables are required. Trentin *et al.* [5] construct these labels for the PDF in a similar way as the KDE approach does with some alterations on bandwidth selection and bias prevention. They show that their method, Parzen neural networks (PNNs), is less sensitive to bandwidth specification than KDE. However, it is more natural to construct labels for the CDF values instead of the PDF values since these values range from 0 to 1 and are monotonic. More importantly, constructing PDF values is more prone to statistical error than CDF values as PDF values are sensitive to the bandwidth, whereas CDF values do not require bandwidth selection.

Manuscript received 11 October 2021; revised 25 March 2022; accepted 30 June 2022. The work of Nalan Başt ürk was supported by the Netherlands Organization for Scientific Research (NWO) under Grant 195.187. *(Corresponding author: Dewi E. W. Peerlings.)*

Dewi E. W. Peerlings and Jan A. van den Brakel are with the Quantitative Economics Department, School of Business and Economics, Maastricht University, 6200 MD Maastricht, The Netherlands, and also with Statistics Netherlands, 6401 CZ Heerlen, The Netherlands (e-mail: d.peerlings@maastrichtuniversity.nl; j.vandenbrakel@maastrichtuniversity.nl).

Nalan Başt ürk is with the Quantitative Economics Department, School of Business and Economics, Maastricht University, 6200 MD Maastricht, The Netherlands (e-mail: n.basturk@maastrichtuniversity.nl).

Marco J. H. Puts is with Statistics Netherlands, 6401 CZ Heerlen, The Netherlands (e-mail: m.puts@cbs.nl).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3190220.

Digital Object Identifier 10.1109/TNNLS.2022.3190220

In this article, we follow the approach mentioned in [12] and [13], i.e., we estimate the CDF by NN and obtain the PDF by differentiating the corresponding MLP. In this way, we show how statistical theory is incorporated in the NN training algorithms using MLP architectures. The novelty of this article lies in a number of extensions. First, PDF estimates of multivariate, discrete, or continuous data are obtained. Second, instead of numerically deriving the PDF from the obtained CDF, analytical derivations are provided. This is similar to the approach taken in [13], but we extend the method from MLPs with a single hidden layer, to MLPs with more than one hidden layer. This allows the estimation of more complex distributions. Unlike [5] and [12], the proposed method in this article can be applied to discrete distributions. Third, our method does not require to selecting an initial bandwidth empirically as opposed to [5]. We conjecture that choosing this bandwidth can be cumbersome especially in a multivariate setting and thus highlight this as an advantage of our method. Finally, unlike the literature that focuses mostly on independent input data, we show through simulations that the interconnectivity of the NN properly accommodates the dependency of correlated multivariate data.

It is conventional to assess the properties of a new estimation method using Monte Carlo simulations where the true data properties are compared to estimated properties and the simulation is replicated several times to reduce the effect of simulation noise on the reported results [14]. We follow this convention for the proposed method and report results for Monte Carlo studies with 100 simulation replications for each DGP. The advantages of a Monte Carlo simulation are that we can evaluate the performance of the method by comparing the PDF approximation with the real PDF. By taking averages over all 100 simulation replications, simulation noise is removed. In this way, we avoid that atypical draws in a simulation sample obscure the expected performance of the method.

The remainder of this article is organized as follows. The methodology to obtain a PDF by analytically differentiating an MLP that models a CDF for multivariate, continuous, and discrete distributions is described in Section II. Results for a bivariate correlated mixed Poisson distribution and a real data application are shown in Section III. A conclusion is given in Section IV. The Supplementary Material [15], which belongs to this article, contains additional details of the analytical differentiation of the MLP in Sections 2 and 3. Section 4 contains a discussion on how a nondecreasing estimate with the proposed NN approach can be obtained. Section 5 of Supplementary Material contains Monte Carlo simulation results for data generated from four distinct distributions: a mixture of normal distributions, a mixture of generalized extreme value distributions, a mixture of Poisson distributions, and a mixture of correlated normal distributions.

II. Methodology

We propose a methodology to obtain the CDF and PDF of univariate or multivariate random variables using NN. The output of the proposed NN approach is the estimated empirical CDF. The PDF is obtained as the analytical derivative of the estimated CDF by NN with respect to the input variables. To the best of our knowledge, such analytical derivatives for a general NN approach for univariate and multivariate PDF estimation have not been considered in the literature. The main advantage of our approach compared to the related literature is reducing the approximation error in PDF estimation. Specifically, the approximation error in the second step of obtaining the PDF from the CDF output is smaller when using analytical derivatives instead of numerical derivatives as in [12]. In this section, we present the proposed NN approach and the analytical derivatives for obtaining the PDF.

A. CDF Estimation Using NN

In the proposed method, the input layer, $\boldsymbol{x}_{\cdot t} = \{x_{1t}, \ldots, x_{Nt}\}$, consists of N-variate random variables and the output layer, and $\hat{y}_t$ is the approximated empirical CDF for the tth observation, $t = 1, \ldots, T$. We have N input neurons, M hidden neurons for each hidden layer, and H hidden layers. It is understood that the number of hidden neurons might differ between the hidden layers. Let q represent the hidden layer of the NN and $\boldsymbol{W}^{[q]}$ be the $N \times M$ matrix of weights from N input neurons in the input layer to M hidden neurons in hidden layer 1.[1] The $M \times M$ matrix of weights $\boldsymbol{W}^{[q]}$ connects layer q to layer $q+1$ for $q = 1, \ldots, H-1$. The $M \times 1$ matrix of weights $\boldsymbol{W}^{[H]}$ connects hidden layer H to the output layer. An $M \times 1$ vector of possible biases $\mathbf{b}^{[q]}$ is introduced in layer $q+1$ for $q = 0, 1, \ldots, H-1$ and $\mathbf{b}^{[H]}$ is the scalar bias introduced in the output layer.

The feedforward algorithms for the layers of the NN are defined as

$$\mathbf{h}_{\cdot t}^{[1]} = \boldsymbol{f}\left(\mathbf{W}^{[0]\top}\mathbf{x}_{\cdot t} + \mathbf{b}^{[0]}\right) \tag{1}$$

$$\mathbf{h}_{\cdot t}^{[q]} = \boldsymbol{f}\left(\mathbf{W}^{[q-1]\top}\mathbf{h}_{\cdot t}^{[q-1]} + \mathbf{b}^{[q-1]}\right) \quad \text{for } q = 2, \ldots, H \tag{2}$$

$$\tilde{y}_t = \mathbf{W}^{[H]\top}\mathbf{h}_{\cdot t}^{[H]} + \mathbf{b}^{[H]} \tag{3}$$

where $\boldsymbol{f}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^M \to \mathbb{R}^M$, with $q = 1, \ldots, H$, see, e.g., [10, Ch. 4]. For operations that map the input neurons $\mathbf{x}_{\cdot t}$ by a linear combination to the M-vector of the first hidden layer in (1), $\boldsymbol{f}(\mathbf{x}_{\cdot t}) : \mathbb{R}^N \to \mathbb{R}^M$. If the output layer would take an activation function in (3), then $\boldsymbol{f}(\mathbf{h}_{\cdot t}^{[H]}) : \mathbb{R}^M \to \mathbb{R}^1$ maps $\mathbf{h}_{\cdot t}^{[H]}$ by a linear combination to the output layer with a single neuron.

The empirical CDF target variable $\hat{y}_t$ can be approximated using grid-based methods over the multivariate input space, such as evenly spaced values γ_{ng} for $g = 1, \ldots, G$ and $n = 1, \ldots, N$, defined over the range of input variables. In this setting, the size of grid space G^N depends on the number of input variables N. In general, the empirical CDF estimation is more accurate with an increase in sample size T and the obtained empirical CDF is smoother with an increase in the number of grids G. For continuous distributions, we set the number of grids $G > T$. For discrete distributions, this does not apply. The smallest grid value is equal to the smallest input variable observation. The largest grid value is the step size plus the largest input variable observation.

[1] H hidden layers imply $H+1$ weight matrices and biases.

In the case of a single input variable $N = 1$, we define the following grid values: $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_G)$ with $\gamma_g < \gamma_{g+1}$ for $g = 1, \ldots, G-1$, where $\gamma_1, \ldots, \gamma_G$ are evenly spaced, capturing the range of input variables. The target values $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)'$ are calculated as follows:

$$\gamma_t = \underset{\gamma_g \in \{\gamma_1, \ldots, \gamma_G\}}{\arg,\min} \left(x_{1t} - \gamma_g \mid x_{1t} \geq \gamma_g\right) \tag{4}$$

$$\hat{y}_t = \frac{\sum_{t'=1}^{T} I(x_{1t'} \leq \gamma_t)}{T} \tag{5}$$

for $t = 1, \ldots, T$ and $I()$ is the indicator function that takes the value 1 if the argument is true and 0 otherwise, i.e., for each observation, we choose the highest grid value that is smaller than the observation in (4). The empirical CDF of each output point is then calculated using the number of observations below the grid point, as in (5).

For the multivariate density problem, the specified grid $\boldsymbol{\Gamma}$ is of dimension $\mathbb{R}^N$ where a vector of grid values are constructed for each combination of input variables. For $\boldsymbol{\Gamma} = (\boldsymbol{\gamma}_{\cdot 1}, \ldots, \boldsymbol{\gamma}_{\cdot G})$ with $\boldsymbol{\gamma}_{\cdot g} = (\gamma_{1g}, \ldots, \gamma_{Ng})'$ for $g = 1, \ldots, G$, the target values $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)'$ are calculated as follows:

$$\boldsymbol{\gamma}_{\cdot t} = \underset{\gamma_{ng} \in \{\gamma_{ng'}\}|_{g'=1}^{G}}{\arg,\min} \left(x_{nt} - \gamma_{ng} \mid x_{nt} \geq \gamma_{ng}\right) \quad \forall n \tag{6}$$

$$\hat{y}_t = \frac{\sum_{t'=1}^{T} I(x_{nt'} \leq \gamma_{nt}, \forall n)}{T} \tag{7}$$

where the grids are defined for each input variable in (6). The empirical CDF of each output point is calculated using the number of observations that are included in the N-dimensional grid box given in (7).

B. MLP Selection

The selection of an appropriate NN is achieved by minimizing a loss function. The loss function represents the discrepancy between the estimated CDF $\tilde{y}_t$ by NN and the target CDF $\hat{y}_t$ represented by the empirical CDF calculated as in (5) or (7). For the simulation experiments and real data application in Section III, the least-squares ($L2$) loss function $\text{L2}_{\widehat{\text{CDF}}}$ is used

$$L2_{\widehat{\text{CDF}}} = \sum_{t=1}^{T} [\tilde{y}_t(\boldsymbol{x}_{\cdot t}) - \hat{y}_t(\boldsymbol{x}_{\cdot t})]^2. \tag{8}$$

The loss function calculates the squared difference between the target CDF $\hat{y}_t$ and the estimated CDF $\tilde{y}_t$. The loss function (8) can be adjusted to ensure a nondecreasing CDF through a defined penalty [12]. However, tuning this penalty was not straightforward in our applications and this requires further research; therefore, we omit this penalty for this current article [see Section 4 of the Supplementary Material [15] for details about incorporating the penalty to the loss function (8)].

The target CDF $\hat{y}_t$ is an approximation of the true CDF y_t. To measure the discrepancy between the true CDF y_t and the estimated CDF $\tilde{y}_t$, the L2_{CDF} loss function is used

$$L2_{\text{CDF}} = \sum_{t=1}^{T} [y_t(\boldsymbol{x}_{\cdot t}) - \tilde{y}_t(\boldsymbol{x}_{\cdot t})]^2. \tag{9}$$

In a real data application, the true CDF is not known. Hence, the feasible loss function is $\text{L2}_{\widehat{\text{CDF}}}$. The approximation of the true CDF y_t by the target CDF $\hat{y}_t$ is therefore important in order to obtain an adequate CDF estimate $\tilde{y}_t$. This relative loss obtained by the empirical CDF $\hat{y}_t$, which approximates the true CDF y_t, is represented by the $\text{L2}_{\widehat{\text{CDF}}}$ loss function

$$L2_{\widehat{\text{CDF}}} = \sum_{t=1}^{T} [y_t(\boldsymbol{x}_{\cdot t}) - \hat{y}_t(\boldsymbol{x}_{\cdot t})]^2. \tag{10}$$

Hence, there are three distinct loss measurements. In the case of simulated data, the loss can be obtained using the difference between the true CDF y_t and the estimated CDF $\tilde{y}_t$, represented by L2_{CDF} in (9), whereas in real data applications, the true CDF is not known; thus, the calculated loss is based on the difference between the target CDF $\hat{y}_t$ and the estimated CDF $\tilde{y}_t$ defined by $\text{L2}_{\widehat{\text{CDF}}}$ in (8). The third loss measures the discrepancy of the approximation, which determines the maximum performance of the NN. Note that the discrepancy between the estimated CDF $\tilde{y}_t$ and the true CDF y_t is not equal to adding the $L2$ losses of (8) and (10). When the trained NN is differentiated resulting in the estimated PDF $\tilde{y}'_t$, the difference between the estimated PDF $\tilde{y}'_t$ and the true PDF y'_t, is summarized in the L2_{PDF} loss function

$$L2_{\text{PDF}} = \sum_{t=1}^{T} \left[y'_t(\boldsymbol{x}_{\cdot t}) - \tilde{y}'_t(\boldsymbol{x}_{\cdot t})\right]^2. \tag{11}$$

For discrete distributions, this is similarly defined and denoted as L2_{PMF} loss. How to obtain the estimated pdf or probability mass function (PMF) $\tilde{y}'_t$ is explained in Section II-C. For the real data application in Section III-C, function (8) is used as this is the only feasible loss function to asses the performance. For the simulation cases in Section III-A and Section 5 of the Supplementary Material [15], all above losses can be used. Specifically, means and corresponding standard errors of these loss functions over 100 simulation replications are shown.

The optimal MLP is chosen based on $\text{L2}_{\widehat{\text{CDF}}}$ loss as this is the feasible loss function for a real data application. Note that training NN can be computationally costly in certain cases [16, Ch. 11]. We propose the following method to consider several MLP structures and choose the best MLP structure to derive the PDF or PMF estimate. We denote these different MLP structures with different numbers of hidden neurons and/or hidden layers as "sub-MLPs" and estimate a large MLP that contains these sub-MLPs, as shown graphically in Fig. 1. The MLP with the smallest loss value is chosen to be used for derivation of the corresponding PDF or PMF. We set the same hyperparameters such as the learning rate and weight initialization and the same activation functions for each MLP, although these can be allowed to change across sub-MLPs. In essence, the computational complexity of a sub-MLP increases with more layers and neurons. The increased complexity of a sub-MLP or the large MLP is handled efficiently by Keras/Tensorflow [17], [18] that exploits the parallel hardware platform. We experience that given the defined sub-MLPs, it is computationally advantageous to train the proposed large MLP compared to training its sub-MLPs

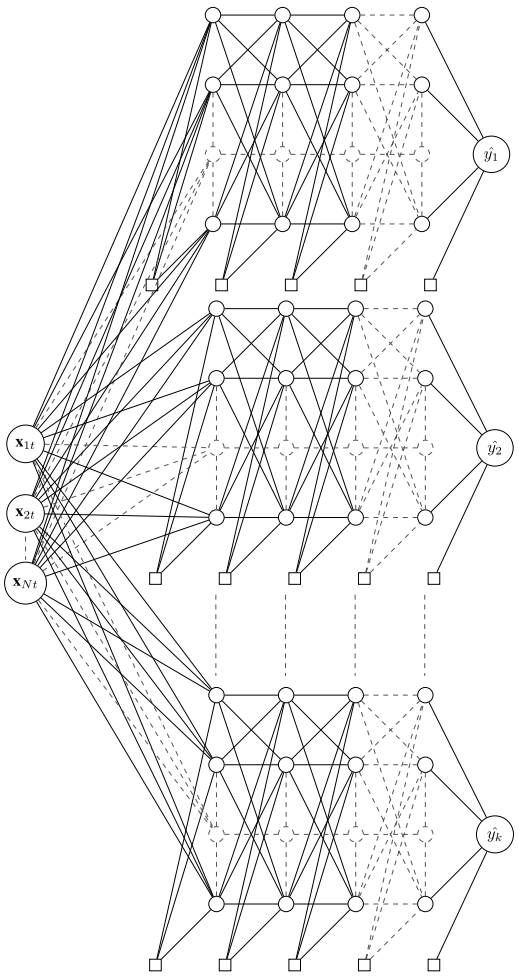

Fig. 1. Architecture of one MLP consisting of several sub-MLPs.

separately. In the latter case, the overhead to handling the extra iterations is found to be computationally expensive.

The ability to obtain the estimate of a complex distribution is determined by the capacity of an NN. The capacity of an NN refers to the ability to fit the target variable correctly. This can be increased by several parameters. In particular, the number of epochs and the model structure of the NN play an important role. There is a substantial difference between setting the number of hidden layers or the number of hidden neurons. Increasing the number of hidden layers is mainly done when the distribution of the target variable is nonlinear. The more hidden layers, the more activation functions are nested within each other, the more nonlinear and complex the NN. There are also several reasons for increasing the number of hidden neurons. Hidden neurons specify the "flat" capacity of the NN opposed to the "deep" capacity that hidden layers create. Note that without enough neurons in each layer, a more complex distribution, e.g., a multinomial distribution, is impossible to fit. Hence, without enough neurons, increasing the number of hidden layers does not create enough capacity. However, if the capacity is larger than needed, estimates are more volatile than the desired one. In particular, including too many hidden layers results in a very erratic empirical CDF.

C. PDF Estimation Using Analytical Derivatives

As the empirical CDF $\tilde{y}_t$ is estimated by NN, the corresponding PDF can be derived from this functional form. Using (1)–(3), the output of the NN can be written as

$$\tilde{y}_t = \mathbf{W}^{[H]^\top} \boldsymbol{f}\left(\mathbf{W}^{[H-1]^\top} \cdots \boldsymbol{f}\left(\mathbf{W}^{[0]^\top} \mathbf{x}_{\cdot t} + \mathbf{b}^{[0]}\right) \cdots + \mathbf{b}^{[H-1]}\right) + \mathbf{b}^{[H]} \quad (12)$$

which is a composition of several nested activation functions in each hidden layer, with no linear activation function on the output layer. The joint PDF is obtained by differentiating (12) with respect to N input variables. Therefore, an activation function that is N differentiable needs to be chosen. The conventional rectified linear unit (ReLU) activation function is not used since higher derivatives are zero. This implies that a lot of information would be lost in order to obtain the joint PDF. Potential activation functions $\boldsymbol{f}(\cdot)$ that are considered are sigmoid functions such as the logistic function $\boldsymbol{\sigma}(\cdot)$ and the hyperbolic tangent function denoted as $\boldsymbol{\tau}(\cdot)$ since these functions are N differentiable (see Appendix A for details of the derivations of these functions). Derivatives are conducted using $\boldsymbol{\sigma}(\cdot)$, but $\boldsymbol{\tau}(\cdot)$ is incorporated similarly. The input of these activation functions is any linear combination of $\boldsymbol{x}_{\cdot t}$ together with weights (and biases) nested with previous layers, which implies that $\boldsymbol{\tau}(\mathbf{x}_{\cdot t}) : \mathbb{R}^N \to \mathbb{R}^M$ and $\boldsymbol{\sigma}(\mathbf{x}_{\cdot t}) : \mathbb{R}^N \to \mathbb{R}^M$ for the input layer, $\boldsymbol{\tau}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^M \to \mathbb{R}^M$ and $\boldsymbol{\sigma}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^M \to \mathbb{R}^M$ for the hidden layer, and $\boldsymbol{\tau}(\mathbf{h}_{\cdot t}^{[H]}) : \mathbb{R}^M \to \mathbb{R}^1$ and $\boldsymbol{\sigma}(\mathbf{h}_{\cdot t}^{[H]}) : \mathbb{R}^M \to \mathbb{R}^1$ for the output layer, defined as

$$\boldsymbol{\sigma}\left(\mathbf{h}_{\cdot t}^{[q]}\right) = \frac{1}{1 + \exp\left(-\mathbf{h}_{\cdot t}^{[q]}\right)}. \quad (13)$$

Since (12) is a combination of nested functions, $\tilde{y}_t$ can also be represented by

$$\tilde{y}_t = \boldsymbol{g}^{[H]} \circ \boldsymbol{f} \circ \boldsymbol{g}^{[H-1]} \circ \boldsymbol{f} \circ \cdots \circ \boldsymbol{f} \circ \boldsymbol{g}^{[1]} \circ \boldsymbol{f} \circ \boldsymbol{g}^{[0]}, \quad (14)$$

where $\boldsymbol{f}(\cdot)$ represents the nonlinear activation function and $\boldsymbol{g}^{[q]}(\cdot)$ represents the linear function at hidden layer q, combining weights, and variables $\mathbf{x}_{\cdot t}$ or $\mathbf{h}_{\cdot t}^{[q-1]}$.

To acquire the PDF function, the estimated CDF function $\tilde{y}$ is differentiated with respect to all N input variables denoted as

$$\tilde{y}'_t = \frac{\partial^N}{\partial x_{1t}, \ldots, \partial x_{Nt}} \tilde{y}_t(\mathbf{x}_{\cdot t}). \quad (15)$$

Specifically, the differentiation of the NN starts from the right-hand side of (14) and ends on the left-hand side of the equation, where each step treats one composite function at a time. The subsequent step takes this resulting composite function as one function, which forms a new composite function together with the next function. In this way, differentiation is performed by tackling composite functions subsequently. Each step, which tackles one composite function, calculates all partial derivatives and the joint derivative. For example, for three input neurons, there are six partial derivatives and one joint derivative. To obtain the Nth derivative of these composite functions, denoted as in (15), the chain rule is used. Faà di Bruno [19] gave a generalization of the chain rule for this higher derivative, which was originally proposed

in [20] 50 years earlier. In this way, differentiating nested composite functions with respect to N variables can be tracked in a structured way. Hence, to differentiate this composition, Faà di Bruno's (FDB) differentiation method is used (see [21] for a more modern approach).

When computing the Nth derivative, N input variables are partitioned in all possible combinations of subsets. For N input variables, there are P different partitions π_p for $p = 1, \ldots, P$. In each partition π_p, the N variables are divided in a unique way over subsets B_{pk}. The number of subsets in partition π_p is represented by the cardinality of π_p, denoted as $|\pi_p|$. Thus, B_{pk} denotes the kth subset of partition π_p, where $k = 1, \ldots, |\pi_p|$. Accordingly, each subset B_{pk} has cardinality $|B_{pk}|$, which is the number of variables in a subset or the size of a subset. In each partition, each variable occurs only one time in one of the $|\pi_p|$ subsets of this partition. Thus, for each π_p, all subsets B_{pk} contain all input variables N combined in a different way than other partitions $\pi_{p'}$, where $p \neq p'$. The intersection of all subsets of a particular partition is empty. Thus, $\cup_{k=1}^{|\pi_p|} B_{kp} = N$ and $\cap_{k=1}^{|\pi_p|} B_{kp} = \emptyset$ (see [22, Ch. 9] for more on partitions).

The nested composite function of (14) consists of two type of functions: nonlinear and linear functions. These are alternately outer and inner functions, $f_1(\cdot)$ and $f_2(\cdot)$, respectively. When considering only one composite function, the following representation of FDB is used:

$$\frac{\partial^N}{\partial x_{1t} \cdots \partial x_{Nt}} f_1(f_2(\mathbf{x}_{\cdot t})) = \sum_{p=1}^{P} \frac{\partial^{|\pi_p|} f_1(f_2(\mathbf{x}_{\cdot t}))}{\partial f_2(\mathbf{x}_{\cdot t})^{|\pi_p|}} \prod_{k=1}^{|\pi_p|} \frac{\partial^{|B_{pk}|} f_2(\mathbf{x}_{\cdot t})}{\prod_{n \in B_{pk}} \partial x_{nt}} \equiv \sum_{p=1}^{P} D_{1p} \prod_{k=1}^{|\pi_p|} D_{2k} \tag{16}$$

where $\partial^{|\pi_p|}$ refers to the $|\pi_p|$th derivative. Similarly, $\partial^{|B_{pk}|}$ refers to the $|B_{pk}|$th partial derivative. For notational reasons, D_{1p} represents $(\partial^{|\pi_p|} f_1(f_2(\mathbf{x}_{\cdot t}))/\partial f_2(\mathbf{x}_{\cdot t})^{|\pi_p|})$ for partition π_p and D_{2k} represents $(\partial^{|B_{pk}|} f_2(\mathbf{x}_{\cdot t})/\prod_{n \in B_{pk}} \partial x_{nt})$ for subset B_{pk} of partition π_p. Hence, for each partition π_p, there is one D_1 and $|\pi_p|$ number of D_2.

As mentioned before, differentiation is performed by differentiating from right to left of (14), treating one composite function in each step. Depending on which composite function in the NN is treated, either a nonlinear function is the outer function and a linear function is the inner function $\boldsymbol{f}(\boldsymbol{g}^{[q]}(\cdot))$ or vice versa $\boldsymbol{g}^{[q]}(\boldsymbol{f}(\cdot))$. There are three different steps that are applied. The first step is called the input step, which deals with the composite function of the input layer to the first hidden layer. This implies that $f_1(\cdot) : \mathbb{R}^M \to \mathbb{R}^M$ and $f_2(\cdot) : \mathbb{R}^N \to \mathbb{R}^M$ since this step operates between the input layer consisting of N input neurons and the first hidden layer consisting of M hidden neurons.

The second step is called the hidden step. This step treats the differentiation of the composite functions within the hidden layers of the NN. This entails that $f_1(\cdot) : \mathbb{R}^M \to \mathbb{R}^M$ and $f_2(\cdot) : \mathbb{R}^M \to \mathbb{R}^M$ since this step operates in between hidden layers that consist of M hidden neurons. The hidden step tackles two different types, treated subsequently. Type 1 accommodates differentiation when the outer function is linear, that is, $\boldsymbol{g}^{[q]} \circ \boldsymbol{f}$. Type 2 corresponds to differentiation when the outer function is nonlinear, that is, $\boldsymbol{f} \circ \boldsymbol{g}^{[q]}$. Note that when the outer function is linear, all higher derivatives D_1 are 0, and when the inner function is linear, all higher derivatives of D_2 are 0. This implies that many parts of the derivatives are eliminated.

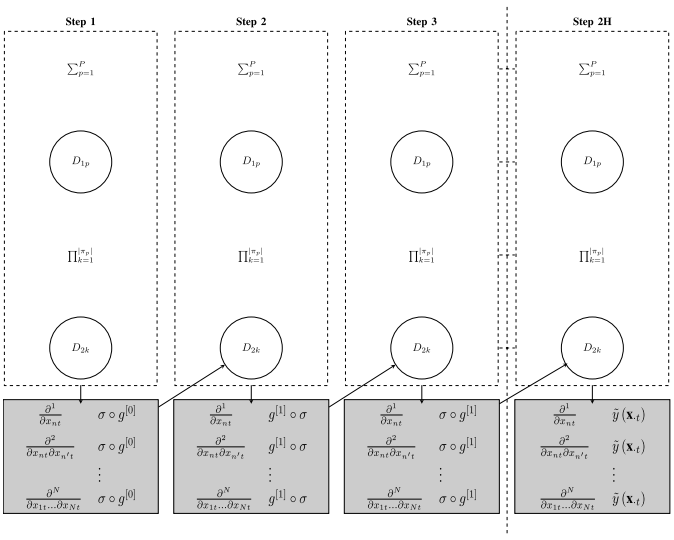

Fig. 2. Function of nested composite functions combined with FDB's method.

The third step is called the output step, performing differentiation from the last hidden layer to the output layer. This means that $f_1(\cdot) : \mathbb{R}^M \to \mathbb{R}^1$ and $f_2(\cdot) : \mathbb{R}^M \to \mathbb{R}^1$. This step is also divided into two types, which depends on imposing an activation function on the output layer or not, that is, types 1 and 2, which correspond to $\boldsymbol{g} \circ \boldsymbol{f}$ and $\boldsymbol{f} \circ \boldsymbol{g}$, respectively. If there is an activation function imposed on the output layer, then types 1 and 2 are treated subsequently.

Fig. 2 shows a schematic overview of all differentiation steps. In each step, the FDB derivative (16) is computed for all possible combinations of N input variables. These FDB derivatives in each step, depicted by the gray boxes, are used as D_{2k} for each $k \in \pi_p$ for all p in the next step. Derivations are denoted on neuron level, that is, derivations of every hidden layer are computed for each hidden neuron. Therefore, the multivariate functions of (12)–(14) turn into univariate functions of hidden neuron m, $f(\cdot)_m : \mathbb{R}^1 \to \mathbb{R}^1$ and $g^{[q]}(\cdot)_m : \mathbb{R}^1 \to \mathbb{R}^1$, which are nonlinear and linear functions, respectively. All the steps are summarized in algorithm 1 (see Section 3 of the Supplementary Material [15] for an extended version of the algorithm).

Algorithm 1: Differentiation of the Functional Form of the Empirical CDF $\tilde{y}_t$

1: Let $\tilde{y}_t$ be the nested function composed of nonlinear activation functions $\boldsymbol{f}(\cdot)$ and linear functions $\boldsymbol{g}^{[q]}(\cdot)$ where q represents the hidden layer of the NN

$$\tilde{y}_t = \boldsymbol{g}^{[H]} \circ \boldsymbol{f} \circ \boldsymbol{g}^{[H-1]} \circ \boldsymbol{f} \circ \cdots \circ \boldsymbol{f} \circ \boldsymbol{g}^{[1]} \circ \boldsymbol{f} \circ \boldsymbol{g}^{[0]}.$$

2: *Input step*: Differentiate $\boldsymbol{f} \circ \boldsymbol{g}^{[0]} = \boldsymbol{f}(\boldsymbol{W}^{[0]^\top}\boldsymbol{x}_{\cdot t} + \boldsymbol{b}^{[0]})$ for each neuron m, where the outer function is a nonlinear activation function $\boldsymbol{f}(\cdot) : \mathbb{R}^M \to \mathbb{R}^M$ and the inner linear function $\boldsymbol{g}^{[q]}(\cdot) : \mathbb{R}^N \to \mathbb{R}^M$ which is the linear combination of input variables and weights, resulting in:

$$\frac{\partial\left(f \circ g^{[0]}\right)_m}{\partial x_{nt}} \quad \frac{\partial^2\left(f \circ g^{[0]}\right)_m}{\partial x_{nt}\partial x_{n't}} \quad \cdots \quad \frac{\partial^N\left(f \circ g^{[0]}\right)_m}{\partial x_{1t}\cdots\partial x_{Nt}},$$

for all n and for all n $\neq$ n', where the derivations are treated univariately for each hidden neuron m in hidden layer 1, that is $(f \circ g^{[0]})_m : \mathbb{R}^1 \to \mathbb{R}^1$. Similarly for each hidden neuron m in hidden layer $q+1$, $(f \circ g^{[q]})_m : \mathbb{R}^1 \to \mathbb{R}^1$.

3: **While** q < H **do**

Hidden step, type 1: Differentiate the composite function that connects two subsequent hidden layers where the outer function is linear: $\boldsymbol{g}^{[q]} \circ \boldsymbol{f}$ for each neuron m of hidden layer $q+1$. Note that each neuron in layer $q+1$ is connected to all neurons in hidden layer q which results in a long sum of derivatives.

$$\frac{\partial\left(g^{[q]} \circ f\right)_m}{\partial x_{nt}} \quad \frac{\partial^2\left(g^{[q]} \circ f\right)_m}{\partial x_{nt}\partial x_{n't}} \quad \cdots \quad \frac{\partial^N\left(g^{[q]} \circ f\right)_m}{\partial x_{1t}\cdots\partial x_{Nt}},$$

for all n and for all n $\neq$ n'.

Hidden step, type 2: Finish the hidden step by differentiating two subsequent hidden layers where the outer function is nonlinear $\boldsymbol{f} \circ \boldsymbol{g}^{[q]}$ for each neuron m of hidden layer $q+1$.

$$\frac{\partial\left(f \circ g^{[q]}\right)_m}{\partial x_{nt}} \quad \frac{\partial^2\left(f \circ g^{[q]}\right)_m}{\partial x_{nt}\partial x_{n't}} \quad \cdots \quad \frac{\partial^N\left(f \circ g^{[q]}\right)_m}{\partial x_{1t}\cdots\partial x_{Nt}},$$

for all n and for all n $\neq$ n'

q = q+1

End while

4: *Output step*: Now the final layer needs to be differentiated for each neuron m of layer H: $\boldsymbol{g}^{[H]} \circ \boldsymbol{f}$ likewise:

$$\frac{\partial\left(g^{[H]} \circ f\right)_m}{\partial x_{nt}} \quad \frac{\partial^2\left(g^{[H]} \circ f\right)_m}{\partial x_{nt}\partial x_{n't}} \quad \cdots \quad \frac{\partial^N\left(g^{[H]} \circ f\right)_m}{\partial x_{1t}\cdots\partial x_{Nt}},$$

for all n and for all n $\neq$ n'. If the output activation function is used, subsequently the following type has to be performed, for each neuron m of layer H:

$$\frac{\partial\left(f \circ g^{[H]}\right)_m}{\partial x_{nt}} \quad \frac{\partial^2\left(f \circ g^{[H]}\right)_m}{\partial x_{nt}\partial x_{n't}} \quad \cdots \quad \frac{\partial^N\left(f \circ g^{[H]}\right)_m}{\partial x_{1t}\cdots\partial x_{Nt}},$$

for all n and for all n $\neq$ n'. We obtain the following joint derivative for each neuron m of the last hidden layer H:

$$\sum_{m=1}^{M} \frac{\partial^N\left(g^{[H]} \circ f\right)_m}{\partial x_{1t}\cdots x_{Nt}}.$$

The FDB derivatives with respect to all combinations of input variables of each step, that is, the FDB derivatives shown in the gray box of Fig. 2, are referred to as the complete (partial) derivatives of each step in the example illustrated later on. These are different from the final derivatives, which are the derivatives resulting from the output step. The final derivatives are the aimed PDF estimates acquired from the CDF $\tilde{y}_t$. The difference between two subsequent steps is that the composite function $f_1(f_2(\cdot))$ in the first step changes to $f_2(f_1(\cdot))$ in the subsequent step, that is, the outer function of the previous step turns into the inner function of the next step. In other words, the FDB derivatives for all combinations of input variables N of the previous step turn into D_{2k} for all partitions π_p in the subsequent step. In total, there are $2 + 2(H-1) = 2H$ steps, the input step, output step, and $2(H-1)$ hidden steps.

We illustrate this PDF estimation procedure with an example. Suppose that the NN has two input neurons and one hidden layer with one hidden neuron. Then, the following is the representation of the output:

$$\tilde{y}_t = w_{11}^{[1]} \cdot f\left(w_{11}^{[0]} \cdot x_{1t} + w_{21}^{[0]} \cdot x_{2t} + b^{[0]}\right) + b^{[1]}.$$

This representation consists of three nested functions: $\tilde{y}_t = g^{[1]} \circ f \circ g^{[0]}$. The last two functions $f \circ g^{[0]}$ are considered first. Next, these two functions are seen as one function, represented as f. Then, the last composite function is treated, $g^{[1]} \circ f$, in order to obtain the joint PDF or in other words the second derivative of this output. Since there are two input variables, we consider the following set of partitions $\{\{\{1,2\}\}, \{\{1\},\{2\}\}\}$.

π_1: $\{\{1,2\}\}$ has $|\pi_1| = 1$ and $|B_{11}| = 2$.
π_2: $\{\{1\},\{2\}\}$ has $|\pi_2| = 2$ and $|B_{2k}| = 1$ for $k = 1$ and 2.

Considering the first composite function $f \circ g^{[0]}$ and following FDB, this results in the following differentiation to obtain the second derivative of $f \circ g^{[0]}$:

$$\begin{aligned}\frac{\partial^2}{\partial x_{1t}\partial x_{2t}} f\left(g^{[0]}(\mathbf{x}_{\cdot t})\right) = &\frac{\partial f\left(g^{[0]}(\mathbf{x}_{\cdot t})\right)}{\partial g^{[0]}(\mathbf{x}_{\cdot t})} \cdot \frac{\partial^2 g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{1t}\partial x_{2t}} \\ &+ \frac{\partial^2 f\left(g^{[0]}(\mathbf{x}_{\cdot t})\right)}{\partial g^{[0]}(\mathbf{x}_{\cdot t})^2} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{1t}} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{2t}}.\end{aligned} \tag{17}$$

As shown in Fig. 2, also the partial derivatives of all combinations of input variables N are derived. In this case, these are the first derivatives with respect to x_{nt} with $n = 1, 2$

$$\frac{\partial}{\partial x_{nt}} f\left(g^{[0]}(\mathbf{x}_{\cdot t})\right) = \frac{\partial f\left(g^{[0]}(\mathbf{x}_{\cdot t})\right)}{\partial g^{[0]}(\mathbf{x}_{\cdot t})} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{nt}}. \tag{18}$$

As there are many nested functions, partial derivatives are needed to complete FDB's differentiation in subsequent steps of the derivation. Equations (17) and (18) are repeated in every step in order to obtain the joint PDF. Every step treats a composite function, which is a part of the nested expression as result of the NN. This implies that three derivatives are calculated for each layer in order to acquire the joint PDF of two input variables, $(\partial g^{[0]}(\mathbf{x}_{\cdot t})/\partial x_{1t})$, $(\partial g^{[0]}(\mathbf{x}_{\cdot t})/\partial x_{2t})$, and $(\partial^2 g^{[0]}(\mathbf{x}_{\cdot t})/\partial x_{1t}\partial x_{2t})$. As is evident, The first derivatives are needed to obtain higher derivatives; in this case, the second

derivative of $g^{[1]} \circ f$ is then

$$\begin{aligned}
&\frac{\partial^2}{\partial x_{1t}\partial x_{2t}} g^{[1]}(f(\mathbf{x}_{\cdot t})) \\
&= \frac{\partial g^{[1]}(f(\mathbf{x}_{\cdot t}))}{\partial f(\mathbf{x}_{\cdot t})} \cdot \frac{\partial^2 f(\mathbf{x}_{\cdot t})}{\partial x_{1t}\partial x_{2t}} + \frac{\partial^2 g^{[1]}(f(\mathbf{x}_{\cdot t}))}{\partial f(\mathbf{x}_{\cdot t})^2} \cdot \frac{\partial f(\mathbf{x}_{\cdot t})}{\partial x_{1t}} \cdot \frac{\partial f(\mathbf{x}_{\cdot t})}{\partial x_{2t}} \\
&= \frac{\partial g^{[1]}(f(\mathbf{x}_{\cdot t}))}{\partial f(\mathbf{x}_{\cdot t})} \cdot \left[\frac{\partial f(g^{[0]}(\mathbf{x}_{\cdot t}))}{\partial g^{[0]}(\mathbf{x}_{\cdot t})} \cdot \frac{\partial^2 g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{1t}\partial x_{2t}} \right. \\
&\quad \left. + \frac{\partial^2 f(g^{[0]}(\mathbf{x}_{\cdot t}))}{\partial g^{[0]}(\mathbf{x}_{\cdot t})^2} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{1t}} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{2t}} \right] + \frac{\partial^2 g^{[1]}(f(\mathbf{x}_{\cdot t}))}{\partial f(\mathbf{x}_{\cdot t})^2} \cdot \\
&\quad \frac{\partial f(g^{[0]}(\mathbf{x}_{\cdot t}))}{\partial g^{[0]}(\mathbf{x}_{\cdot t})} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{1t}} \cdot \frac{\partial f(g^{[0]}(\mathbf{x}_{\cdot t}))}{\partial g^{[0]}(\mathbf{x}_{\cdot t})} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{2t}} \qquad (19)
\end{aligned}$$

where $(\partial^2 f(\mathbf{x}_{\cdot t})/\partial x_{1t}\partial x_{2t})$ is substituted with (17) and $(\partial f(\mathbf{x}_{\cdot t})/\partial x_{1t})$ and $(\partial f(\mathbf{x}_{\cdot t})/\partial x_{2t})$ with (18). Hence, the final joint pdf is obtained by (19) by applying FDB with respect to all input variables N. To obtain the final marginal pdf, FDB is executed with respect to input variable n

$$\begin{aligned}
\frac{\partial}{\partial x_{nt}} g^{[1]}(f(\mathbf{x}_{\cdot t})) &= \frac{\partial g^{[1]}(f(\mathbf{x}_{\cdot t}))}{\partial f(\mathbf{x}_{\cdot t})} \cdot \frac{\partial f(\mathbf{x}_{\cdot t})}{\partial x_{nt}} \\
&= \frac{\partial g^{[1]}(f(\mathbf{x}_{\cdot t}))}{\partial f(\mathbf{x}_{\cdot t})} \cdot \frac{\partial f\left(g^{[0]}(\mathbf{x}_{\cdot t})\right)}{\partial g^{[0]}(\mathbf{x}_{\cdot t})} \cdot \frac{\partial g^{[0]}(\mathbf{x}_{\cdot t})}{\partial x_{nt}} \qquad (20)
\end{aligned}$$

where $(\partial f(\mathbf{x}_{\cdot t})/\partial x_{nt})$ is again replaced with (18). Each hidden neuron has an FDB derivative. As a consequence, the final derivative is a sum over all these hidden neurons, as stated in Algorithm 1. The treated example has only one hidden neuron, and hence, there is no sum in (19) and (20). See Section 2 of the Supplementary Material [15] for the derivative of a 3-D case.

III. Results

In this section, we present the approximation properties of the proposed method using simulated data and real data. For simulated data experiments, we present the detailed results for data generated from a bivariate correlated mixed Poisson distribution, and we briefly summarize various other simulated data experiments with different DGPs, see [15] for more details.

A. Bivariate Correlated Mixed Poisson Simulation

We specifically consider the case of estimating the PMF of discrete multivariate data where the underlying variables are correlated. The data are simulated from the following bivariate correlated mixed Poisson distribution:

$$p(x_1, x_2) = e^{-(\lambda_1+\lambda_2+\lambda_{12})} \sum_{i=0}^{\min(x_1,x_2)} \frac{\lambda_1^{x_1-i}\lambda_2^{x_2-i}\lambda_{12}^{i}}{(x_1-i)!(x_2-i)i!} \qquad (21)$$

where $\lambda_1 = 4, \lambda_2 = 7$, and $\lambda_{12} = 10$ with a correlation of 0.65 (see [23] for further details).

We generate 6000 observations from the DGP in (21) and split the data such that we use 5000 observations for estimation (training and validation) and the remaining 1000 observations for testing. In order to reduce simulation noise, we replicate the simulation experiment 100 times. Regarding the MLP architecture, nine different models are considered with combinations of $H \in \{1, 2, 3\}$ hidden layers and $M \in \{10, 25, 50\}$ hidden neurons. The activation function is the logistic function in all layers except for the output layer. For each model, training is done using 10 000 training epochs, a batch size equal to the test sample size, and a learning rate of 0.001. The proposed method's performance is evaluated using the following loss functions over 100 Monte Carlo simulations: 1) $\text{L2}_{\widetilde{\text{CDF}}}$ based on the difference between the target CDF $\hat{y}_t$ and the estimated CDF $\tilde{y}_t$ and 2) the L2_{PMF} loss between the estimated PMF $\tilde{y}'_t$ and true PMF y'_t. To choose the MLP architecture and the corresponding weights, we use the $\text{L2}_{\widetilde{\text{CDF}}}$ since this is the feasible loss function that can be used in a real data application when we do not know the data DGP.

The mean and standard deviations of different $L2$ losses from all alternative models are summarized in Table I for the fivefold cross-validation sample and the test sample. The reported $L2$ losses for the validation sample correspond to average $L2$ losses over fivefold cross-validation samples and over 100 Monte Carlo simulations. Similarly, reported $L2$ losses for the test sample correspond to averages over 100 Monte Carlo simulations. The average $L2$ loss using the model with the smallest $\text{L2}_{\widetilde{\text{CDF}}}$ loss for each Monte Carlo simulation corresponds to the results for the "optimal model." This serves as a benchmark toward the alternative models.

Table I shows that the smallest $\text{L2}_{\widetilde{\text{CDF}}}$ is obtained by models 6–9 in both the validation sample and test sample. These models also have smaller L2_{PMF} losses compared to the remaining models. In this example, it is found to be important to have enough ($H \geq 2$) hidden layers, and the choice of the number of neurons, M, does not seem to be important, provided that the number of hidden layers is large enough. In addition, according to $\text{L2}_{\widetilde{\text{CDF}}}$, model 8 is selected as the best performing model obtaining the smallest $\text{L2}_{\widetilde{\text{CDF}}}$ for the validation sample. Model 8 also leads to a smaller L2_{PMF} loss in the test sample compared to models 1–7. Only model 9 has slightly smaller PMF losses. Thus, the chosen model according to the CDF loss also leads to a good PMF approximation.

We next analyze the variability of our results within the Monte Carlo replications. Fig. 3 summarizes the CDF and PMF estimation results for model 8 for three Monte Carlo simulations leading to the best, average, and worst $\text{L2}_{\widetilde{\text{CDF}}}$ losses in the test samples. The target and estimated CDFs as well as true and estimated PMFs are very similar (see Appendix B), and therefore, differences are shown with the consequence that these differences are magnified. Fig. 3(a), 3(c), and 3(e) show that the CDF errors span the whole surface and the number of negative errors is higher than the positive ones. The PMF errors presented in Fig. 3(b), 3(d), and 3(f) also span the whole surface with the addition that positive and negative errors are clustered. More specifically, the positive PMF errors in all three simulations are mainly clustered at the bottom-left tail of the distribution. We further note that the difference between CDF and PMF errors in Fig. 3 results from the fact that the

TABLE I

SAMPLE RESULTS OF BIVARIATE MIXED POISSON DATA

L2 Losses for Bivariate Mixed Poisson Data. The Table Reports the Mean (Standard Error in Parentheses) of the L2 Losses Using the Differences Between the Target and Estimated CDF and Between the True and Estimated PMF Obtained by the Proposed Method for the Validation and Test Samples

Model	Validation sample		Test sample	
	$L2_{\widetilde{CDF}}$	$L2_{PMF}$	$L2_{\widetilde{CDF}}$	$L2_{PMF}$
Optimal	7.403	0.865	13.299	0.871
	(0.177)	(0.020)	(0.891)	(0.021)
1 $(H = 1, M = 10)$	35.432	2.555	31.891	2.235
	(0.636)	(0.034)	(1.019)	(0.053)
2 $(H = 1, M = 25)$	48.235	3.386	42.888	2.878
	(0.855)	(0.046)	(1.002)	(0.056)
3 $(H = 1, M = 50)$	51.493	2.746	48.027	2.389
	(6.197)	(0.030)	(2.944)	(0.035)
4 $(H = 2, M = 10)$	51.445	2.522	43.664	2.341
	(0.705)	(0.027)	(0.819)	(0.030)
5 $(H = 2, M = 25)$	65.869	2.789	66.003	2.728
	(1.040)	(0.035)	(2.838)	(0.051)
6 $(H = 2, M = 50)$	16.088	0.875	19.222	0.869
	(0.484)	(0.017)	(1.595)	(0.021)
7 $(H = 3, M = 10)$	8.864	0.979	11.826	0.957
	(0.208)	(0.020)	(0.398)	(0.020)
8 $(H = 3, M = 25)$	8.543	0.867	11.729	0.864
	(0.246)	(0.018)	(0.864)	(0.021)
9 $(H = 3, M = 50)$	10.284	0.770	14.032	0.779
	(0.368)	(0.017)	(1.113)	(0.020)

Values in the table scaled by 10^{-3}

CDF error is defined as the error between the estimate and the target, where the latter is the empirical CDF, whereas the PMF error is the error between the estimate and the true value. Despite this difference, we find that the CDF approximation using the target variable still leads to good PMF estimates.

We finally present sample squared errors from the CDF and PMF estimates in order to assess the approximation properties of the proposed method over all Monte Carlo simulations. Fig. 4 presents the squared errors ($S2$) of model 8 for each data point and 100 Monte Carlo replications as the difference between the target CDF $\hat{y}_t$ and the estimated CDF $\tilde{y}_t$ ($S2_{\widetilde{CDF}}$) and as the difference between the estimated PMF $\tilde{y}'_t$ and true PMF y'_t ($S2_{PMF}$). The squared errors of CDF and PMF seem to span the whole area, with some extreme errors concentrated in the top-left and bottom-right tails in both subfigures. Similar to Fig. 3, the differences between the CDF and PMF errors in Fig. 4 result from the fact that the former is defined as the difference between the target and estimated CDFs, whereas the latter is defined as the difference between the true and estimated PMFs. Thus, even though we obtain the PMF estimates with analytical differentiation, the locations of extreme CDF errors can be different from the PMF errors. Fig. 4 shows that this is indeed the case: CDF errors are relatively higher in the middle of the distribution, whereas PMF errors are relatively higher in the lower right tail of the distribution. We conjecture that this slight difference in the locations of extreme errors stem from the approximation of the target CDF $\hat{y}_t$ in (8), but the proposed method has

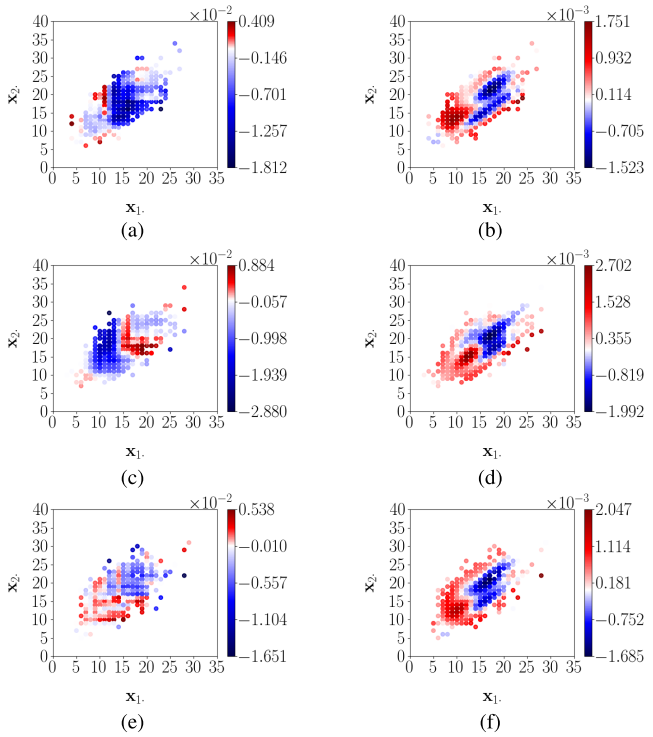

Fig. 3. Sample CDF and PMF errors. (a), (c), and (e) Differences between target and estimated CDF and (b), (d), and (f) differences between true and estimated PMF for Model 8. The results correspond to three Monte Carlo replications leading to the best, average, and worst $L2_{\widetilde{CDF}}$ loss. (a) cdf difference of smallest $L2_{\widetilde{CDF}}$(b) Smallest $L2_{\widetilde{CDF}}$ loss. (c) Average $L2_{\widetilde{CDF}}$ loss. (d) Average $L2_{\widetilde{CDF}}$ loss. (e) Largest $L2_{\widetilde{CDF}}$ loss. (f) Largest $L2_{\widetilde{CDF}}$ loss.

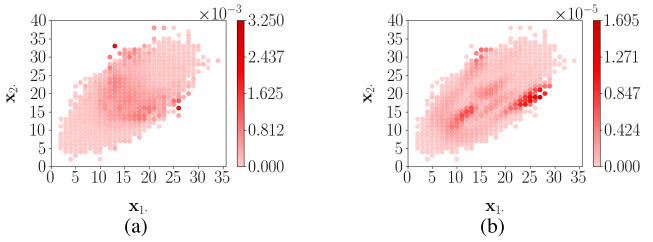

Fig. 4. Sample squared errors. Squared errors $S2_{\widetilde{CDF}}$ and $S2_{PMF}$ differences for Model 8. (a) $S2_{\widetilde{CDF}}$. (b) $S2_{PMF}$.

generally good approximation properties both for the CDF and PMF estimates.

B. Application to Other DGPs

We extend the simulation applications to other DGPs and compare the proposed method with the existing methods when applicable. Section 5 of the Supplementary Material [15] contains these additional results for simulated data from a univariate mixed normal, univariate mixed generalized extreme value, a univariate mixture of Poisson distributions, and bivariate and trivariate mixed normal with small, medium, and large sample sizes. The results are summarized in this section.

The proposed method is applied to two discrete distributions: the univariate mixed Poisson and bivariate mixed Poisson. For small samples, it is observed that the target CDF obtains approximation errors, which results in erratic

estimates. This can be due to the few amounts of distinct event values of a discrete distribution. For larger samples, this approximation error vanishes.

In the Supplementary Material [15], the proposed method is also compared to KDE and PNNs [5]. For univariate simulation cases, the proposed method benefits more from using more observations compared to these methods, by selecting smaller models and obtaining smaller losses. The proposed method generally outperforms KDE in terms of $L2$ losses. For multivariate simulation cases, the superiority of the proposed method holds as long as the MLP contains a sufficient number of neurons. Overall, the approximation errors of the PMFs obtained by KDE are largest around the modes of the distributions. Under the proposed method, on the other hand, the approximation errors are smaller and more equally spread over the entire distribution. The PDF/PMF errors show a clustering pattern flowing from positive/negative to negative/positive loss areas with in between loss values close to 0. The PDF estimates from PNN are in general more stable than those for KDE. A drawback of PNN in these simulations seems to be that an appropriate initial bandwidth has to be specified and that there is a tradeoff between variance and bias by the choice of this bandwidth. It is observed that the $L2$ loss decreases as the size of the bandwidth increases, but at the same time, a bias is introduced, e.g., by introducing a nonsymmetrical distribution around the mode in the case of the mixed normal distribution. The bias decreases as the bandwidth becomes smaller, but at the same time, the $L2$ loss increases resulting in more volatile estimates that are not as smooth as the proposed method.

TABLE II

RESULTS OF ROAD SENSOR DATA WITH A FIXED NUMBER OF HIDDEN NEURONS

$L2_{\widetilde{\text{CDF}}}$ Losses Using the Differences Between the Target and Estimated CDF Obtained by the Proposed Method for the Validation and Test Samples

Model	Validation sample	Test sample
1 $(H = 1, M = 25)$	9.686	8.819
2 $(H = 1, M = 50)$	15.913	12.411
3 $(H = 1, M = 100)$	17.898	12.700
4 $(H = 2, M = 25)$	3.702	4.515
5 $(H = 2, M = 50)$	3.730	3.541
6 $(H = 2, M = 100)$	2.369	3.420
7 $(H = 3, M = 25)$	4.393	5.558
8 $(H = 3, M = 50)$	3.676	5.288
9 $(H = 3, M = 100)$	3.231	3.352
	Values in the table scaled by 10^{-3}	

TABLE III

RESULTS OF ROAD SENSOR DATA WITH A VARYING NUMBER OF HIDDEN NEURONS

$L2_{\widetilde{\text{CDF}}}$ Losses Using the Differences Between the Target and Estimated CDF Obtained by the Proposed Method for the Validation and Test Samples

Model	Validation sample	Test sample
1 $(H = 1, M = 25)$	10.650	9.076
2 $(H = 1, M = 50)$	16.077	13.500
3 $(H = 1, M = 100)$	16.303	17.561
4 $(H = 2, M = 50, 25)$	3.010	5.959
5 $(H = 2, M = 100, 50)$	2.568	4.587
6 $(H = 2, M = 200, 100)$	2.827	2.248
7 $(H = 3, M = 100, 50, 25)$	2.262	2.692
8 $(H = 3, M = 200, 100, 50)$	2.610	2.541
9 $(H = 3, M = 300, 200, 100)$	2.547	7.148
	Values in the table scaled by 10^{-3}	

C. Application to Road Sensor Data

We apply the proposed method to estimate the PMF of traffic intensity, measured as the number of vehicle counts recorded by road sensors. Similar road sensor data are frequently used in the transportation and mobility literature to analyze traffic behavior [24]–[26]. Due to the complex nature of the data, fitting a given PMF distribution can be too restrictive and the flexible NN approach can be beneficial. We estimate the PMF of vehicle counts per minute for 24 h of a full day obtained with a sensor placed on the Dutch Highway A13 on January 11, 2016. The vehicle counts per minute lead to 1440 observations for PMF estimation. We split the data into 1260 training and 240 test observations. For this application, we consider nine different models with a fixed number of hidden neurons for the hidden layers and nine models with a varying number of hidden neurons for the hidden layers. The models with a fixed number of hidden neurons are combinations of $H \in \{1, 2, 3\}$ hidden layers and $M \in \{25, 50, 100\}$ hidden neurons. The models with a varying number of hidden neurons are selected combinations of $H \in \{1, 2, 3\}$ hidden layers and $M \in \{25, 50, 100, 200, 300\}$ hidden neurons. The logistic function is used as an activation function in all layers. For each model, training is done using 10 000 training epochs with early stopping and patience parameter 250, a batch size equal to the test sample size, and a learning rate of 0.0025. As this is a real data application, we use the $L2_{\widetilde{\text{CDF}}}$ as a feasible loss function.

Tables II and III show that the smaller values of $L2_{\widetilde{\text{CDF}}}$ are obtained by models with at least two hidden layers irrespective of the number of selected hidden neurons in the validation and test samples. Hence, the differences between models with a varying number of hidden neurons or a fixed number of hidden neurons are limited. Though for the test sample of table III, models 1 and 9 obtain similar $L2_{\widetilde{\text{CDF}}}$ losses. This highlights that too many hidden neurons are unnecessary in this case. Model 6, from the models with a fixed number of neurons, and model 7, from the models with a varying number of neurons, are selected as the best performing models since they obtain the smallest $L2_{\widetilde{\text{CDF}}}$ for the validation sample.

Figs. 5 and 6 show the CDF and PMF estimates using model 6 with two hidden layers and 100 hidden neurons and model 7 with three hidden layers and 100, 50, and 25 hidden neurons. Both CDFs and PMFs are very similar with a small difference in the skewed shape of the mode around 55 vehicle counts. The target CDFs are highly nonlinear due to the variation in its slope values. The CDF estimates are capable of following the target CDFs very closely. The corresponding PMFs reflect this nonlinearity by the multimodal nature. With both models, a similar PMF for the vehicle counts per minute on this Dutch highway section is obtained. There is a high peak around five vehicle counts. Hence, over the day, there are many minutes at which only a few vehicles are counted on the highway, probably at night. During many minutes of the day, around 25, 55, and 75 vehicles are counted. These

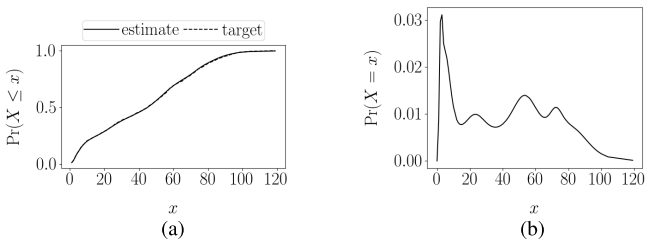

Fig. 5. CDF and PMF estimates of models with a fixed number of hidden neurons Target and estimated CDFs are together with the estimated PMF using the proposed method, with two hidden layers and 100 neurons. (a) CDF. (b) PMF.

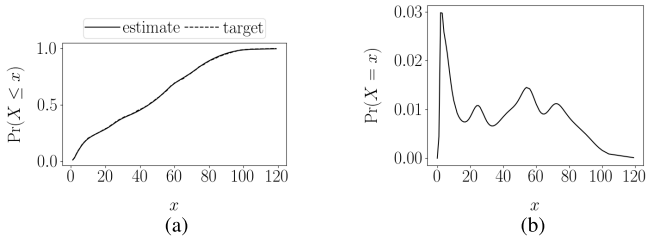

Fig. 6. CDF and PMF estimates of models with a varying number of hidden neurons Target and estimated CDF together with the estimated PMF using the proposed method, with three hidden layers and 100, 50 and 25, neurons. (a) CDF. (b) PMF.

are probably the number of vehicles during rush hour when a traffic jam is detected. The PMF contains a thick right tail, which is probably found just before rush hour when vehicles are still driving at high speed on the highway.

IV. Conclusion

This article proposes a new method to obtain the PDF of a data source, without imposing any assumptions on the DGP, using NN. The algorithm can be applied to any model in the MLP structure, irrespective of the number of input variables, hidden layers, and hidden neurons. How to further improve the models with varying numbers of neurons in the hidden layers is left for future work. Our approach builds on the literature on CDF estimation using NN, where a PDF is obtained by numerical differentiation of the MLP. We extend this literature by providing the analytical derivatives of the obtained CDF from the MLP. Our approach hence removes the numerical approximation error in differentiating the CDF output, leading to more accurate PDF estimates. We show that the proposed solution to obtain the PDF from the CDF output of an NN holds for continuous as well as discrete distributions. Moreover, in line with not imposing any assumptions about the underlying DGP, correlation in the multivariate setting is accounted for, particularly in the simulation examples we provide. Further research is needed to enforce the statistical properties of a CDF through a penalty on violation of a monotone increasing CDF.

The performance of the proposed method is tested in a large simulation study to illustrate that NN can be used to estimate the corresponding CDF and PDF of a range of complex distributions. Among different univariate mixture and multivariate distributions for continuous data, the proposed method is applied to a bivariate mixed Poisson and a univariate mixed Poisson. From this point of view, the proposed method is an extension of methods proposed in the literature since these cannot treat discrete distributions. The PMF differences show a pattern flowing from positive/negative to negative/positive loss areas with in between loss values close to 0 similar to the multivariate normal distributions illustrated in Section 5.4 of the Supplementary Material [15]. Furthermore, the proposed method is applied to road sensor data on Dutch highways. Models with a fixed number and a varying number of hidden neurons show similar results. The resulting PDFs are highly nonlinear characterized by multimodality.

Appendix A
Differentiation of Sigmoid Functions

Two sigmoid functions are considered, the logistic output function denoted by $\boldsymbol{\sigma}(\cdot)$ and the hyperbolic tangent function $\boldsymbol{\tau}(\cdot)$. These functions receive as inputs either a linear combination of input variables $\boldsymbol{x}_{\cdot t}$ together with weights (and biases) or a linear combination of previous layers $\boldsymbol{h}_{\cdot t}^{[q]}$ together with weights (and biases), see the following:

$$\sigma\left(\mathbf{h}_{\cdot t}^{[q]}\right) = \frac{1}{1+e^{-\left[\mathbf{W}^{[q+1]\top}\mathbf{h}_{\cdot t}^{[q]}+\mathbf{b}^{[q+1]}\right]}}$$
$$\tau\left(\mathbf{h}_{\cdot t}^{[q]}\right) = \tanh\left(\mathbf{W}^{[q+1]\top}\mathbf{h}_{\cdot t}^{[q]}+\mathbf{b}^{[q+1]}\right).$$

Note that $\boldsymbol{\sigma}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^N \to \mathbb{R}^M$ and $\boldsymbol{\tau}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^N \to \mathbb{R}^M$ for the input layer and $\boldsymbol{\sigma}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^M \to \mathbb{R}^M$ and $\boldsymbol{\tau}(\mathbf{h}_{\cdot t}^{[q]}) : \mathbb{R}^M \to \mathbb{R}^M$ for the hidden layers. The output layer would take the last hidden layer as input. Then, $\boldsymbol{\sigma}(\mathbf{h}_{\cdot t}^{[H]}) : \mathbb{R}^M \to \mathbb{R}^1$ and $\boldsymbol{\tau}(\mathbf{h}_{\cdot t}^{[H]}) : \mathbb{R}^M \to \mathbb{R}^1$ are operations that map $\mathbf{h}_{\cdot t}^{[H]}$ by a linear combination to the output layer that consists of only one neuron. In the simulation study, no activation functions are imposed on the last layer. See [27] for the derivations of the Nth derivative of the logistic function

$$\boldsymbol{\sigma}^{(n)} = \sum_{k=1}^{n}(-1)^{k-1}A_{n,k-1}\boldsymbol{\sigma}^k(1-\boldsymbol{\sigma})^{n+1-k}$$
$$A_{n,k-1} = \sum_{l=0}^{k}(-1)^l\binom{n+1}{l}(k-l)^n \tag{22}$$

where $A_{n,k-1}$ gives the formula for the Eulerian number. See [28] for the derivations of the Nth derivative of the hyperbolic tangent function

$$\boldsymbol{\tau}^{(n)} = (-2)^n(\boldsymbol{\tau}+1)\sum_{k=0}^{n}\frac{k!}{2^k}\left\{{n \atop k}\right\}(\boldsymbol{\tau}-1)^k$$
$$\left\{{n \atop k}\right\} = \frac{1}{k!}\sum_{l=0}^{k}(-1)^l\binom{k}{l}(k-l)^n \tag{23}$$

where $\left\{{n \atop k}\right\}$ are the Stirling numbers of the second kind. This is incorporated similarly for $\boldsymbol{\sigma}(\cdot)$ explained in Section II-C.

There is a crucial difference between the logistic function and the hyperbolic tangent function though. The hyperbolic tangent function is centered around 0, whereas the logistic function is centered around $(1/2)$. This results in differences

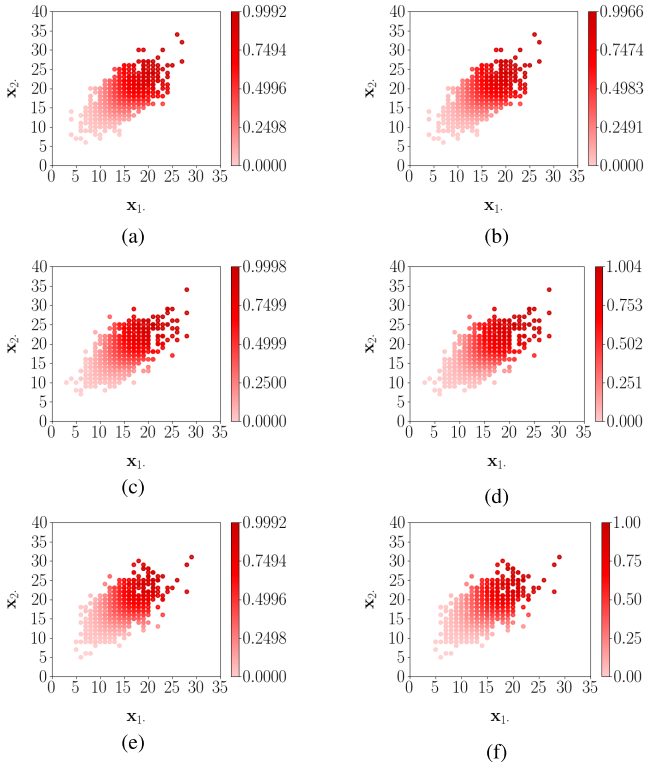

Fig. 7. Best, average, and worst CDF estimates. Target and estimated CDF using the proposed method for mixed Poisson data. The best, average, and worst estimates in terms of $L2_{\widetilde{CDF}}$ loss values out of 100 simulation replications for Model 8. (a), (c), and (e) Target CDF. (b), (d), and (f) Estimated CDF.

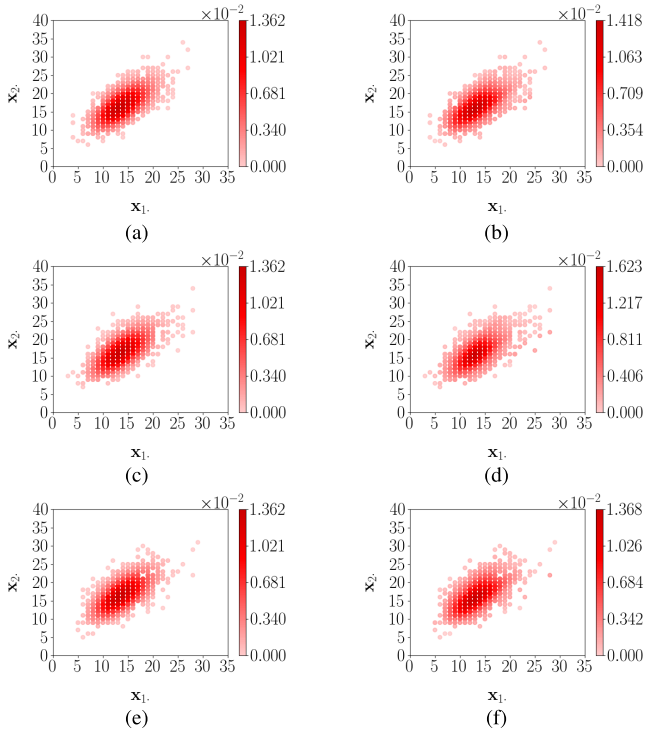

Fig. 8. Best, average, and worst PMF estimates. True and estimated PMF using the proposed method for mixed Poisson data. The best, average, and worst estimates in terms of $L2_{\widetilde{CDF}}$ loss values out of 100 simulation replications for Model 8. (a), (c), and (e) True PMF. (b), (d), and (f) Estimated PMF.

in the value of the gradient of the activation function, which decides upon the speed of training. Activation functions that are centered around 0 result in faster convergence of the algorithm. The output of each hidden layer will be centered around 0, hence some positive values and some negative values. When the gradient needs to change direction, learning is easier than when all output values are either positive or negative. Hence, the hyperbolic tangent function explores more areas of the loss function than the logistic function does. This can result in obtaining the global minimum, in which a logistic function does not reach due to the starting value. However, this could potentially also result in skipping this global minimum because of greater gradients.

APPENDIX B
BIVARIATE MIXED POISSON DISTRIBUTION FIGURES

This section presents the CDF and PMF estimates of the simulation study in detail. Fig. 7(a) and (b) presents the best CDF estimate and target CDF in terms of smallest $L2_{\widetilde{CDF}}$ loss. This implies that Fig. 3(a) corresponds to the difference of Fig. 7(a) and 7(b). This is also done for the average CDF estimate and target CDF in Fig. 7(c) and (d) that correspond to Fig. 3(c) and the worst CDF estimate and target CDF in Fig. 7(e) and (f) that correspond to Fig. 3(e). Fig. 8(a) and (b) presents the best PMF estimate and true PMF in terms of smallest $L2_{\widetilde{CDF}}$ loss. This implies that Fig. 3(b) corresponds to the difference of Fig. 8(a) and 8(b). This is also done for the average PMF estimate and true PMF in Fig. 8(c) and (d) that correspond to Fig. 3(d) and the worst PMF estimate and true PMF in Fig. 8(e) and (f) that correspond to Fig. 3(f). As can be retrieved from these figures, the differences between the target/true and estimated CDF/PMF are negligible.

ACKNOWLEDGMENT

The authors would like to thank Prof. Edmondo Trentin from the University of Siena, Siena, Italy, and two anonymous referees for carefully reading a former draft of the manuscript and providing constructive comments that proved to be very helpful to improve this article. The views expressed in this article are those of the authors and do not necessarily reflect the policy of Statistics Netherlands.

REFERENCES

[1] H. Cramér, *Mathematical Methods of Statistics*. Princeton, NJ, USA: Princeton Univ. Press, 1946.
[2] J. R. Magnus, "The asymptotic variance of the pseudo maximum likelihood estimator," *Econ. Theory*, vol. 23, no. 5, pp. 1022–1032, 2007.
[3] S. R. Eliason, *Maximum Likelihood Estimation: Logic and Practice*, no. 96. Newbury Park, CA, USA: Sage, 1993.
[4] J. S. Cramer, *Econometric Applications of Maximum Likelihood Methods*. Cambridge, U.K.: CUP Archive, 1989.
[5] E. Trentin, L. Lusnig, and F. Cavalli, "Parzen neural networks: Fundamentals, properties, and an application to forensic anthropology," *Neural Netw.*, vol. 97, pp. 137–151, Jan. 2018.

[6] Y. Nakamura and O. Hasegawa, "Nonparametric density estimation based on self-organizing incremental neural network for large noisy data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 1, pp. 8–17, Jan. 2017.
[7] T. Hu, Q. Guo, Z. Li, X. Shen, and H. Sun, "Distribution-free probability density forecast through deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 612–625, Feb. 2020.
[8] L. Bu, C. Alippi, and D. Zhao, "A pdf-free change detection test based on density difference estimation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 2, pp. 324–334, Feb. 2016.
[9] Y. Cao, H. B. He, and H. Man, "SOMKE: Kernel density estimation over data streams by sequences of self-organizing maps," *IEEE Trans. Neural Netw., Learn. Syst.*, vol. 23, no. 8, pp. 1254–1268, Aug. 2012.
[10] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.
[11] D. G. Stork, R. O. Duda, P. E. Hart, and D. Stork, *Pattern Classification*. Hoboken, NJ, USA: Wiley, 2001.
[12] M. Magdon-Ismail and A. Atiya, "Density estimation and random variate generation using multilayer networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 3, pp. 497–520, May 2002.
[13] S. Zhang, "From CDF to PDF—A density estimation method for high dimensional data," 2018, *arXiv:1804.05316*.
[14] C. Z. Mooney, *Monte Carlo Simulation*, no. 116. Newbury Park, CA, USA: Sage, 1997.
[15] D. E. W. Peerlings, J. A. Van Den Brakel, N. Baştürk, and M. J. H. Puts. *Supplementary File to the Paper: Multivariate Density Estimation by Neural Networks*. [Online]. Available: http://www.tobeedited.com
[16] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, vol. 1, no. 10. New York, NY, USA: Springer, 2001.
[17] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras
[18] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org/
[19] F. Faà di Bruno, "Sullo sviluppo delle funzioni," *Annali di Scienze Matematiche e Fisiche*, vol. 6, pp. 479–480, Dec. 1855.
[20] L. F. A. Arbogast, *Du Calcul des Dérivations*. Strasbourg, France: Levrault, Frères, 1800.
[21] M. Hardy, "Combinatorics of partial derivatives," *Electron. J. Combinatorics*, vol. 13, no. 1, pp. 1–13, Jan. 2006.
[22] R. Wilson and J. J. Watkins, *Combinatorics: Ancient & Modern*. Oxford, U.K.: OUP, 2013.
[23] K. Shin and R. Pasupathy, "A method for fast generation of bivariate Poisson random vectors," in *Proc. Winter Simulation Conf.*, 2007, pp. 472–479.
[24] E. L. Manibardo, I. Lana, and J. D. Ser, "Deep learning for road traffic forecasting: Does it make a difference?" *IEEE Trans. Intell. Transp. Syst.*, early access, Jun. 7, 2021, doi: 10.1109/TITS.2021.3083957.
[25] D. Ma, X. Song, and P. Li, "Daily traffic flow forecasting through a contextual convolutional recurrent neural network modeling inter-and intra-day traffic patterns," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 5, pp. 2627–2636, May 2020.
[26] H. Lee, J. Lee, and Y. Chung, "Traffic density estimation using vehicle sensor data," *J. Intell. Transp. Syst.*, to be published, doi: 10.1080/15472450.2021.1966626.
[27] A. A. Minai and R. D. Williams, "On the derivatives of the sigmoid," *Neural Netw.*, vol. 6, no. 6, pp. 845–853, 1993.
[28] K. N. Boyadzhiev, "Derivative polynomials for tanh, tan, sech and sec in explicit form," *Fibonacci Quart.*, vol. 45, no. 4, pp. 291–303, 2007.

Dewi E. W. Peerlings received the bachelor's and master's degrees in econometrics and operations research from the School of Business and Economics, Maastricht University, Maastricht, The Netherlands, in 2016 and 2018, where she is currently pursuing the Ph.D. degree in collaboration with Statistics Netherlands, Heerlen, The Netherlands.

Her current research interests include neural networks, time series, and state-space models.

Jan A. van den Brakel is currently a Senior Statistician with the Methodology Department, Statistics Netherlands, Heerlen, The Netherlands, and an Extended Professor of survey methodology with the Department of Quantitative Economics, School of Business and Economics, Maastricht University, Maastricht, The Netherlands. His research areas include sampling, design and analysis of experiments, small area estimation, time series analysis, and the use of nonprobability data for official statistics.

Nalan Baştürk is currently an Associate Professor with the Department of Quantitative Economics, School of Business and Economics, Maastricht University, Maastricht, The Netherlands. Her research areas include econometric modeling, time series analysis, and Bayesian inference with a wide area of applications.

Marco J. H. Puts is currently a Methodologist and a Data Scientist with Statistics Netherlands, Heerlen, The Netherlands. His research focuses on the usability of big data and machine learning in official statistics.