Karlsruhe Institute of Technology

High Performance Humanoid Technologies

Conditional Density Estimation with Normalizing Flows

Bachelor's Thesis
of

Simon Böhm

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
High Perfomance Humanoid Technologies Lab (H^2T)

Referees: Prof. Dr.-Ing. Tamim Asfour

Advisors: Jonas Rothfuss

Duration: May 27th, 2019 – August 26th, 2019

Erklärung:

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den August 26th, 2019

Simon Böhm

Abstract:

Quantifying the complex uncertainty associated with a prediction is important in many machine learning tasks. In conditional density estimation (CDE), the conditional density of the output given the input variables is modeled. We propose the Normalizing Flow Network (NFN), a high-capacity, neural network based model for CDE. By utilizing Normalizing Flows, a novel method of parametrizing densities, the NFN is capable of estimating complex conditional distributions. However, the NFN's expressiveness leads to increased overfitting, particularly on small datasets. We review common neural network regularization techniques and show how, due to the model's structure, they are inadequate when used in the context of CDE. To address these shortcomings, we develop a model-agnostic noise regularization approach that adds noise to the data during training. We derive how the proposed regularization introduces an inductive bias towards smooth distributions. Across 8 datasets, we demonstrate that this regularization approach outperforms existing methods. The NFN, combined with noise regularization, makes for a powerful novel CDE model capable of capturing complex predictive uncertainties, even in scenarios where data is scarce.

Kurzzusammenfassung:

Die Quantifizierung der mit einer Vorhersage verbundenen Unsicherheit ist zentraler Bestandteil vieler Methoden und Anwendungen des maschinellen Lernens. Bei der bedingten Wahrscheinlichkeitsschätzung (BWS) wird die bedingte Wahrscheinlichkeit der Ausgabevariablen anhand der Eingabevariablen modelliert. Wir stellen das Normalisierende Ströme Netzwerk (NSN) vor, ein leistungsfähiges, auf neuronalen Netzen basierendes Modell für BWS. Durch die Verwendung von Normalisierenden Strömen, einer neuartigen Methode zur Modellierung von Wahrscheinlichkeitsdichten, ist das NSN in der Lage, komplexe bedingte Wahrscheinlichkeiten anzunähern. Die Ausdrucksstärke des NSN führt jedoch zu vermehrter Überanpassung, insbesondere auf kleinen Datensätzen. Wir untersuchen gängige Techniken zur Regularisierung neuronaler Netze und zeigen, wie diese, aufgrund der inhärenten Modellstruktur, im Rahmen von BWS unzureichend sind. Um diese Mängel zu beheben, entwickeln wir eine Methode zur modellunabhängigen Rauschregularisierung. Wir zeigen, dass durch leichtes Verrauschen der Daten während des Trainings glattere Verteilungen gelernt werden und somit Überanpassung verhindert werden kann. Anhand von acht Datensätzen zeigen wir, dass der vorgestellte Regularisierungsansatz existierende Methoden in Effektivität übertrifft. Das NSN, kombiniert mit der Rauschregularisierung, bildet ein leistungsfähiges, neuartiges BWS-Modell, das in der Lage ist, komplexe bedingte Wahrscheinlichkeiten zu modellieren, auch wenn nur wenige Lerndaten zur Verfügung stehen.

Contents

1. Introduction

Conditional density estimation (CDE) generalizes regression by modelling the full conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ instead of just the mean $\mathbb{E}[\boldsymbol{y}|\boldsymbol{x}]$. It provides a way of precisely quantifying the uncertainty associated with a prediction. CDE is helpful in situations where the mean provides a poor representation of the conditional distribution, e.g. in the face of multimodality. Among others, it has been used successfully for predicting return densities in finance (Engle, 2001), treatment effects in medicine (Strobl and Visweswaran, 2019) and value functions in reinforcement learning (Bellemare et al., 2017). Of particular interest to us is parametric CDE, where the conditional is assumed to be a member of a family of distributions $C = \{p(\cdot|\theta)|\theta \in \Theta\}$, with θ being the finite parameter vector that specifies the respective distribution.

Normalizing Flows (NFs) are one such distributional family that has recently received much interest in machine learning. For instance, they have been used successfully for estimating high dimensional densities, like those of images (Kingma and Dhariwal, 2018), sounds (van den Oord et al., 2016a) and posteriors of probabilistic models (Kingma et al., 2016). By defining a transformation of a simple base density through learnable, invertible mappings, NFs provide a novel way of parametrizing distributions. They enable the modelling of non-Gaussian characteristics such as multimodality, heavy tails and skewness. Previous research focuses on NFs for unconditional density estimation. We generalize and show how to use NFs in the context of CDE. In this vein, we present the Normalizing Flow Network (NFN), a neural network based, parametric CDE model using NFs as its density model. Given enough capacity, NFNs allow us to model highly complex conditional densities.

In this work we focus especially on real-world regression datasets, where often only little data is available. This increases the danger of overfitting for high-capacity CDE models, like the NFN, making appropriate regularization necessary. Standard regularization techniques for neural networks are difficult to analyze in the context of CDE. This is due to the neural network having only an indirect influence on the output of the full model as it merely specifies the parameters of the output distribution and not the output itself. As a solution, we propose a noise regularization method that adds small random perturbations to the data during training. Intuitively, this slightly smears the datapoints and makes overfitting more difficult. We derive, how noise regularization introduces favourable inductive biases in CDE by incentivizing smooth conditional distributions. We compare noise regularization to common regularization techniques, like weight decay (Hanson and Pratt, 1989) and L1/L2 penalties. In addition, we also investigate the effectiveness of a Bayesian treatment of the problem. This involves interpreting the learnable parameters of the NN as random variables. We can now place highly configurable prior distributions, that have a regularizing effect, on the weights and biases of NN. We fit the resulting Bayesian Neural Network (BNN, Neal (2012)) using variational inference (Blei et al., 2017). Out of the mentioned methods, noise regularization is the only technique that works in the data space, irrespective of the model's parameters, making it model agnostic.

In our experiments, we compare the performance of the NFN to two other CDE models and assess the usefulness of noise regularization relative to the parameter-space regularization techniques. This is done on 8 different datasets, ranging from simulated densities to real-world financial data. We show that the NFN performs favourably for modelling non-Gaussian conditional densities. Moreover, we demonstrate the superior performance of the proposed noise regularization method across models and datasets. Combined, the NFN, regularized with additive noise, makes for a powerful and novel CDE model that can be effectively applied in many scenarios, even on small datasets with highly non-Gaussian properties.

The code for the Normalizing Flow Network has been published as part of an open-source Python package for CDE[1], as well as in a standalone repository[2]. The work presented in this thesis has contributed to a publication on noise regularization for conditional density estimation (Rothfuss et al., 2019).

The thesis is structured as follows: We begin by formally introducing the concepts of density estimation and conditional density estimation. In chapter 3, we move on to presenting normalizing flows and explaining the structure of the Normalizing Flow Network. In chapter 4, we cover regularization, first outlining the more common techniques, then describing Bayesian neural networks and finally introducing our noise regularization approach. An empirical evaluation of the CDE models as well as the regularization schemes is presented in chapter 5. Finally, we wrap-up the thesis in chapter 6 and 7 with an outlook onto further work and a conclusion.

[1]`https://github.com/freelunchtheorem/Conditional_Density_Estimation`
[2]`https://github.com/siboehm/NormalizingFlowNetwork`

2. Background

2.1. Density Estimation

Density estimation is concerned with estimating the probability density function (PDF) p($\boldsymbol{x}$) of a random variable X. For most real world problems, this density function is unknown and only samples $\boldsymbol{x}_i \sim p(\boldsymbol{x})$ can be observed. The aim of density estimation is then to use a finite set $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ of independent and identically distributed (i.i.d.) samples to find an estimate $\hat{p}(\boldsymbol{x})$ of the true density $p(\boldsymbol{x})$. Finding the distribution that generated a dataset without further making assumptions is an ill-posed problem. Any PDF that is non-zero at each $\boldsymbol{x}_i$ could have given rise to the observations (Bishop, 2006, p. 67). Hence, further assumptions about the structure of $p(\boldsymbol{x})$ have to be made. Parametric and non-parametric density estimation are two different approaches for imposing necessary assumptions and solving this problem.

2.1.1. Parametric Density Estimation

In parametric density estimation, the general problem is simplified by assuming that $\hat{p}(\boldsymbol{x})$ belongs to a family of probability densities $F = \{\hat{p}_\theta(\cdot)|\theta \in \Theta\}$, specified by a finite-dimensional parameter vector θ. An example might be the family of multivariate Gaussians with the density function:

$$\mu \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}, \Sigma \succeq 0 \quad \mathcal{N}(\boldsymbol{x}|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\mu)^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x}-\mu)\right) \tag{2.1}$$

where a particular PDF can be specified using $\theta = (\mu, \Sigma)$ with the mean vector μ and covariance matrix Σ. In a frequentist setup, the optimal parameter vector θ^* is determined by maximizing some function $f(\theta, D)$, for example the likelihood function

$$L(\theta, D) = \prod_{n=1}^{N} \hat{p}_\theta(\boldsymbol{x}_n) \tag{2.2}$$

This results in the optimal parameter θ^*,

$$\theta^* = \arg\max_\theta L(\theta, D) = \arg\max_\theta \sum_{n=1}^{N} \log \hat{p}_\theta(\boldsymbol{x}_n) \tag{2.3}$$

where the computation in log-space is often numerically preferable. This can be viewed as empirical approximation to minimizing the Kullback-Leibler-divergence (KL-divergence) between $p(\boldsymbol{x})$ and $p_\theta(\boldsymbol{x})$ (Bishop, 2006),

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \sim p(\boldsymbol{x}) \quad \mathbb{E}_{p(\boldsymbol{x})}[f] \simeq \frac{1}{N} \sum_{n=1}^{N} f(\boldsymbol{x}_n) \tag{2.4}$$

$$D_{KL}(p||\hat{p}_\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right] \simeq \frac{1}{N} \sum_{n=1}^{N} \log(p(\boldsymbol{x})) - \log(p_\theta(\boldsymbol{x})) \tag{2.5}$$

where only the second term in (2.5) is dependent on θ and needs to be minimized.

A drawback of parametric density estimation is that if the true density $p(\boldsymbol{x})$ is not in the density family F, then the parametrized density $p_\theta(\boldsymbol{x})$ will never fully converge, even if infinitely many datapoints are provided.

2.1.2. Nonparametric Density Estimation

Nonparametric density estimation, on the other hand, does not explicitly restrict the family of distributions. Rather, it imposes implicit smoothness assumptions onto the density estimate. Popular nonparametric density estimators include nearest-neighbour methods such as histograms and kernel density estimators (KDE). The latter involves placing a kernel, a non-negative, normalized function, around every observed datapoint (Parzen (1962), Rosenblatt (1956)). A common choice is the Gaussian kernel:

$$K(\boldsymbol{x}) = \frac{1}{(2\pi)^{-n/2}} \exp\left(-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}\right) \tag{2.6}$$

The resulting distribution is then the equally weighted, normalized mixture of the N kernels, each centered in a data point $\boldsymbol{x}_n$:

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) \tag{2.7}$$

In that, h is the bandwidth parameter which controls the regularization of the density estimator. When using a Gaussian kernel, h can be interpreted as the standard deviation. The bandwidth parameter allows trading off between smoothness (underfitting) and flexibility (overfitting) of the resulting density function. Properly setting this parameter is crucial to producing an estimator that fits the data well and ensures asymptotic convergence to the true density (Li and Racine (2007), Devroye (1983)). Because this technique uses a separate kernel for every observed datapoint, the computational burden of evaluating the density grows linearly with the amount of data available. Furthermore, nonparametric methods often generalize poorly when applied to higher-dimensional problems since increasing the dimensions makes the datasets inherently sparse and reduces the information gained through measuring the distance between points.

2.2. Conditional Density Estimation

Estimating the conditional density of the target variables, given some input variables, is the core problem of supervised machine learning. More formally, let X, Y be two random variables with realizations $\boldsymbol{x}$, $\boldsymbol{y}$ and the conditional probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$. The aim of conditional density estimation (CDE) is to find an estimate of the unknown, true PDF $p(\boldsymbol{y}|\boldsymbol{x})$ given a dataset $D = \{(\boldsymbol{x}_1,\boldsymbol{y}_1),\ldots,(\boldsymbol{x}_n,\boldsymbol{y}_n)\}$ sampled i.i.d. from the joint probability distribution $p(\boldsymbol{x},\boldsymbol{y})$.

Most work in the machine learning literature focusses on parametric CDE. This approach can be formalized by choosing:

1. a family of parametric densities $C = \{\hat{p}(\cdot|\theta)|\theta \in \Theta, \hat{p}(\cdot|\theta) : Y,\Theta \to [0,1]\}$
2. a family of functions $H = \{h_\omega(\cdot)|\omega \in \Omega, h_\omega(\cdot) : X \to \Theta\}$
3. an inference procedure $I(\cdot)$ where $I(D, p_{prior}) = p_{post}$ with p_{prior} being the prior distribution and p_{post} being the posterior distribution over the parameters.

Given a sample ω from the posterior distribution p_{post}, the approximated conditional probability distribution takes the form $\hat{p}(\boldsymbol{y}|h_\omega(\boldsymbol{x}))$. The function $h_\omega(\cdot)$ maps a sample $\boldsymbol{x}$ to a parameter vector $\boldsymbol{\theta}$ which then specifies the distribution over $\boldsymbol{y}$. If the families C and H are picked to be more expressive, this often reduces the speed of optimization and increases the danger of overfitting.

For now, we will focus on the common maximum likelihood approach in its most basic form, where the prior p_{prior} is a uniform distribution over the parameter space and the posterior p_{post} is a dirac delta function with center point ω_{ML}. ω_{ML} is set to maximize the conditional likelihood:

$$\omega_{ML} = \arg\max_{\omega}\prod_{n=1}^{N}\hat{p}\left(\boldsymbol{y}_n|h_\omega(\boldsymbol{x}_n)\right) = \arg\max_{\omega}\sum_{n=1}^{N}\log\hat{p}(\boldsymbol{y}_n|h_\omega(\boldsymbol{x}_n)) \tag{2.8}$$

The common regression algorithm implicitly performs maximum likelihood CDE. It is a special case of the above framework where only the mean of the conditional distribution is being modeled: Let g_ω be a regression model outputting a point estimate $g_\omega(\boldsymbol{x}) = \mathbb{E}[\boldsymbol{y}|\boldsymbol{x}] = \hat{\boldsymbol{y}}, \boldsymbol{x} \in \mathbb{R}^{d_1}; \boldsymbol{y}, \hat{\boldsymbol{y}} \in \mathbb{R}^{d_2}$, for example a neural network. Setting ω_{ML} by minimizing the mean-squared-error (MSE)

$$\omega_{ML} = \arg\min_{\omega} \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 \tag{2.9}$$

is equivalent to maximizing the log-likelihood under the assumption of a Gaussian conditional distribution, where the output of the model is the mean of the Gaussian,

$$\omega_{ML} = \arg\min_{\omega} \sum_{n=1}^{N} \log \mathcal{N}(\boldsymbol{y}_n | g_\omega(\boldsymbol{x}_n), I) = \arg\min_{\omega} \sum_{n=1}^{N} \frac{1}{2} (\boldsymbol{y}_n - g_\omega(\boldsymbol{x}_n))^2 \tag{2.10}$$

using the definition in equation 2.1. It can be seen that the CDE model given in 2.10 and the standard regression model given in 2.9 will yield the same result.

2.2.1. Characteristics of the conditional distributions

Our aim is to build a model that is capable of modelling complex conditional distributions. Three characteristics of particular importance to us are multimodality, heteroscedasticity and complex tail behaviour. These can be observed in real-world datasets. A Gaussian conditional distribution, as assumed by many common regression models, fails to model either of those characteristics.

Multimodality

Multimodality refers to the PDF having multiple local maxima, called modes. This can result in the mean being an insufficient representation of the overall distribution. Detection of those modes has been studied extensively in density estimation (Silverman, 1981). A real-world example is the distribution of the salaries of first-year lawyers seen in figure 2.1, which is strongly bi-modal and has much of the probability mass away from the mean. Reducing this distribution down to its mean is not representative. We will consider distributional families C that can represent multimodality.

Heteroscedasticity

For a heteroscedastic distribution, the conditional variance $\mathrm{Var}(Y|X)$ is not constant. The opposite, homoscedasticity, is an assumption often made in the context of regression models like linear regression. The variance of the PDF outputted by the model can be interpreted as a measure of uncertainty about the actual value of the dependent variable Y given the input variable X. Using a family of conditional distributions that allows for changing variance would then enable the model to output changes in uncertainty over the input space, in effect, signalling the amount of uncertainty about the prediction. The dataset in figure 2.2 exhibits heteroscedasticity, with the variance of y being higher for $x \leq 0$. Heteroscedasticity can be observed in many finance datasets, where the variance of returns often strongly varies over time (Damodar, 2004). As a countermeasure, financial models, like the ARMA-GARCH method, have been developed to model the mean and variance of a Gaussian distribution (Hamilton (1994), Engle (1982)). Yet those models still make strong assumptions about the characteristics of the data, e.g. assuming unimodality, and it is unclear how consistent these assumptions are with the empirical data.

Complex tail behaviour

The tail of a distribution refers to the parts of the PDF that are far away from the mean. The behaviour of such outliers can strongly influence the usefulness of an estimate. Of specific interest to us are fat-tailed distributions since their characteristics have been observed in manys fields, from wage distribution to meteorology and finance (Mandelbrot, 2001). They are characterized by their tails decaying according

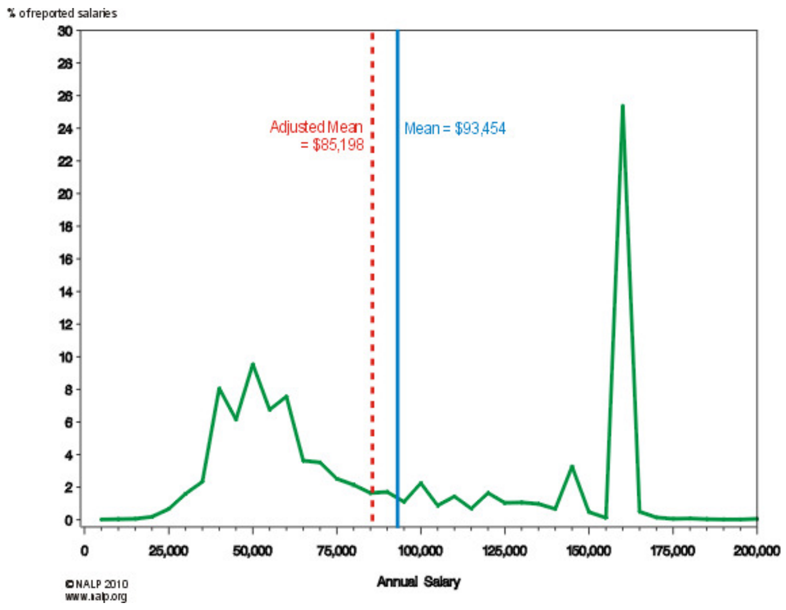

Figure 2.1.: Distribution of annual salaries earned by first-year lawyers (in green). Source: NALP (2010)
.

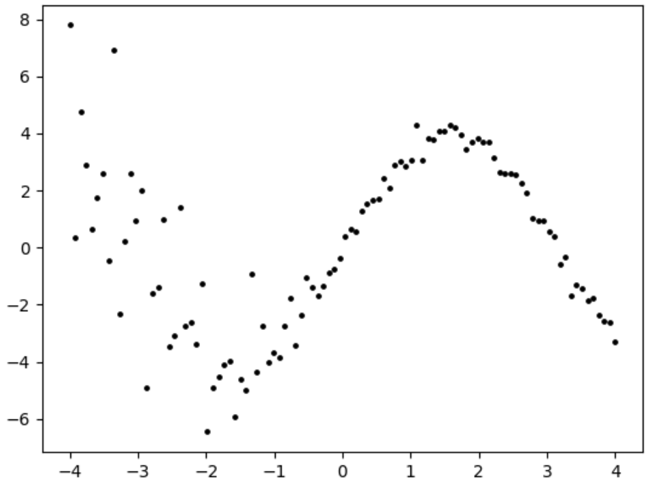

Figure 2.2.: An example dataset, generated by adding heteroscedastic Gaussian noise, to a sinusoid. The variance of Y is higher for $X \leq 0$.

to the power-law $x^{-\alpha}$, especially for $\alpha \leq 2$. Distributions like this have a significant amount of their mass located away from the mean of the distribution. This results in outliers being much more likely than under a Normal distribution, where the tails decay according to e^{-x^2}. One example of a fat-tailed distribution is the Cauchy distribution,

$$a \in \mathbb{R}, b \in \mathbb{R} \quad p(x|a,b) = \left\{ \pi b \left[1 + \left(\frac{x-a}{b} \right)^2 \right] \right\}^{-1} \tag{2.11}$$

where a is the location parameter and b is the scale parameter (Forbes et al., 2011). In figure 2.3, the cauchy distribution is plotting against a normal distribution, highlighting the heavy tails. Given a enough flexibility within the parametric family C, our estimators can model complex and fat tail behaviour.

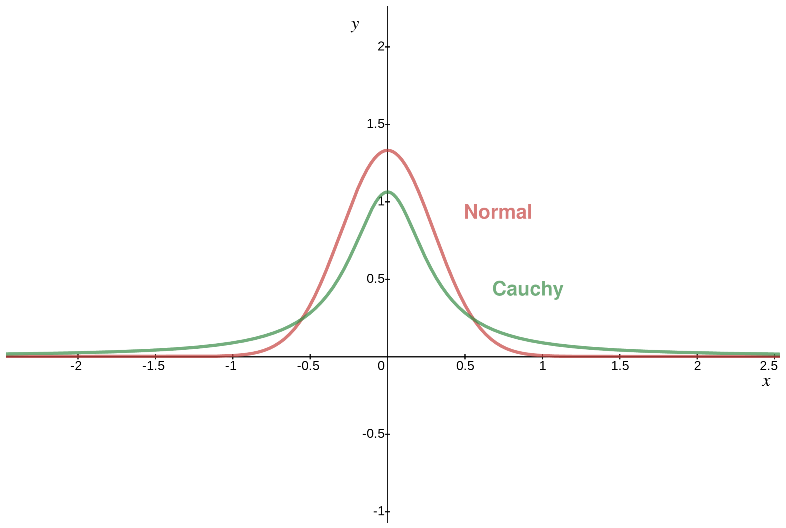

Figure 2.3.: A Cauchy distribution (green) and Normal distribution (red). The Cauchy distribution has heavier tails..

3. Normalizing Flow Network

The Normalizing Flow Network (NFN) is a novel CDE model designed for accurately estimating low-dimensional, highly non-Gaussian conditional densities. It builds upon the generic architecture presented in section 2.2. As the family of functions we employ neural networks. For our family of densities we use normalizing flows. By building our model as the combination of these two expressive families, we can model arbitrarily complex conditional densities and cover all distributional characteristics listed in section 2.2.1. Whereas most of the recent research on normalizing flows has focused on high-dimensional densities, we focus on flows that work well on low-dimensional data. We begin by introducing the distributional family of normalizing flows, before outlining the structure of the NFN.

3.1. Normalizing Flows

Normalizing Flows (Tabak and Turner (2013), Tabak and Vanden-Eijnden (2010)) were originally developed for density estimation. They work by transforming a random variable Z_0, distributed according to a simple base distribution, through a series of invertible mappings $f = (f_1, \dots, f_N)$, which results in a transformed random variable Z_N with a potentially much more complex distribution. In our experiments, the base distribution is set to a multivariate Gaussian:

$$Z_0 \sim \mathcal{N}(0, I) \tag{3.1}$$

$$f = (f_1, \dots, f_N), f_n : \mathbb{R}^d \to \mathbb{R}^d, \exists f_n^{-1} : \forall x \in \mathbb{R}^d : f^{-1}(f(x)) = x \tag{3.2}$$

$$Z_N = f_N \circ \dots \circ f_1(Z_0) \tag{3.3}$$

This setup can already be used for sampling. To sample $z_N \sim p(Z_N)$, one can draw a sample $z_0 \sim \mathcal{N}(0, I)$ and transform it using $f_N(\dots f_1(z_0)) = z_N$, which results in a sample from the more complex, transformed distribution. To evaluate the probability density of a sample transformed through a single flow $f_1(z_0)$, the change-of-variable formula is applied:

$$p(z_1) = p(f_1^{-1}(z_1)) \left| \det \frac{\partial f_1^{-1}}{\partial z_1} \right| = p(z_0) \left| \det \frac{\partial f_1}{\partial z_0} \right|^{-1} \tag{3.4}$$

Intuitively, since the absolute determinant of the Jacobian of a function corresponds to the change of volume incurred through the function, it's inverse can be used to renormalize. Eq. 3.4 can be deducted using the inverse-function theorem $J_{f^{-1}}(f(\boldsymbol{x})) = J_f(\boldsymbol{x})^{-1}$ for invertible functions f and the equation $\det(A)\det(A^{-1}) = \det(AA^{-1}) = 1 = \det(A)\det(A)^{-1}$. It can be seen that large gradients of f lead to a small $p(z_1)$, stretching out the density. Small gradients, on the other hand, have a compressing effect. By compositing normalizing flows, a new, more expressive flow can be built. Generalizing equation 3.4 to a chain of N flows and moving to log space gives:

$$\log p(z_N) = \log p(z_0) - \sum_{n=1}^{N} \log \left| \det \frac{\partial f_n}{\partial z_{n-1}} \right| \tag{3.5}$$

The computational cost grows linearly with the amount of flows used. To enable fast calculation of likelihoods, the inverse log determinant of the Jacobian (ILDJ) needs to be efficiently computable. Any function that is invertible and differentiable with a strictly non-zero derivative can be used as a normalizing flow. To be useful in density estimation, the flow needs to be parametrized and differentiable with regards to the parameters. Many such functions have been proposed. Due to their invertibility, the flows are also called bijectors. The NFs shown here are, when properly combined, able to model all three desirable characteristics of conditional densities mentioned in section 2.2.1.

3.1.1. Affine Flows

The affine flow is the simplest normalizing flow, apart from the identity bijector. It performs a scale-and-shift operation on the underlying density and is easily invertible. This flow is given by

$$\boldsymbol{a},\boldsymbol{b} \in \mathbb{R}^d \quad f(\boldsymbol{z}) = \exp(\boldsymbol{a}) \odot \boldsymbol{z} + \boldsymbol{b} \tag{3.6}$$

$$f^{-1}(\boldsymbol{z}) = \exp(\boldsymbol{a})^{-1} \odot (\boldsymbol{z} - \boldsymbol{b}) \tag{3.7}$$

$$\log \left| \det \frac{\partial f}{\partial \boldsymbol{z}} \right|^{-1} = -\sum_{i=1}^{D} a_i \tag{3.8}$$

Where $\odot$ represents element-wise multiplication. Every distribution in the location-scale family can be represented as an affine flow transforming the base distribution of the family, where the base distribution has zero mean and uniform variance. Using a univariate normal distribution as an example, let f_θ be an affine flow with $\theta = \{a, b\}$. The standard normal distribution, transformed into a normal distribution with mean b and variance e^{2a} is given by

$$\boldsymbol{x}_0 \sim \mathcal{N}(0,1), \boldsymbol{x}_1 = f_\theta(\boldsymbol{x}_0) \tag{3.9}$$

$$p(\boldsymbol{x}_1) = p(f_\theta^{-1}(\boldsymbol{x}_0))e^{-a} = \frac{1}{\sqrt{2\pi e^{2a}}} \exp\left(\frac{-(\boldsymbol{x}_0 - b)^2}{2e^{2a}}\right) \tag{3.10}$$

In figure 3.1, we can observe a 2D standard Gaussian (left) being transformed through an affine flow, resulting in the density being shifted and scaled (right). While this flow is simple and has a fast-to-compute ILDJ, it doesn't change the underlying density family and cannot model multimodal PDFs.

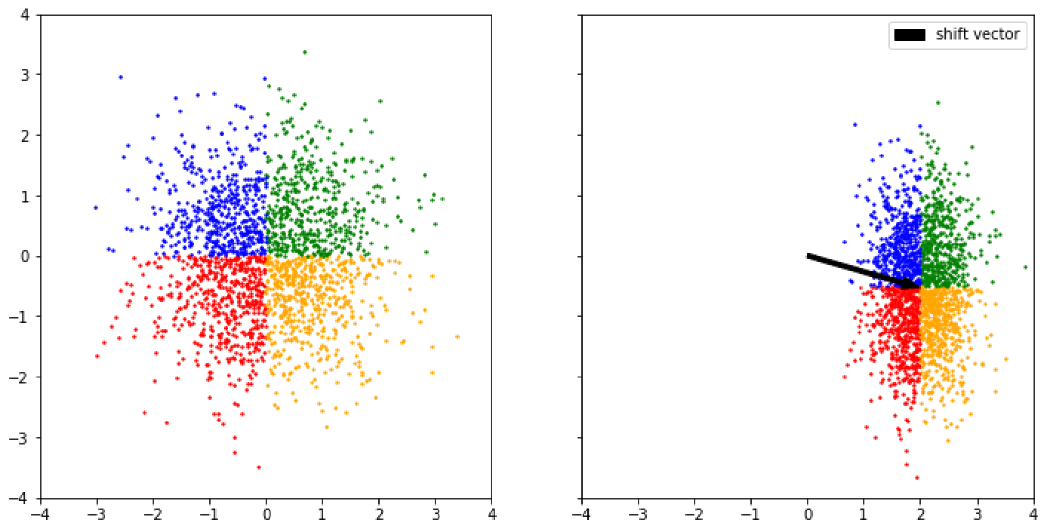

Figure 3.1.: Samples from an affine flow transforming a 2D Gaussian. The distribution is scaled and shifted but stays Gaussian. Color-coding is given to demonstrate how data in each quadrant shifts with the transformation.

3.1.2. Planar Flows

Planar flows (Rezende and Mohamed, 2015) transform the base density to be non-Gaussian and can turn a unimodal distributions into a multimodal ones. These flows expand or contract the base density around a hyperplane. The planar flow is given by

$$\boldsymbol{w},\boldsymbol{u} \in \mathbb{R}^d, b \in \mathbb{R} \quad f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u}\tanh(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{z} + b) \tag{3.11}$$

The hyperbolic tangent can be replaced with any differentiable element-wise non-linearity. We have found tanh to work well. The following equations will be specific to its use. The hyperplane, around which the density is transformed the most, is characterized by the equation $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{z} + b = 0$. $\boldsymbol{u}$ specifies the direction of the expansion/contraction. In figure 3.3 we can see a 2D standard Gaussian being transformed through a planar flow, resulting in the distribution becoming bimodal. Figure 3.2 illustrates how chaining two planar flow results in a more expressive parametric density. The ILDJ for a planar flow is computable in $O(d)$ as

$$\log\left|\det\frac{\partial f}{\partial \boldsymbol{z}}\right|^{-1} = -\log(|I + \boldsymbol{u}^{\mathrm{T}}\tanh'(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{z} + b)\mathbf{w}|) \tag{3.12}$$

Function 3.11 may not be invertible depending on the parameters. To ensure invertibility, the the inequality

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{u} \geq -1 \tag{3.13}$$

needs to hold. Intuitively, if this constraint is not fulfilled, the distribution is compressed towards the hyperplane to a degree where the two sides overlap and make an inversion impossible. This can be avoided by constraining $\boldsymbol{u}$:

$$\hat{u}(\boldsymbol{w},\boldsymbol{u}) = \boldsymbol{u} + \left(-1 + \zeta(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{u}) - (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{u})\right)\frac{\boldsymbol{w}}{||\boldsymbol{w}||^2} \tag{3.14}$$

where ζ is the scalar softplus function $\log(1+e^x)$. This works by adjusting $\boldsymbol{u}$ using its orthogonal projection onto $\boldsymbol{w}$, i.e. $\boldsymbol{u}_{||} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{w}\frac{\boldsymbol{w}}{||\boldsymbol{w}||^2}$, as to fulfill equation 3.13. In practice equation 3.14 suffers numerical problems, with exploding gradients resulting from computing $\frac{\boldsymbol{w}}{||\boldsymbol{w}||^2}$ for small $\boldsymbol{w}$. Proper initialization (i.e. $\boldsymbol{w} = 1$) mitigates some of those problems.

Improving on the Original Parametrization of Planar Flows

In our early experiments, when comparing the origin planar flow parametrization against radial flows, the radial flows converged more stably and yield better results. Therefore most of our further experiments focused on radial flows. Here we propose two approaches that could result in a more stable convergence and therefore better performance when using planar flows:

1-D Optimization: If the planar flow is being used for transforming one-dimensional densities, a simplification can be applied. The symmetry of the hyperbolic tangent, $tanh(-x) = -tanh(x)$ can be used to constrain $w \in \mathbb{R}$ to $w \in \mathbb{R}^+$ without loss of expressivity. This simplifies equation 3.14 to

$$u \in \mathbb{R}, w \in \mathbb{R}^+ \quad \hat{u}(w,u) = \frac{1}{w}(-1 + \zeta(wu)) \tag{3.15}$$

and thereby removes the quadratic denominator $||w||^2$.

Weight Normalization: The right hand side of equation 3.11 can be interpreted as a one layer neural network. The concept of weight normalization (Salimans and Kingma, 2016) has been recently suggested in the context of training neural networks as a way of improving the conditioning of the optimization problem. It works by reparametrizing the weight vector $\boldsymbol{w}$ as

$$\boldsymbol{w},\boldsymbol{v} \in \mathbb{R}^d, g \in \mathbb{R}^+ \quad \boldsymbol{w} = g\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} \tag{3.16}$$

This effectively treats the direction $\boldsymbol{v}$ and the length g of the original weight vector as separate trainable variables. It creates a self-stabilizing effect when using Stochastic Gradient Descent (SGD), but also shows empirical effectiveness when using adaptive gradient methods like Adam (Kingma and Ba, 2014). Using weight normalization, equations 3.11 and 3.14 become:

$$\boldsymbol{v},\boldsymbol{u} \in \mathbb{R}^d, b \in \mathbb{R}, g \in \mathbb{R}^+ \quad f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u}\tanh\left(\frac{g}{\|\boldsymbol{v}\|}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{z} + b\right) \tag{3.17}$$

$$\hat{u}(\boldsymbol{v},\boldsymbol{u},g) = \boldsymbol{u} + \left(-1 + \zeta\left(\frac{g}{\|\boldsymbol{v}\|}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{u}\right) - \left(\frac{g}{\|\boldsymbol{v}\|}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{u}\right)\right)\frac{\boldsymbol{v}}{g\|\boldsymbol{v}\|} \tag{3.18}$$

which also removes the quadratic denominator $\|\boldsymbol{w}\|^2$.

While a planar flow is invertible, no closed-form inverse is known, and the inverse mapping can only be approximated. For samples generated through the flow, the likelihood can be computed through caching the inverse without needing to compute it. For externally provided data, this is not possible. This makes the planar flow in it's normal form difficult to use for CDE, but a solution will be discussed in Section 3.2.

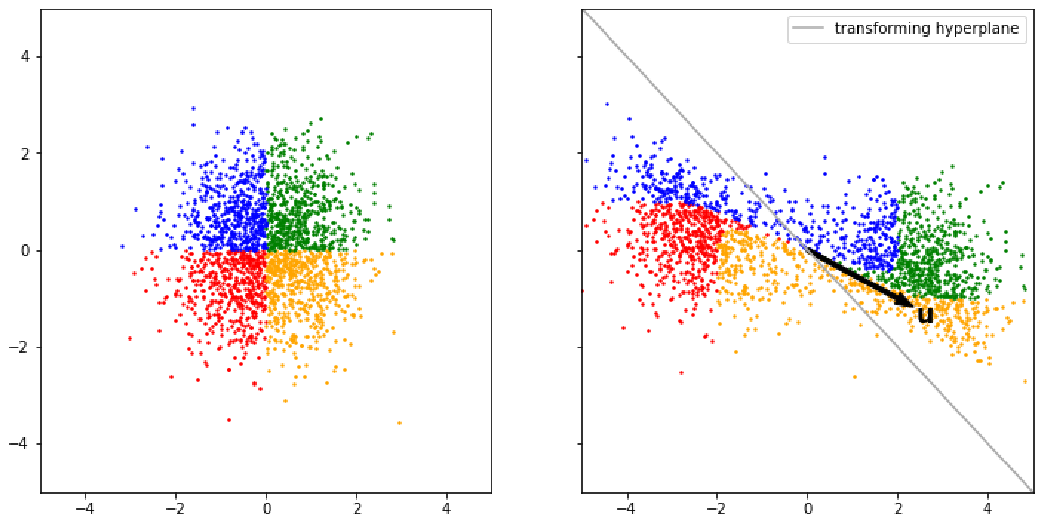

Figure 3.2.: Samples from a planar flow transforming a multivariate Gaussian. The expansion away from the hyperplane makes the resulting density bimodal and non-Gaussian. Color-coding is given to demonstrate how data in each quadrant shifts with the transformation.

3.1.3. Radial Flows

As an alternative to planar flows, we used radial flows (Tabak and Turner (2013), Rezende and Mohamed (2015)). They define an isotropic expansion of the base density around a center point $\boldsymbol{\gamma}$. We use the alternative parametrization defined in Trippe and Turner (2018), since it simplifies the parameter constraints necessary to ensure invertibility of the flow. This flow is given by

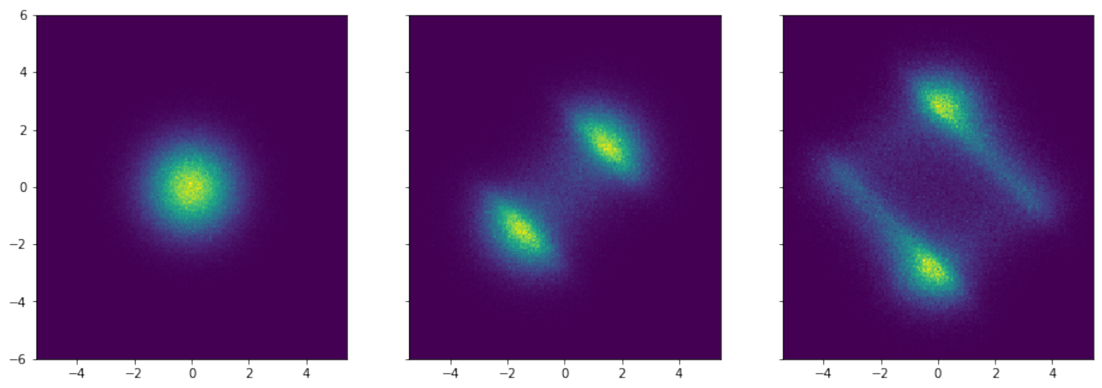
Figure 3.3.: A multivariate Gaussian transformed successively through two planar flows. Chaining multiple flows allow the creation of multimodal complex densities.

$$\boldsymbol{\gamma} \in \mathbb{R}^d, \alpha, \beta \in \mathbb{R} \quad f(\boldsymbol{z}) = \boldsymbol{z} + \frac{\alpha\beta(\boldsymbol{z}-\boldsymbol{\gamma})}{\alpha + \|\boldsymbol{z}-\boldsymbol{\gamma}\|} \tag{3.19}$$

Intuitively, β sets the maximum magnitude of the distortion, whereas α defines how quickly the distortion will decay when moving away from the reference point $\boldsymbol{\gamma}$. The ILDJ is given as:

$$\log\left|\det(\frac{\partial f}{\partial \boldsymbol{z}})\right|^{-1} = -\left[1+\alpha\beta h(\alpha,r)\right]^{d-1}\left[1+\alpha\beta h(\alpha,r)+\alpha\beta h'(\alpha,r)r\right] \tag{3.20}$$

$$h(\alpha,r) = \frac{1}{\alpha+r}, r = |\boldsymbol{z}-\boldsymbol{\gamma}| \tag{3.21}$$

In figure 3.4 we can observe a 2D standard Gaussian being transformed through a radial flow, resulting in the density being contracted around the reference point. Figure 3.5 illustrates how a radial flow can be used to parametrize heavy-tailed distributions.

In order for Function 3.19 to be invertible,

$$\alpha \geq 0, \quad \beta \geq -1 \tag{3.22}$$

must hold. This can be ensured using the transformation:

$$\hat{\alpha} \in \mathbb{R}, \alpha = \zeta(\hat{\alpha}) \quad \hat{\beta} \in \mathbb{R}, \beta = -1+\zeta(\hat{\beta}) \tag{3.23}$$

This is different from the activation function used in Trippe and Turner (2018) where $\beta = -1+\exp(\hat{\beta})$. Although both their activation function and this one achieve the same constraining effect, we have found the softplus function to lead to smoother gradients. Radial flows have a closed form inverse as originally derived in Tabak and Turner (2013). For our parametrization the inverse takes the form:

$$f^{-1}(\boldsymbol{z}') = \frac{r(\boldsymbol{z}'-\boldsymbol{\gamma})}{s} + \boldsymbol{\gamma} \tag{3.24}$$

$$s = \|\boldsymbol{z}'-\boldsymbol{\gamma}\|, \quad r = \frac{s-\alpha(1+\beta)}{2} + \sqrt{\left(\frac{s-\alpha(1+\beta)}{2}\right)^2 + \alpha s} \tag{3.25}$$

Similar to planar flows, radial flows were optimized for fast computation of samples and their associated likelihoods. As a result, the forward function 3.19 is faster to compute than its inverse 3.24.

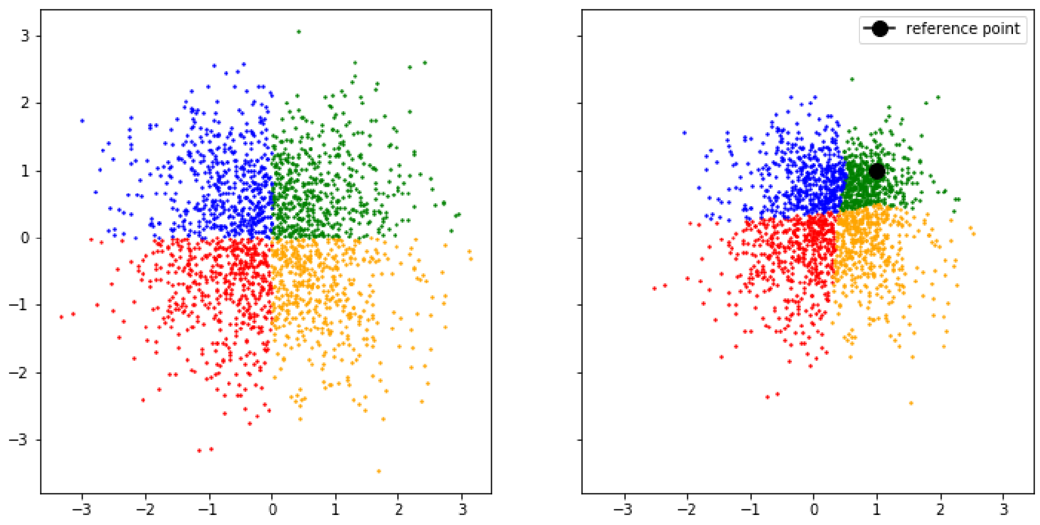

Figure 3.4.: A radial flow transforming a Gaussian. The density is compressed around the reference point.

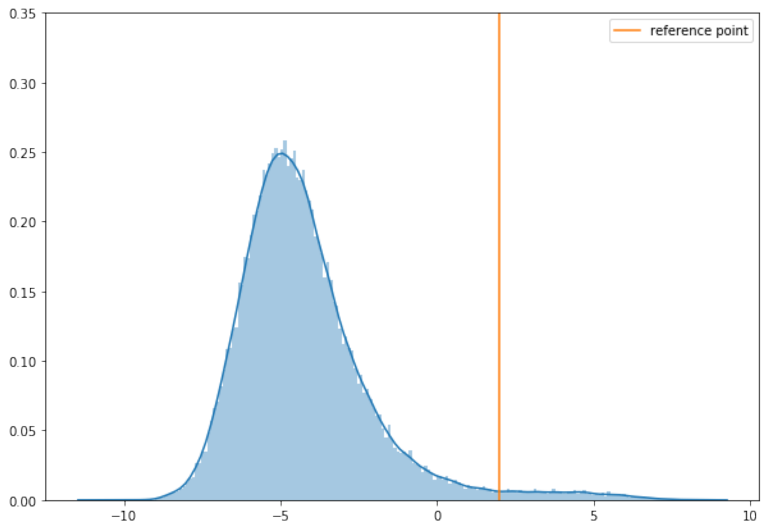

Figure 3.5.: A radial flow transforming a univariate standard Gaussian. The density is expanded around the reference point $\boldsymbol{\gamma}$ creating a heavy-tailed distribution.

3.2. Normalizing Flow Network

NFs are a useful and expressive model for density estimation. We will now generalize them and show how to use NFs in the context of CDE as part of the Normalizing Flow Network (NFN). As noted, the planar and radial flows are optimized for fast drawing of samples and calculation of their likelihood. For CDE, we need the likelihood evaluation for externally provided data to be quick. Therefore we invert the flows, making them define a mapping from the transformed distribution $p(\boldsymbol{z}_n)$ to the base Gaussian distribution $p(\boldsymbol{z}_0)$. This is given by:

$$Z_0 \sim \mathcal{N}(0,I), \quad Z_0 = f_1 \circ \cdots \circ f_N(Z_N) \tag{3.26}$$

$$\log p(\boldsymbol{z}_N) = \log p(\boldsymbol{z}_0) + \sum_{n=1}^{N} \log \left| \det \frac{\partial f_n}{\partial z_n} \right| \tag{3.27}$$

where $f_k(\boldsymbol{z}_k) = \boldsymbol{z}_{k-1}$. Importantly, the sign of the sum is now flipped compared to equation 3.5. It is no longer necessary to compute the inverse to calculate the likelihood of a datapoint. We now consider the parameters $\theta = \{\theta_1, \dots, \theta_N\}$ of an N-stage flow $f_\theta(\cdot) = f_{\theta_1} \circ \cdots \circ f_{\theta_N}$ as tuneable, to be set to the output of a neural network $h_\omega(\boldsymbol{x})$ with trainable weights ω. The structure of this model is illustrated in figure 3.6 The conditional density estimate for the datapoint $(\boldsymbol{x}, \boldsymbol{y})$ is

$$h_\omega(\boldsymbol{x}) = \theta, \quad \log \hat{p}(\boldsymbol{y}|\theta) = \log p(f_\theta(\boldsymbol{y})) + \sum_{n=1}^{N} \log \left| \det \frac{\partial f_{\theta_n}}{\partial z_n} \right| \tag{3.28}$$

The weights of the neural network can then be trained using a negative log-likelihood loss,

$$E_D(\omega) = -\sum_{n=1}^{N} \log \hat{p}(\boldsymbol{y}_n | h_\omega(\boldsymbol{x}_n)) \tag{3.29}$$

which can be optimized using standard backpropagation techniques.

Before training the neural network weights, the hyperparameters of the NFN need be specified. Apart from the configuration of the neural network (activation function, layer sizes, ...) and the inference procedure (learning algorithm, learning rate, ...), the normalizing flow itself also has hyperparameters. The length of the flow can be used to configure the complexity of the resulting distribution. However this is not as intuitive as setting the number of kernels of the MDN or KMN, where the number of kernels provides an upper bound on the number of modes. Furthermore the types of flows can be specified, and mixing different types of flows is possible. In contrast to planar and radial flows, chaining affine flows back-to-back does not result in a more expressive density since shiftings and scalings are additive.

3.3. Related Work: Normalizing Flows

Recently there has been increasing interest in developing density models based on normalizing flows. They provide a few advantages over other successful types of generative models. Unlike generative adversarial networks (Goodfellow et al., 2014) and variational autoencoders (Kingma and Welling (2013), Rezende et al. (2014)), NFs can compute the likelihood of every generated sample. The recent research in the field of normalizing flows has mainly been on how to estimate high dimensional distributions using large datasets, such as images or posteriors in variational inference. As we have observed with the flows introduced previously, often, trade-offs have to be made between potentially orthogonal features like expressiveness and computational complexity. The NF research can be roughly divided into two categories: purely NF based models and models using the autoregressive property (Weng, 2018).

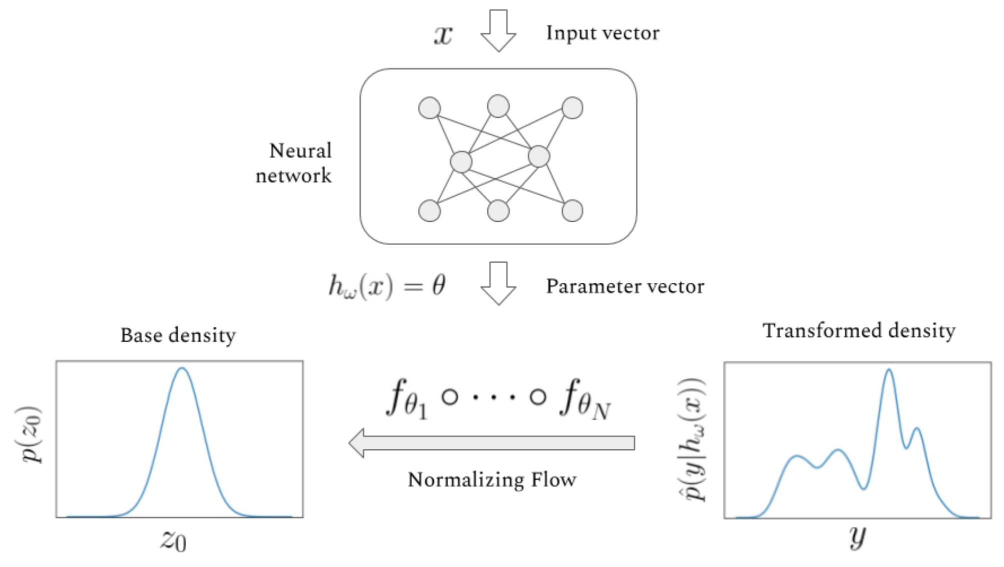

Figure 3.6.: Outline of a Normalizing Flow Network. A neural network $h_\omega(\boldsymbol{x})$ outputs the parameters $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$ necessary to parametrize the N-stage normalizing flow. The standard Gaussian base distribution is transformed into a complex distribution parametrized by $\boldsymbol{\theta}$.

3.3.1. Pure Normalizing Flows

The following models work similarly to the previously introduced normalizing flows by transforming a base distribution through invertible functions into a more complex distribution and employing the change of variable formula.

Sylvester Normalizing Flows

Sylvester Normalizing Flows (van den Berg et al., 2018) are a generalization of the planar flows introduced above. Computing $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ in equation 3.11 is a bottleneck since it reduces the information in $\boldsymbol{x}$ down to a single dimension. This can be interpreted as a single-node neural network. As a remedy, Silvester NFs replace this scalar product by a matrix product, given by

$$\mathbf{A} \in \mathbb{R}^{D\times M}, \mathbf{B} \in \mathbb{R}^{M\times D}, \boldsymbol{b} \in \mathbb{R}^{M}, M \leq D \quad f(\boldsymbol{z}) = \boldsymbol{z} + \mathbf{A}h(\mathbf{B}\boldsymbol{z} + \boldsymbol{b}) \tag{3.30}$$

The right side of this equation is now a single layer MLP with M hidden units instead of just one. The parameters of this flow need to be appropriately constrained to ensure invertibility. Since Sylvester NFs promise expressiveness even when used on low dimensional density, they would be a promising addition to the NFs employed in the NFN.

Real NVP

The real-valued non-volume preserving flows (Real NVP) developed by Dinh et al. (2016) are a generalization of the earlier model called non-linear independent components estimation (NICE) (Dinh et al., 2014), extended by using non-volume preserving transformations. It uses a structure similar to a feistel network (in Dinh et al. (2016) referred to as a coupling layer) to build an invertible, expressive transformation. This structure is outline in figure 3.7. The flow is given by:

$$f(\boldsymbol{x}) = \begin{cases} \boldsymbol{y}_{1:d} & = \boldsymbol{x}_{1:d} \\ \boldsymbol{y}_{d+1:D} & = \boldsymbol{x}_{d+1:D} \odot \exp(s(\boldsymbol{x}_{1:d})) + t(\boldsymbol{x}_{1:d}) \end{cases} \tag{3.31}$$

where s,t are referred to a scale and transformation functions. They can be arbitrary, trainable, real-valued functions. In the paper they are implemented as deep neural networks. In every stage of the model, $\boldsymbol{x}$ is split into two parts x_1,x_2 using a binary mask. x_2 is then transformed by an affine flow parametrized by a scale and transformation function that take x_1 as input. To calculate f^{-1} it suffices to calculate the inverse of the affine flow using 3.7 without needing to invert s or t. This enables both sampling, latent representation and likelihood calculation to happen in linear time. To enable every dimension of $\boldsymbol{x}$ to be changed, the flow is repeated using different binary masks.

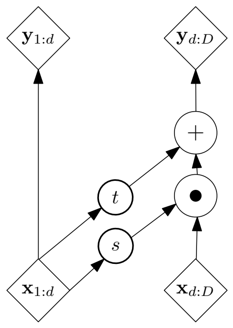

Figure 3.7.: The structure of a single affine coupling layer in Real NVP

GLOW

GLOW (Kingma and Dhariwal, 2018), a generative flow with invertible 1x1 convolutions, expands upon Real NVP. It employs activation normalization instead of batch normalization which works better for the small batch sizes required when training on big images. This enables them to train on and sample high resolution images. The masking employed in Real NVP is changed into a trainable function by using 1x1 convolutions, while keeping the affine flow coupling layer the same.

3.3.2. Autoregressive Normalizing Flows

A multivariate probability density can be deconstructed using the chain rule of probability,

$$p(X_N,\ldots,X_1) = p(X_{N-1},\ldots,X_1)\cdot p(X_N|X_{N-1},\ldots,X_1) = p(X_1)\cdot\prod_{i=2}^{N} p(X_i|X_{i-1}) \tag{3.32}$$

enabling the distribution to be represented as a product of univariate distributions without introducing any assumptions. To perform parametric density estimation, $p(X_i|X_{i-1},\ldots,X_1)$ can then be modelled as a univariate parametric distribution using $f:\mathbb{R}^i \to \Theta, p_\theta(X_i|f(X_{i-1},\ldots,X_1))$. Fixing some ordering on the dimensions of a variable and predicting each dimension utilizing previous dimensions is referred to as the autoregressive property.

MADE

Masked autoencoder for distribution estimation (MADE, Germain et al. (2015)) enables the efficient enforcement of the autoregressive property. By creating suitable masks that zero out connections between the nodes of a neural network, the output nodes can be made to only depend on the inputs that are lower in the ordering. Looking at every output node as a separate function, the output node $o_i(\cdot)$ it takes $x_1,\dots,x_{i-1}$ as an input and uses them to compute the parameters θ_i of the conditional distribution over x_i, $p_\theta(x_i|x_1,\dots,x_{i-1})$. Using masking enables this to be done for all dimensions in a single forward pass of the autoregressive neural network, enabling fast training of the model. The structure of the binary masks are visualized in 3.8. In the original paper this was used to model high-dimensional binary distributions, though the same basic structure can be used to for any parametric univariate distribution. As a drawback of the autoregressive structure, sampling requires N forward passes as the distributions for each dimension can only be computed sequentially. Furthermore if the ordering of dimensions of the input vector is suboptimal, the model will not be able to converge to a good estimate.

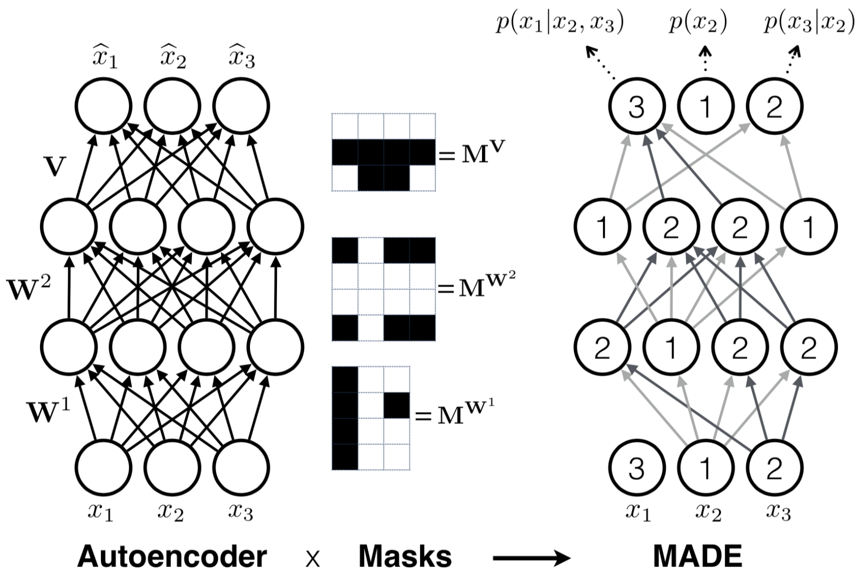
$p(x_1|x_2, x_3)$
$p(x_2)$
$p(x_3|x_2)$
$\widehat{x}_1$ $\widehat{x}_2$ $\widehat{x}_3$
$\mathbf{V}$
$\mathbf{W}^2$
$\mathbf{W}^1$
$= \mathbf{M}^{\mathbf{V}}$
$= \mathbf{M}^{\mathbf{W}^2}$
$= \mathbf{M}^{\mathbf{W}^1}$
x_1 x_2 x_3
Autoencoder
x
Masks
MADE

Figure 3.8.: By applying a binary mask to an autoencoder neural network, the autoregressive property can be enforced. Image source: Germain et al. (2015)

Similar masking techniques have then been used to produce autoregressive models specialized on certain tasks, like PixelCNN (van den Oord et al., 2016b) for images and WaveNet (van den Oord et al., 2016a) for audio.

MAF

Masked autogressive flow for density estimation (MAF, Papamakarios et al. (2017)) shows that autoregressive models like MADE can be interpreted as a normalizing flow. When using a suitable parametric density (e.g. an affine flow transforming a standard univariate Gaussian), MADE transforms a sample from the base distribution using an invertible mapping with a tractable jacobian that is parametrized by an autoregressive function h. The structure of this transformation is visualized in figure 3.9 and given by:

$$y_i = x_i \exp(a_i) + b_i \quad \text{with} \quad a_i = h_{a_i}(x_1,\dots,x_i), b_i = h_{b_i}(x_1,\dots,x_i), x_i \sim \mathcal{N}(0,1) \tag{3.33}$$

$$\log\left|\det \frac{\partial f}{\partial z}\right|^{-1} = -\sum_{i=1}^{D} a_i, \quad a_i = h_{a_i}(x_1,\dots,x_i) \tag{3.34}$$

As one advantage of the autoregressive model, the jacobian is triangular and therefore has a fast-to-compute determinant. As the MAF outputs a normalized density $p(\boldsymbol{y})$ it can be chained to form a more

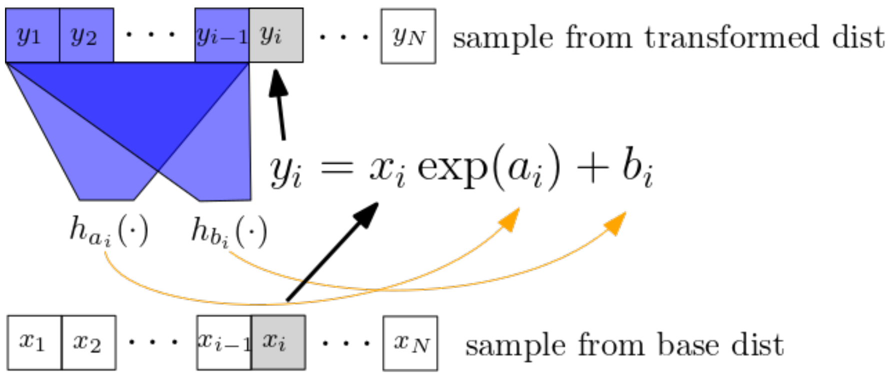

Figure 3.9.: MAF transforms $\boldsymbol{x}$ sampled from the base distribution into $\boldsymbol{y}$ sampled from the transformed distribution. a_i and b_i are computed using the autoregressive functions h_{a_i} and h_{b_i}. While the forward pass $\boldsymbol{y} = f(\boldsymbol{x})$ is sequential, the inverse $\boldsymbol{x} = f^{-1}(\boldsymbol{y})$ can be easily parallelized.

expressive flow, similar to chaining planar/radial flows. By changing the ordering of the dimensions of x for each further stage of MAF, one limitation of MADE can be overcome. Similar to MADE, sampling is expensive. The strongly related model inverse autoregressive flows (IAF, Kingma et al. (2016)), developed for posterior estimation in variational inference, use a very similar structure to MAF except setting different trade-offs by optimizing for fast sampling and in turn dealing with slow likelihood calculation. Estimating conditional densities of the form $p(\boldsymbol{y}|\boldsymbol{x})$ with MAF can be achieved by setting y_i as extra input nodes that come before x_1 in the ordering and are therefore never masked out. Using our normalizing flow network with only affine flows on data where the dependent variable x is one-dimensional is equal to such a conditional MAF. Unlike our NFN, MAF cannot model multimodality for one-dimensional x. In higher dimensions MAF relies on its autoregressivity to model multimodality, whereas the NFN achieves the same by using more expressive parametric densities like radial and planar flows.

NAF

Neural autoregressive flows (NAF, Huang et al. (2018)) are a more expressive version of MAF that uses invertible neural networks instead of affine flows as it's density model. Such invertible neural networks are regular neural networks, restricted to strictly positiv weights and strictly monotonic activation functions. In their paper two examples, deep sigmoidal flows (DSF) and deep dense sigmoidal flows (DDSF) are presented. Calculating the jacobian of the flows with regard to $\boldsymbol{x}$ can be done efficiently via backpropagation. Since deep sigmoidal flows can model multimodality even for univariate densities, they are a potential extension of the planar, radial and affine flows used in our normalizing flow network. It would have to be investigated how useful they are for modelling $p(\boldsymbol{x})$ with $\boldsymbol{x} \in \mathbb{R}^d$ for $d > 1$, as computing the determinant for arbitrary matrices lies in $O(d^3)$. The neural network structure of this flow is visualized in figure 3.10.

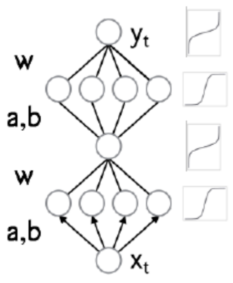

Figure 3.10.: The deep sigmoidal flow used by NAF. $w, a, b \in \mathbb{R}$ are the parameters of the scaled sigmoid implemented by each neuron. Image source: Huang et al. (2018)

3.4. Related Work: CDE with Neural Networks

In the evaluation chapter we will compare the NFN against two other neural network based conditional density estimators: The Mixture Density Network and the Kernel Mixture Network. Both can in theory, given an infinite-dimensional parameter space, model arbitrary conditional densities. Their architecture is similar and visualized in figure 3.11.

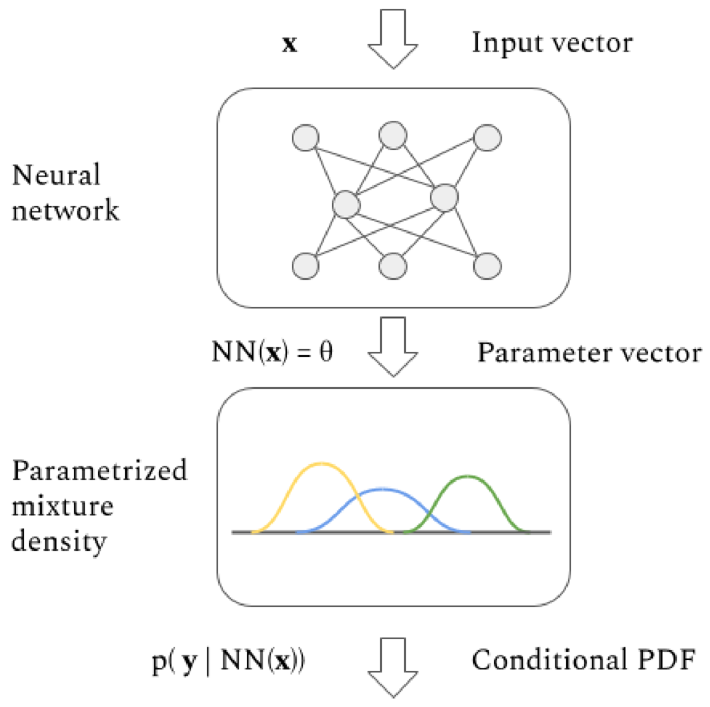

Figure 3.11.: Structural outline of both KMN and MDN

3.4.1. Mixture Density Network

The Mixture Density Network (MDN), introduced by Bishop (1994), combines a neural network with a parametric mixture density. The neural network, given the input vector $\boldsymbol{x}$, outputs a parameter vector $h_\omega(\boldsymbol{x}) = \theta$ that is used to parametrize the mixture distribution $\hat{p}(\boldsymbol{y}|\theta)$, leading to $\hat{p}(\boldsymbol{y}|h_\omega(\boldsymbol{x}))$. Following Bishop (1994), we are be using a Gaussian mixture as parametric family. To keep the parameter space from growing quadratically with the dimension of $\boldsymbol{y}$, we use diagonal covariance matrices. The mixture density network is given by

$$NN_\omega(\boldsymbol{x}) = \theta = \bigcup_{k=1}^{K} \{\mu_k, \sigma_k^2, \lambda_k\} \quad \hat{p}(\boldsymbol{y}|NN_\omega(\boldsymbol{x})) = \sum_{k=1}^{K} \lambda_k \mathcal{N}(\boldsymbol{y}|\mu_k, \sigma_k^2 I) \tag{3.35}$$

where the output of the neural network $h_\omega(\boldsymbol{x}) = \theta$ is split into mean vectors μ_k, variance vectors σ_k^2 and mixture weights λ_k. In order to make a valid probability distribution, the mixture weights λ_k must resemble a categorical distribution. Therefore, it must hold that $\sum_{k=1}^{K} \lambda_k = 1$ and $\forall k \in K : \lambda_k \geq 0$. We can enforce these constraints by using a softmax activation function for the relevant final layer nodes,

$$\lambda_k = \frac{\exp(u_\omega^{\lambda,k}(\boldsymbol{x}))}{\sum_{k=1}^K \exp(u_\omega^{\lambda,k}(\boldsymbol{x}))} \tag{3.36}$$

where $u_\omega^{\lambda,k}(\boldsymbol{x}) \in \mathbb{R}$ refers to the value of the output nodes responsible for the mixture weights λ_k. As a further constraint, the variances σ_k^2 must be positive. We choose the softplus activation function

$$\sigma_k^2 = \log\left(1 + \exp\left(u_\omega^{\sigma^2,k}\right)\right) \tag{3.37}$$

to enforce this. Out of many possible activation functions, this one is preferable, as it is differentiable, positive and partly linear. The number of kernels K to use is one of the hyperparameters of the model. Increasing K results in a more expressive model with an increased danger of overfitting.

3.4.2. Kernel Mixture Network

The Kernel Mixture Network (KMN) can be seen as a combination of parametric and non-parametric methods (Ambrogioni et al., 2017). Similar to the MDN, the KMN is a neural network combined with a mixture density. Here, the network controls only the weights of the mixture. The means of the kernels are calculated based the data and fixed during training. In Ambrogioni et al. (2017), the kernel means are set by subsampling from the dataset and recursively removing points $\boldsymbol{y}_i$ that are closer than a threshold δ to their predecessor. This approach has drawbacks, since it depends on the ordering of the dataset and the amount of kernels is only bounded by the size of the dataset given an inadequate threshold δ. Since this subsampling approach can be seen as a form of data clustering, we instead determine the kernel means using the well-established K-means clustering method. Also, this allows us to directly specify the number of kernels K, we want to use. There are multiple ways to set the kernel scales. One option is to treat them as hyperparameters of the model and to set one or multiple scales per kernel before training the neural network. This enables the use of non-differentiable kernels since the gradient is only calculated with respect to the weights. In our experiments, we find it preferable to instead treat the kernel scales as trainable parameters that are learned independently of the neural network weights. The estimated conditional density calculates as

$$h_\omega(\boldsymbol{x}) = \theta = \bigcup_{k=1}^K \{\lambda_k\} \quad \hat{p}(\boldsymbol{x}|h_\omega(\boldsymbol{x})) = \sum_{m=1}^M \sum_{k=1}^K \lambda_{m,k} \mathcal{N}(y|\mu_k, \sigma_m^2 I) \tag{3.38}$$

with the trainable variances $\left\{\sigma_m^2 | m \in M\right\}$ and fixed means $\{\mu_k | k \in K\}$. As with the MDN, the weights λ_k have to resemble a categorical distribution and are transformed with a softmax activation function before assignment. Given the same number of kernels, the non-trainable mean vectors make the KMN less flexible than a MDN, but also reduce the danger of overfitting.

4. Regularization for Conditional Density Estimation

The conditional density estimators introduced in the previous chapter use flexible distributional families, capable of modelling complex conditional densities. Combined with the expressivity of the neural network, this can lead to severe overfitting when the parameters are trained using the Maximum Likelihood approach. Adequate regularization is necessary. Regularization techniques work by introducing inductive biases (i.e. assumptions about the function other than consistency with the data) in return for lowering the variance of the trained estimator. The popular weight decay method (Hanson and Pratt (1989), Krogh and Hertz (1992)), for example, introduces a bias against weights taking on high values even though this might impede reaching a lower training loss. Regularization techniques for neural networks have been thoroughly investigated (Kukačka et al., 2017) and have been shown to work well in regression and classification settings. For CDE, their effects are more difficult to analyze. This is due to the NN only having an indirect influence on the output of the model as it merely specifies the parameters θ of the output distribution instead of the output itself. For this reason, we develop a noise regularization technique that functions in the data-space. It is therefore independent of the parameters or the specific model being used. Through adding random noise to the data, we lower the variance of the estimator in an easy-to-implement way. We show how this method introduces favourable biases when used in the context of CDE. In this section, we review the most common methods as well as a Bayesian approach to regularization in order to later conduct a benchmark evaluation of our noise regularization technique.

4.1. Common Methods

We start with looking at some of the most common techniques of regularization for neural networks. All mentioned methods will subsequently be evaluated in the experiments section to see how well they apply to CDE. Both weight decay and L_1 / L_2 regularization are typically applied to the weights, but not the biases of a neural network. The biases often have less variance and regularizing them quickly leads to underfitting (Goodfellow et al., 2016). Hence, for this section, the parameter vector ω refers only to the weights of the neural network.

4.1.1. Weight decay

Weight decay as introduced by Hanson and Pratt (1989) refers to scaling down the weight vector at every gradient update step:

$$\omega_{i+1} = -\lambda \nabla E_i + (1-\alpha)\omega_i \tag{4.1}$$

where α specifies the amount of weight decay, λ refers to the learning rate and ∇E_i is the gradient of the loss function at time step i w.r.t. the parameters. It has been shown to improve generalization and to favour local minima of the loss function with smaller weight vectors (Krogh and Hertz, 1992). For standard stochastic gradient descent (SGD, Robbins and Monro (1951)), this is equivalent to adding an L_2 regularization term to the loss function.

$$E^{reg}(\omega_i) = E(\omega_i) + \frac{\alpha'}{2}\|\omega_i\|_2^2 \tag{4.2}$$

where $\alpha' = \frac{\alpha}{\lambda}$. This simplifies the implementation of weight decay as the learning algorithm does not need to be modified. For adaptive gradient methods like Adam (Kingma and Ba, 2014), this equivalence does not hold, as shown by Loshchilov and Hutter (2017). To properly implement this regularization technique for Adam, the weight decay needs to be decoupled from the gradient update step. When implemented properly, weight decay has been shown to lead to improved generalisation even for adaptive gradient methods. In our experiments we use Adam with decoupled weight decay, also referred to as AdamW.

4.1.2. L1 / L2 Regularization

L_1 and L_2 regularization are two closely related methods. They both add an extra term to the loss function that depends solely on the weights ω of the neural network resulting in a regularized loss term:

$$E^{reg}(w_i) = E(w_i) + \Omega(\omega_i) \tag{4.3}$$

For L_1 regularization we have:

$$\Omega(\omega_i) = \|\omega\|_1 \tag{4.4}$$

This is also referred to as LASSO-regularization when used with linear regression models (Tibshirani, 1996). It favours sparse weight vectors that have few non-zero entries (Goodfellow et al., 2016). It has been used as a feature selecting algorithm in linear regression, discarding features whose variables take on a value of zero.

For L_2 the regularization term is:

$$\Omega(\omega_i) = \|\omega\|_2 \tag{4.5}$$

It is also known as ridge regression or Tikonov regularization. As with L_1 regularization, it introduces a biases towards small entries of the weight vector.

4.1.3. Dropout

Dropout (Srivastava et al., 2014) is another frequently used regularization technique. It is only tangentially related to weight decay and L_1 / L_2 regularization and can easily be combined with the previously mentioned methods. Dropout can be interpreted as a computationally inexpensive way of implementing more complex ensemble methods. It works by randomly dropping nodes from the network during training. For every minibatch of the stochastic gradient descent algorithm, a binary mask is sampled that determines which of the input units and hidden units will be kept. This binary random variable is sampled independently for every node. Switching off a node can be implemented as multiplying its output by zero. The distribution of the random variable is set as a hyperparameter of the regularization technique in advance and usually does not change during the training process. $p(1) = p(0) = 0.5$ is a common choice. The forward pass, backward pass, and gradient update are then run as usual.

4.2. Bayesian Neural Networks

Next we will look at using Bayesian inference instead of maximum likelihood estimation (MLE) as a means of regularizing our estimation problems. In particular we will replace the neural network with a Bayesian neural network (Neal, 2012), turning the NFN into a probabilistic model. This works by interpreting the weights and biases ω as random variables. This regularization approach works through setting the prior distributions over these random variables in a way that encodes inductive biases e.g. incentivizes smaller weights.

4.2.1. Bayesian Inference

Take a probabilistic model with parameters ω (also called latent variables) and a dataset D (also called visible variables) (Goodfellow et al., 2016). In a probabilistic model, uncertainties about values and parameters are expressed as probability distributions. Such a model is fully specified by the joint distribution $p(\omega, D)$ (Deisenroth et al., 2020). Given the joint distribution, Bayesian inference amounts to computing the posterior probability $p(\omega|D)$:

$$p(\omega|D) = \frac{p(D|\omega)p(\omega)}{p(D)} \tag{4.6}$$

where

- $p(\omega)$ is referred to as the prior distribution. It describes the probability of the parameters before any data is observed. The prior can be used to encode previous assumptions about the parameters.

- $p(D|\omega)$ is the likelihood. It is the probability that the observed data was generated by the model given the parameters ω.

- $p(D)$ is the marginal likelihood or evidence. It is computed by marginalizing over the parameters: $p(D) = \int p(D|\omega)p(\omega)d\omega$. It serves as normalization for the posterior.

Here, the result of our inference procedure is a probability distribution over weights $p(\omega|D)$. Both MLE and Maximum a posteriori inference (MAP) can be interpreted as special cases of Bayesian inference. MLE corresponds to Bayesian inference with a uniform prior where the result is a mode of the posterior distribution ω^*. As the evidence $p(D)$ is not dependent on ω and therefore does not change the position of the mode, we can forgo its calculation in MLE.

In MAP the result is also just the mode of the posterior. Yet contrary to MLE an informative, non-uniform prior is used. The previously mentioned L_1 and L_2 regularization techniques can be interpreted as MAP inference with Laplace and Gaussian priors respectively (Figueiredo, 2003). The BNN is visualized as a graphical model with its latent and observed variables in figure 4.1.

4.2.2. Variational Inference

The problem with inference in deep, expressive probabilistic models is that the posterior becomes difficult to compute. Marginalizing over the parameter space to compute the evidence $\int p(D|\omega)p(\omega)d\omega$ is unfeasible for deep neural networks with thousands of parameters. Variational inference (VI, Jordan et al. (1999), Blei et al. (2017)) is one solution to this problem. It describes a way of approximating difficult-to-compute probability densities through casting it as an optimization problem. In contrast to the popular alternative method Markov Chain Monte Carlo (MCMC), VI scales better with the dimensionality of the parameter space Ω and size of the training data set.

Let $\mathscr{Q}$ be a family of densities, where a member $q(\omega) \in \mathscr{Q}$ is specified using a parameter vector $\theta \in \Theta$. The goal of VI is to pick the one member $q^*_\theta(\omega) \in \mathscr{Q}$ that is closest to the true posterior $p(\omega|D)$, according to some distance measure. Often the KL divergence is used, an asymmetric, non-negative measure of proximity. This leads us to Kullback Leibler variational inference, which reads as follows:

$$q^*_\theta(\omega) = \arg \min_{q_\theta(\omega)\in\mathscr{Q}} D_{\text{KL}}(q_\theta(\omega)\|p(\omega|D)) \tag{4.7}$$

The family $\mathscr{Q}$ should be expressive enough to enable one to find a close approximation of the true posterior. It should also be efficiently computable to enable fast optimization. Picking the approximate posterior from a parametric family can be considered one of the drawbacks of VI. If the actual posterior is not a member of the parametric family, the algorithm will never converge to the true solution even given infinite datapoints.

By using VI, we have replaced the integration problem present in equation 4.6 with an optimization problem that is more easily tractable.

4.2.3. The Evidence Lower Bound

To solve (4.7), we attempt to find the minimum of the following functional with regard to $q_\theta(\omega)$:

$$D_{\text{KL}}(q_\theta(\omega)\|p(\omega|D)) = \mathbb{E}_{q_\theta(\omega)}[\log q_\theta(\omega)] - \mathbb{E}_{q_\theta(\omega)}[\log p(\omega|D)] \tag{4.8}$$

$$= \mathbb{E}_{q_\theta(\omega)}[\log q_\theta(\omega)] - \mathbb{E}_{q_\theta(\omega)}[\log p(\omega,D)] + \log p(D) \tag{4.9}$$

This still requires calculating the log evidence $p(D)$ and therefore cannot be optimized directly. Instead we resort to optimizing a different objective, the evidence lower bound (ELBO):

$$\text{ELBO}(q(\omega)) = -\mathbb{E}_{q_\theta(\omega)}[\log q_\theta(\omega)] + \mathbb{E}_{q_\theta(\omega)}[\log p(\omega,D)] \tag{4.10}$$

$$= -\mathbb{E}_{q_\theta(\omega)}[\log q_\theta(\omega)] + \mathbb{E}_{q_\theta(\omega)}[\log p(\omega|D)] + \log p(D) \tag{4.11}$$

$$= -D_{\text{KL}}(q_\theta(\omega)\|p(\omega|D)) + \log p(D) \tag{4.12}$$

The ELBO is similar to the negative KL divergence plus the log evidence. Since the evidence $p(D)$ does not depend on $q(\omega)$, maximizing the ELBO yields the same approximate posterior as minimizing the KL divergence. The ELBO gets its name from the inequality

$$\text{ELBO}(q(\omega)) \leq \log p(D) \tag{4.13}$$

Which follows from (4.12) and $D_{KL}(\cdot) \geq 0$. The ELBO therefore lower-bounds the log evidence. It is sometimes referred to as the negative free energy $\mathscr{F}$. To get some insights into the ELBO objective, we rewrite equation 4.10 as

$$\text{ELBO}(q_\theta(\omega)) = \mathbb{E}_{q_\theta(\omega)}[\log p(D|\omega)] + \mathbb{E}_{q_\theta(\omega)}[\log p(\omega)] - \mathbb{E}_{q_\theta(\omega)}[\log q_\theta(\omega)] \tag{4.14}$$

$$= \mathbb{E}_{q_\theta(\omega)}[\log p(D|\omega)] - D_{\text{KL}}(q_\theta(\omega)\|p(\omega)) \tag{4.15}$$

Therefore, in maximizing the ELBO we try to find an approximate posterior that results in a high expected likelihood while being close to the prior (Blei et al., 2017).

4.2.4. Gradient Estimators for the ELBO

Next, we look at the two terms of (4.15). When using suitable distributions for the priors, as well as for the approximate posterior, the KL-divergence in this term is analytically tractable. This is true, for example, for conjugate distributions like a Gaussian prior and a Mean-field posterior, introduced in the next section.

Calculating the expected likelihood, i.e. the first term in (4.15), is not as simple. It can only be approximated using Monte Carlo methods. To optimize the ELBO we need to calculate the gradient of this expected likelihood w.r.t. θ. One approach is a score function gradient estimator:

$$\nabla_\theta \mathbb{E}_{q_\theta(\omega)}[\log p(D|\omega)] = \mathbb{E}_{q_\theta(\omega)}[\log p(D|\omega)\nabla_\theta \log q_\theta(\omega)] \tag{4.16}$$

$$= \frac{1}{M}\sum_{i=0}^{M} \log p(D|\omega_i)\nabla_\theta \log q_\theta(\omega_i) \tag{4.17}$$

where $\omega_i \sim q_\theta(\omega)$. Equation 4.16 can be obtained using the log derivative trick. For further information on this derivation we refer to Mohamed et al. (2019), section 4.2. In practice this gradient estimation has a high variance, which makes it difficult to use. One solution is to utilize a pathwise gradient estimator by employing the reparametrization trick (Kingma and Welling, 2013). It works by replacing the random variable ω with a deterministic function $g(\varepsilon,\theta)$ and an auxiliary random variable ε that has a simpler, non-parametric distribution $p(\varepsilon)$. It needs to hold that:

$$\omega \sim q_\theta(\omega) \quad \equiv \quad \omega = g(\varepsilon,\theta), \varepsilon \sim p(\varepsilon) \tag{4.18}$$

As an example, consider an arbitrary Gaussian:

$$\omega \sim \mathcal{N}(\mu, \sigma^2) \quad \equiv \quad \omega_i \sim g(\varepsilon, \mu, \sigma) = \sigma \cdot \varepsilon + \mu, \quad \varepsilon \sim \mathcal{N}(0,1) \tag{4.19}$$

Another examples would be using an affine flow to parametrize log-scale distributions as explained in section 3.1.1. The reparametrization trick allows us to formulate the gradient of the expectation of the likelihood as:

$$\nabla_\theta \mathbb{E}_{p_\theta(\omega)}[\log p(D|\omega)] = \nabla_\theta \mathbb{E}_{p(\varepsilon)}[\log p(D|g(\varepsilon, \theta))] = \mathbb{E}_{p(\varepsilon)}[\nabla_\theta \log p(D|g(\varepsilon, \theta))] \tag{4.20}$$

$$= \frac{1}{M} \sum_{i=0}^{M} \nabla_\theta \log p(D|g(\varepsilon_i, \theta)) \tag{4.21}$$

where $\varepsilon_i \sim p(\varepsilon)$. This usually results in estimates of much lower variance (Kingma and Welling, 2013).

Instead of calculating the gradient $\nabla_\theta \log p(D|g(\varepsilon_i, \theta))$ over the whole dataset D, we use minibatches. Minimizing the negative ELBO using SGD with minibatches has been named stochastic variational inference (Hoffman et al., 2012). Adding to this the stochasticity of approximating the gradient using MC is called doubly stochastic variational inference (Titsias and Lázaro-Gredilla, 2014). It allows us to perform variational inference on complex models and large datasets.

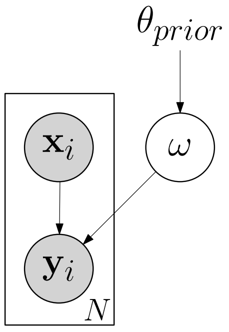

Figure 4.1.: A Bayesian neural network represented as a graphical model. $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are the observed variables. In using a BNN we assume that the conditional random variable $\boldsymbol{y}_i$ is dependent only on the input variable $\boldsymbol{x}_i$ and on the weights and biases ω of our neural network. The parameter θ_{prior} governing our prior $p_{\theta_{prior}}(\omega)$ is another latent variable. As can be seen, we have no per-datapoint latent variables, therefore our posterior distribution will be fixed and independent of the input.

4.2.5. The Mean-field Distributional Family

To finalize the VI procedure, we are now left with picking a parametric distribution for the approximate posterior distribution. One of the simplest options is the mean-field distribution family, leading to mean-field variational inference.

$$q(\boldsymbol{\omega}|\theta) = \prod_{i=0}^{N} q_{\theta_i}(\omega_i) \tag{4.22}$$

This makes the rather bold assumption that the posterior can be fully factorized into independent distributions over each dimension. This assumption makes the posterior efficient to optimize. Often a univariate Gaussian is picked as the factorized distribution, leading to the following parametric posterior:

$$\theta = \left\{(\mu_i, \delta_i^2), \ldots, (\mu_N, \delta_N^2)\right\}, \quad \mu_i \in \mathbb{R}, \delta_i^2 \in \mathbb{R}^+, \quad q_\theta(\boldsymbol{\omega}) = \prod_{i=0}^{N} \mathcal{N}(\omega_i|\mu_i, \delta_i^2) \tag{4.23}$$

A mean field posterior is just one of the options. In recent years, many alternatives have been proposed, like inverse autoregressive flows that have yielded performance gains (Kingma et al., 2016). However, for some of those posterior distributions, calculating the KL divergence between posterior and prior is not possible analytically. It has to be estimated using Monte-Carlo methods, resulting in another source of stochasticity. A more expressive posterior family would allow the convergence to approximations that are closer to the true posterior. A trade-off has to be made between expressiveness and speed of computation.

4.2.6. Bayesian Inference as Regularization

BNNs can be used as a regularization technique by setting appropriate priors over the weights and biases. As can be seen in equation 4.15, maximizing the ELBO is a trading-off between maximizing the likelihood while keeping the approximate posterior close to the prior. In our experiments, we set the prior distribution to be a standard Gaussian, thereby biasing the parameters of the NN to be close to zero. More advanced ways of setting the prior (e.g. setting a less informative prior for the biases of the NN since they tend to overfit less (Goodfellow et al., 2016)) could be explored. Hyperparameter-tuning the priors is also be possible. In total, BNNs and VI are complex to implement compared to the previously introduced regularization approaches.

4.3. Noise regularization

All previously introduced regularization methods work in a parameter focused way, either having a direct effect on how the parameters are updated (Dropout, weight decay, BNN) or adding a parameter dependent loss (L1/L2). The biases introduced by those methods are harder to analyze in the context of CDE compared to standard regression. This is due to the neural network merely having an indirect effect on the output of the model, as it specifies just the parameters of the output distribution and not the output itself. Furthermore, all four previously introduced methods require separate implementations for each CDE model.

In this section we introduce a novel approach, noise regularization for CDE. Noise regularization refers to adding random noise the data during each minibatch. We show how this methods introduces favourable biases and leads to a smoothing of conditional distribution while being easy to implement and model agnostic. For a more in-depth analysis and consistency results, we refer to the associated paper Rothfuss et al. (2019).

Adding noise to the datapoints during training has been a commonly used regularization technique for regression and classification (Holmstrom and Koistinen (1992), Bishop (1995)). In contrast, we analyze such noise regularization in the more general case of conditional density estimation. Adding noise can be understood as a form of dataset augmentation. The original data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ are replaced by random variables $(\boldsymbol{x}_i + \boldsymbol{\xi}_x, \boldsymbol{y}_i + \boldsymbol{\xi}_y)$, where the noise is sampled from the distributions $K_x(\boldsymbol{\xi}_x)$ and $K_y(\boldsymbol{\xi}_y)$. Since we can sample infinitely many noise vectors, we can increase the size of the dataset arbitrarily. We require the noise distribution to be zero-centered and independently as well as identically distributed in each dimension with a standard deviation of h, that is,

$$\mathbb{E}_{\boldsymbol{\xi} \sim K(\boldsymbol{\xi})}[\boldsymbol{\xi}] = 0, \quad \mathbb{E}_{\boldsymbol{\xi} \sim K(\boldsymbol{\xi})}\left[\boldsymbol{\xi}\boldsymbol{\xi}^\top\right] = h^2 I \tag{4.24}$$

The strength of the regularization can be controlled through the variances h_x^2, h_y^2, with a higher variance leading to more smoothing. When using stochastic gradient descent with mini-batches, we apply this augmentation to every mini-batch of datapoints, as outlined in algorithm 1.

Algorithm 1 Conditional MLE with Noise Regularization using Mini-Batch Gradient Descent

```
Require: $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), \dots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, standard deviation of the noise $h$
Require: learning rate $\alpha$, mini-batch size $m$
  Initialize $\theta$
  while $\theta$ not converged do
    Sample minibatch $\{(\boldsymbol{x}_i, \boldsymbol{y}_i), \dots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\} \subset \mathscr{D}$
    for j = 1 to m do
      Draw perturbation $\boldsymbol{\xi} \sim K$
      Set $\tilde{\boldsymbol{x}}_j = \boldsymbol{x}_j + h\boldsymbol{\xi}$
      Set $\tilde{\boldsymbol{y}}_j = \boldsymbol{y}_j + h\boldsymbol{\xi}$
    end for
    $\theta \leftarrow \theta + \alpha \nabla_\theta \sum_{j=1}^{m} \log \hat{p}(\hat{\boldsymbol{y}}_j | h_\omega(\hat{\boldsymbol{x}}_j))$
  end while
  return optimized parameter $\theta$
```

By adding these small random perturbations, the data points are smeared out, making it harder for the model to overfit to individual data points. Intuitively, this leads to a conditional density estimate that is smoother w.r.t. $\boldsymbol{x}$ as well as $\boldsymbol{y}$.

This intuition can be supported mathematically. We employ a second order Taylor expansion of the negative log likelihood loss $l(\boldsymbol{z}_i + \boldsymbol{\xi})$ around each datapoint $\boldsymbol{z}_i \in D$:

$$l(\boldsymbol{z}_i + \boldsymbol{\xi}) \approx l(\boldsymbol{z}_i) + \boldsymbol{\xi}^\top \nabla_{\boldsymbol{z}} l(\boldsymbol{z})\big|_{\boldsymbol{z}_i} + \frac{1}{2}\boldsymbol{\xi}^\top \nabla^2_{\boldsymbol{z}} l(\boldsymbol{z})\big|_{\boldsymbol{z}_i}\boldsymbol{\xi} + \mathscr{O}(\boldsymbol{\xi}^3) \tag{4.25}$$

Assuming that the noise $\boldsymbol{\xi}$ is small, we can neglect $\mathscr{O}(\boldsymbol{\xi}^3)$. Now we take the expectation of this loss and use the constraints for $\boldsymbol{\xi}$ defined in (4.24) to simplify:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})}\left[l(\boldsymbol{z}_i + \boldsymbol{\xi})\right] &\approx \mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})} l(\boldsymbol{z}_i) + \mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})}\left[\boldsymbol{\xi}^\top \nabla_x l(\boldsymbol{z})\big|_{\boldsymbol{z}_i}\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})}\left[\boldsymbol{\xi}^\top \nabla^2_x l(\boldsymbol{z})\big|_{\boldsymbol{z}_i}\boldsymbol{\xi}\right] && (4.26)\\
&= l(\boldsymbol{z}_i) + \mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})}\left[\boldsymbol{\xi}\right]^\top \nabla_{\boldsymbol{z}} l(\boldsymbol{z})\big|_{\boldsymbol{z}_i} + \frac{1}{2}\mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})}\left[\boldsymbol{\xi}^\top \nabla^2_{\boldsymbol{z}} l(\boldsymbol{z})\big|_{\boldsymbol{z}_i}\boldsymbol{\xi}\right] && (4.27)\\
&= l(\boldsymbol{z}_i) + \frac{1}{2}\mathbb{E}_{\boldsymbol{\xi}\sim K(\boldsymbol{\xi})}\left[\sum_j\sum_k \boldsymbol{\xi}_j\boldsymbol{\xi}_k \frac{\partial^2 l(\boldsymbol{z})}{\partial z^{(j)}\partial z^{(k)}}\bigg|_{\boldsymbol{z}_i}\right] && (4.28)\\
&= l(\boldsymbol{z}_i) + \frac{1}{2}\sum_j \mathbb{E}_{\boldsymbol{\xi}}\left[\boldsymbol{\xi}_j^2\right] \frac{\partial^2 l(\boldsymbol{z})}{\partial z^{(j)}\partial z^{(j)}}\bigg|_{\boldsymbol{z}_i} + \frac{1}{2}\sum_j\sum_{k\neq j}\mathbb{E}_{\boldsymbol{\xi}}\left[\boldsymbol{\xi}_j\boldsymbol{\xi}_k\right] \frac{\partial^2 l(\boldsymbol{z})}{\partial z^{(j)}\partial z^{(k)}}\bigg|_{\boldsymbol{z}_i} && (4.29)\\
&= l(\boldsymbol{z}_i) + \frac{\eta^2}{2}\sum_j \frac{\partial^2 l(\boldsymbol{z})}{\partial z^{(j)}\partial z^{(j)}}\bigg|_{\boldsymbol{z}_i} && (4.30)
\end{aligned}$$

Where $z^{(j)}$ denotes the elements of the column vector $\boldsymbol{z}$. The negative log likelihood loss on a dataset is the sum over the losses for each datapoint $\boldsymbol{z}_i \in D$. We transform (4.30) by moving to the negative log likelihood loss over the dataset and expanding $\boldsymbol{z}_i = (\boldsymbol{x}_i, \boldsymbol{y}_i)$:

$$E_\omega(D) \approx -\sum_{i=1}^{n} \log \hat{p}(\boldsymbol{y}_i | h_\omega(\boldsymbol{x}_i)) - \frac{h_y^2}{2}\sum_{i=1}^{n}\sum_{j=1}^{d_y} \frac{\partial^2 \log \hat{p}(\boldsymbol{y}_i | h_\omega(\boldsymbol{x}_i))}{\partial y^{(j)}\partial y^{(j)}}\bigg|_{\substack{x=x_i\\y=y_i}} - \frac{h_x^2}{2}\sum_{i=1}^{n}\sum_{j=1}^{d_x} \frac{\partial^2 \log \hat{p}(\boldsymbol{y}_i | h_\omega(\boldsymbol{x}_i))}{\partial x^{(j)}\partial x^{(j)}}\bigg|_{\substack{x=x_i\\y=y_i}} \tag{4.31}$$

The first term of this equation is the regular negative log likelihood without any regularization. The second and third term are a regularization penalty on the second derivative of the loss function with regards to $\boldsymbol{x}$ and $\boldsymbol{y}$, proportional to the respective noise variances. The first term pushes the parameters towards high likelihood of the data, while the other terms incentivize smoothness of the density estimate w.r.t. $\boldsymbol{x}$ and $\boldsymbol{y}$.

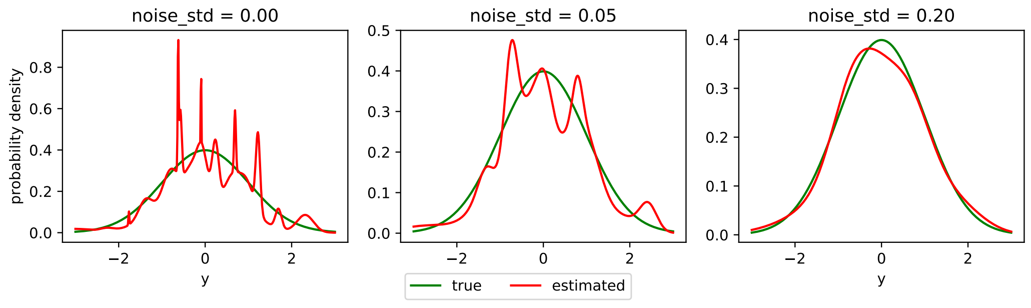

Figure 4.2.: A conditional MDN density estimate (red) and the true conditional density (green) for different intensities of noise regularisation h. We fit the MDN using 3000 samples drawn from a conditional Gaussian. No noise regularization results in the model overfitting to individual samples. By increasing the noise intensity, the model captures the underlying patterns, allowing it to generalize better.

The effects of this can be observed in figure 4.3, where we fit a MDN to 3000 datapoints. Without noise regularization, the conditional is jagged and poorly captures the underlying patterns in the data. Increasing the amount of noise regularization through raising the standard deviation h results in a smoothing of the conditional, leading to visibly better generalization beyond the training data.

To summarize, in this section we introduced noise regularization and mathematically showed how it adds a favourable smoothness bias when used in CDE. Compared to the previously introduced regularization techniques that work in the parameter space, noise regularization makes no further assumptions about the CDE model being used. It only requires manipulating the data in each minibatch through adding random noise. Noise regularization is therefore among the easiest regularization techniques to implement, especially compared to complex methods like weight decay and BNNs. In the next section we show how this methods is effective across models and datasets, making it the preferable approach for regularizing high-capacity CDE models when data is scarce.

5. Experiments and Comparisons

In this chapter we evaluate our CDE models as well as the presented regularization techniques across 3 simulated densities and 5 real-world datasets. We compare the NFN's performance against that of the MDN and KMN. We compare the effectiveness of noise regularization against L1 & L2 regularization, weight decay and the BNN. For both the simulated densities and the real world datasets, this chapter first introduces the data before then presenting and discussing the results.

5.1. Evaluation on Simulated Densities

5.1.1. Gaussian Mixture Model

We use the GMM to test how well each model can fit a multimodal conditional. In this simulated density, the joint distribution $p(\boldsymbol{x},\boldsymbol{y})$ is given by a mixture of K Gaussians where $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^l$ can be factorized. The conditional distribution can be expressed as:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{i=1}^{K} W_k(x)\mathcal{N}(\boldsymbol{y}|\mu_{y,k},\Sigma_{y,k}) \tag{5.1}$$

The mixture weights are a categorical distribution that is a function of $\boldsymbol{x}$:

$$W_k(\boldsymbol{x}) = \frac{w_k\ \mathcal{N}(\boldsymbol{x}|\mu_{x,k},\Sigma_{x,k})}{\sum_{j=1}^{K} w_k\ \mathcal{N}(\boldsymbol{x}|\mu_{x,j},\Sigma_{x,j})} \tag{5.2}$$

The number of kernels and the dimensionality of $\boldsymbol{x},\boldsymbol{y}$ are hyperparameters of this simulation that can be set in advance. A plot of three conditionals of a GMM consisting of 5 kernels can be seen in figure 5.1.1.

The mixture weights, kernel means, and kernel convariance matrices are all sampled at random, allowing us to do cross-validation by using different instances of the same simulation density.

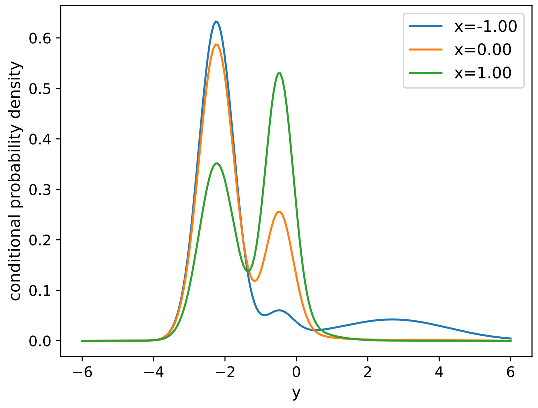

Figure 5.1.: Gaussian mixture with 5 components. $\boldsymbol{x},\boldsymbol{y}$ are both one-dimensional.

5.1.2. Skew-normal

For the Skew-normal simulated density, the data is sampled from a joint distribution $p(x,y)$ in two dimensions. $x \in \mathbb{R}$ is normally distributed and $y \in \mathbb{R}$ follows a conditional univariate Skew-normal distribution (Azzalini, 2005) whose parameters (ξ, ω, α) are dependent on x in the following way:

$$x \sim \mathcal{N}\left(x \,\middle|\, 0, \frac{1}{2}\right) \tag{5.3}$$

$$\xi(x) = ax + b \qquad a, b \in \mathbb{R} \tag{5.4}$$

$$\omega(x) = cx^2 + d \qquad c, d \in \mathbb{R} \tag{5.5}$$

$$\alpha(x) = \alpha_{low} + \frac{1}{1+e^{-x}}(\alpha_{high} - \alpha_{low}) \tag{5.6}$$

$$y \sim p(y|x) = \frac{2}{\omega(x)} \mathcal{N}\left(\frac{y - \xi(x)}{\omega(x)}\right) \Phi\left(\alpha(x) \frac{y - \xi(x)}{\omega(x)}\right) \tag{5.7}$$

Here, $\Phi(x)$ refers the the cumulative distribution function of a normal distribution. y follows a distribution whose skewness changes over x. It allows us to gauge the performance of the three estimators on skewed distributions, which are often observed in financial markets. A plot of three conditionals of a Skew Normal density can be observed in figure 5.1.2.

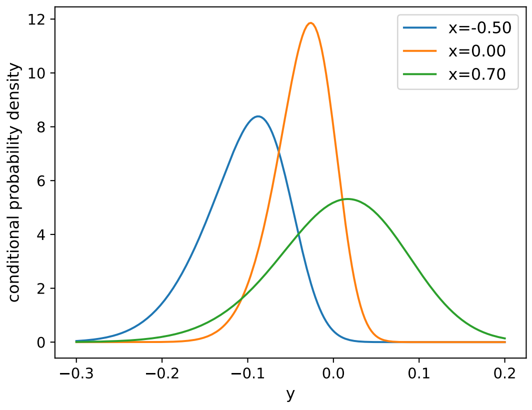

Figure 5.2.: Conditional probability density of the Skew Normal simulated density evaluated a different values of $\boldsymbol{x}$. Both $\boldsymbol{x}$ and $\boldsymbol{y}$ are one-dimensional.

5.1.3. Econ density

Data from this density is sampled as follows:

$$x = |\varepsilon_x|, \quad \varepsilon_x \sim \mathcal{N}(\varepsilon_x|0,1) \tag{5.8}$$

$$y \sim \mathcal{N}(y|\mu = x^2, \sigma = 1 + x) \tag{5.9}$$

The idea behind this density was for x to represent market volatility and for y to be a financial measure that is explained by volatility. For higher market volatility, the noise in the financial measure y increases. Furthermore, the relationship between the two variables is non-linear. This is a fairly simple simulated density with the conditional distribution being Gaussian. Therefore, we can expect our models with Gaussian output distributions (the MDN and KMN) to perform well. A plot of three conditionals of the

Econ density can be observed in figure 5.1.3.

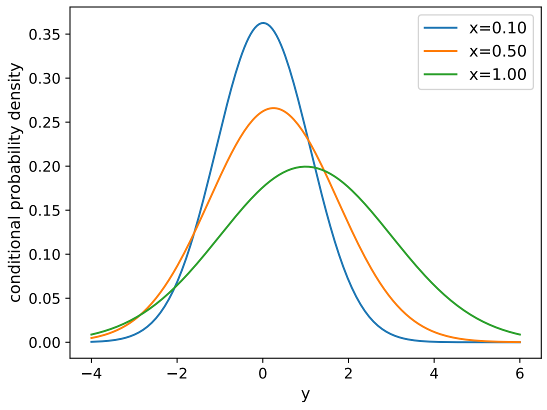

Figure 5.3.: Conditional probability density of the Econ density evaluated a different values of x.

5.1.4. Results

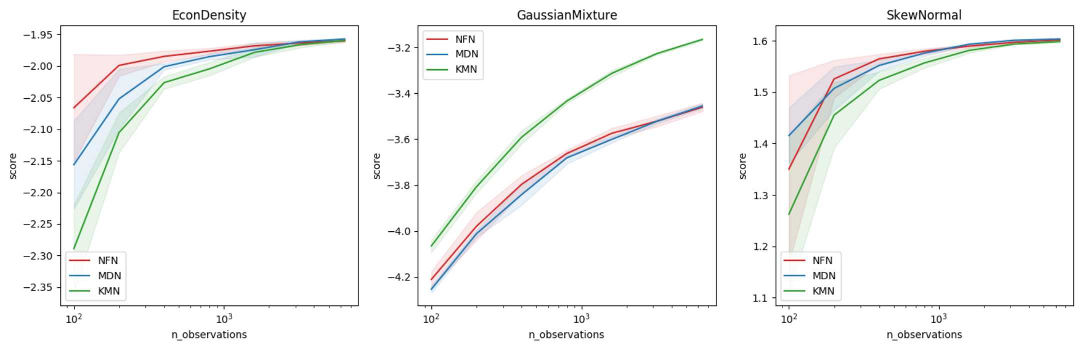

Figure 5.4.: Results of running the Normalizing Flow Network, the Mixture Density Network and the Kernel Mixture Network on the three simulated datasets. All models have noise regularization applied according to the rule-of-thumb noise schedule with an initial bandwidth of 0.5 (Rothfuss et al., 2019).

First we analyze how well the NFN performs compared to the MDN and KMN on the three simulated datasets, as seen in figure 5.4. Noise regularization was used. The variance of the added noise was set to temper off with increasing number of datapoints, according to the rule-of-thumb noise schedule, with an inital noise bandwidth of 0.5. More information on noise schedules can be found in the associated paper Rothfuss et al. (2019). Before computing the results on any of the datasets, we perform hyperparameter tuning on the three simulated densities. The resulting configuration for every estimator can be found in the appendix A.1.

The KMN, with its Gaussian Mixture conditional, where the means are sampled from the data, is ideally suited for the GMM dataset and can be seen outperforming the NFN and MDN. For the two other simulated datasets, the NFN performs best. As observed in the results for the Econ density, the NFN is perfectly capable of modelling variants of the Gaussian distribution through transformations using

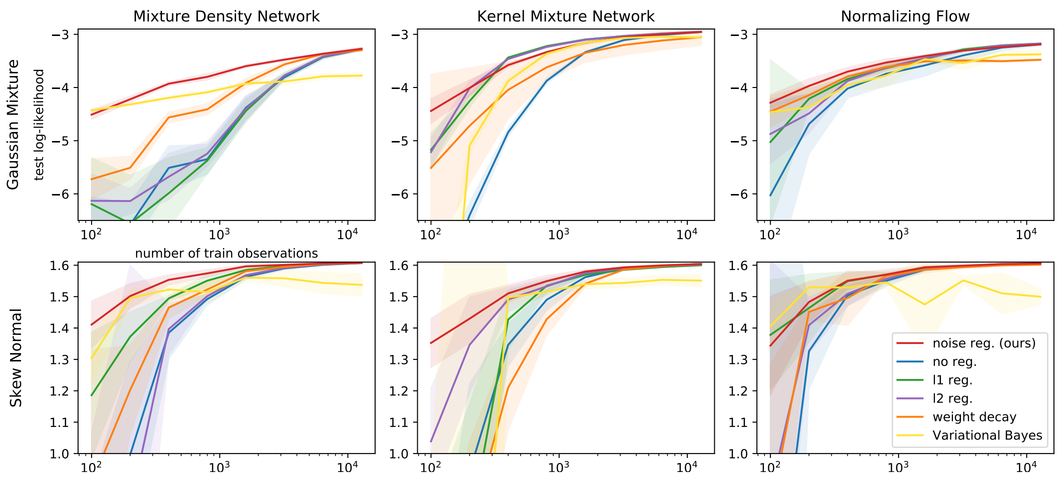

Figure 5.5.: Results of running our three parametric conditional density estimators on the Gaussian Mixture and Skew-normal simulated densities with different regularization techniques applied.

an affine flow. As suspected, the normalizing flows are better at modelling skewed distribution than a mixture of Gaussians, which can be observed in the final plot of figure 5.4.

Next we look at comparing the different regularization techniques for conditional density estimators introduced in section 4. The results are visualized in figure 5.5. The presented noise regularization technique can be seen outperforming all other methods across models and datasets. The other methods' performance is significantly dependent on the specific model being used. For example, we can see the BNN performing somewhat well at regularizing the NFN, while showing poor performance when used with a KMN. As the noise regularization is works in the data-space, irrespective of the parameters, this invariance to the specific model was expected. The Bayesian neural networks has difficulties of converging to an optimal parameter setting, even given large datasets. We suspect this is due to the inherent noise in the gradient estimates.

To summarize the evaluation results on the simulated densities, we can state that the NFN performs slightly better than a MDN while both outperforming the KMN, except for very tailored datasets. The presented noise regularization scheme works well across models and datasets. The other approaches to regularization require significant model- and dataset-specific tuning to reach comparable performances.

5.2. Evaluation on Real-world datasets

We use three common regression benchmark, one location prediction dataset and one financial dataset to evaluate the real-world performance of the NFN and our other CDE models.

5.2.1. UCI benchmarks

The regression datasets from the UCI repositories (Dua and Graff, 2017) are used as common benchmarks for regression models. We test our estimators on three of them:

- **Boston housing dataset**: This small dataset with only 506 datapoints contains information collected by the U.S. Census service concerning the price of housing in the area around Boston. The prediction target is the value of the house in question.

  URL: https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

- **Concrete compressive strength datset** (Yeh, 1998): This dataset contains eight explanatory variables describing the composition and attributes of a piece of concrete. The prediction target is the

		Euro Stoxx	NYC Taxi	Boston	Concrete	Energy
MDN	noise (ours)	**3.94±0.03**	**5.25±0.04**	**-2.49±0.11**	**-2.92±0.08**	**-1.04±0.09**
	weight decay	3.78±0.06	5.07±0.04	-3.29±0.32	-3.33±0.14	-1.21±0.10
	l1 reg.	3.19±0.19	5.00±0.05	-4.01±0.36	-3.87±0.29	-1.44±0.22
	l2 reg.	3.16±0.21	4.99±0.04	-4.64±0.52	-3.84±0.26	-1.55±0.26
	Bayes	3.26±0.43	5.08±0.03	-3.46±0.47	-3.19±0.21	-1.25±0.23
KMN	noise (ours)	**3.92±0.01**	**5.39±0.02**	**-2.52±0.08**	**-3.09±0.06**	**-1.62±0.06**
	weight decay	3.85±0.03	5.31±0.02	-2.69±0.15	-3.15±0.06	-1.79±0.12
	l1 reg.	3.76±0.04	**5.39±0.02**	-2.75±0.13	-3.25±0.07	-1.82±0.10
	l2 reg.	3.71±0.05	5.37±0.02	-2.66±0.13	-3.18±0.07	-1.79±0.13
	Bayes	3.33±0.02	4.47±0.02	-3.40±0.11	-4.08±0.05	-3.65±0.07
NFN	noise (ours)	**3.90±0.01**	**5.20±0.03**	**-2.48±0.11**	**-3.03±0.13**	-1.21±0.08
	weight decay	3.82±0.06	5.19±0.03	-3.12±0.39	-3.12±0.14	-1.22±0.16
	l1 reg.	3.50±0.10	5.12±0.05	-12.58±12.76	-3.91±0.52	-1.29±0.16
	l2 reg.	3.50±0.09	5.13±0.05	-14.22±9.60	-3.99±0.66	-1.34±0.19
	Bayes	3.34±0.33	5.10±0.03	-5.99±2.45	-3.55±0.46	**-1.11±0.22**

Table 5.1.: Comparison of various regularization methods for three neural network based CDE models across 5 data sets. We report the test log-likelihood and its respective standard deviation (higher log-likelihood values are better).

compressive strength.

URL: https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

- **Energy efficiency dataset** (Tsanas and Xifara, 2012): The dataset comes from performing energy analysis of different building shapes using a simulation. The predicted density is two-dimensional between the heating and cooling load.

  URL: https://archive.ics.uci.edu/ml/datasets/energy+efficiency

5.2.2. EuroStoxx 50

This is a proprietary dataset from the Computational Risk and Asset Management Research Group (C-RAM) at KIT. It contains data from 3169 trading days between January 2003 until June 2015. We use 14 different explanatory variables to predict the conditional probability density of 1-day log returns. This dataset gives us a way to apply our estimator to real-world financial data, where uncertainty quantification is important and densities are often severely non-Gaussian.

5.2.3. NYC Yellow Taxi

This dataset contains information on trips of NYC taxis in the Manhattan area recorded during January 2016. We predict the spacial drop-off location for each trip conditioned on the pickup location, the day of the week and the time of day. We follow the setup in Dutordoir et al. (2018). The weekday and time are represented as sine and cosine with natural periods, resulting in 6 explanatory variables when combined with the 2D pickup location From the ca. 1 million trips we sample 10.000 trips to train on.

The full dataset is available at: http://www.andresmh.com/nyctaxitrips/

5.2.4. Results

When looking at the evaluation of the regularization techniques on the various real-world datasets in table 5.1, we can draw similar conclusions as for the simulated densities. The NFN's performance is comparable with a MDN, with the MDN having a slight advantage on some datasets. When looking at the effectiveness of noise regularization, it performs better than all other regularization approaches across datasets and models. Among the other, worse-performing approaches, weight decay shows the most consistent performance across datasets. L1/L2 regularization and the Bayesian neural network exhibit very variable results. The BNN for example is the preferable choice for the NFN on Energy, but a poor choice on Boston, where it exhibits very high variance.

6. Outlook and further work

Compared to previous high-capacity models in CDE, such as the MDN and KMN, that rely on mixture densities, the NFN uses a novel and fundamentally different approach of representing the conditional density. The planar and radial flows used in this work were originally designed to model much higher-dimensional densities than what we used them on in this thesis. Our simulated as well as real-world datasets never exceeded 2 dimensions in the target variable $\boldsymbol{y}$, whereas the original paper used planar and radial flows to model posteriors in VI that are of much higher dimension (Rezende and Mohamed, 2015). Future work could investigate more expressive flows such as Deep Dense Sigmoidal Flows (Huang et al., 2018) that implement an invertible neural network. They are computationally more complex than planar or radial flows with their determinant calculation lying in $O(d^3)$, where d is the dimension of the distribution being modeled. Yet the are significantly more expressive and might therefore be effective in low dimensional problems. The Sylvester Normalizing Flows (van den Berg et al., 2018) are another option. They are a generalization of the planar flows used in this work. Whereas planar flows can be interpreted as a single-layer, single-unit MLP with a skip connection, Sylvester NFs remove this bottleneck by allowing one to add more units. Their determinant lies in $O(M)$, where M is a hyperparameter that tunes the expressiveness.

Further improvements should be possible by advancing the implementation of the Bayesian neural network given in section 4.2. Previous work has seen improved convergence using annealing (Rezende and Mohamed, 2015). This describes the technique of slowly ramping up the contribution of the KL divergence to the ELBO loss, reducing the likelihood of getting stuck in local minima. For the approximate posterior family used in the Bayesian neural network, two approaches can be pursued. One option is to implement a more expressive posterior distribution like the IAF, allowing for a closer approximation of the true posterior. This has yielded improved results for many other Bayesian models (Kingma et al., 2016). Another option is given in Trippe (2017). They compare more constrained mean-field approximations with fixed variances against more expressive mean-field approximations and generally find improved performance with the less expressive posterior families.

Another interesting direction for future work is a different Bayesian treatment of NFNs w.r.t. regularization. In section 4.2, we regularize the parameters of the neural network through a Bayesian prior. Instead, one could directly treat the parameters of the flow as random variables, allowing us to set prior distributions for them. In particular, posterior inference on latent variables can be performed using amortized variational inference (Zhang et al., 2018), similar to Rezende et al. (2014), where a neural network is used to predict the local variational parameters. This would yield a model similar in structure to the NFN, yet with priors over the flow instead of over the NN, providing a more direct and interpretable means of regularizing the conditional distribution.

7. Conclusion

In this thesis we present the Normalizing Flow Network, a novel conditional density model designed to accurately estimate low-dimensional, non-Gaussian densities. We compare this model against the Mixture Density Network and Kernel Mixture Network across 8 datasets. We find the NFN to perform favourably on non-Gaussian densities, while delivering performance comparable to a Mixture Density Network on other densities. We highlight the expandability and configureability of normalizing flows, the density model used by the NFN, showing how they provide a novel way of efficiently parametrizing a wide variety of distributions.

As a second contribution, we review common neural network regularization approaches and test their effectiveness when used in the context of CDE. We show how Variational Inference and Bayesian Neural Networks can be used for regularization. Most regularization methods work in the parameter space, making the biases they introduce more difficult to analyze in the context of CDE. In this light, we propose a novel regularization approach, noise regularization for CDE, that works in the data space and does not have these shortcomings. We show mathematically how noise regularization introduces favourable biases by incentivizing smooth distribution estimates. Empirically, we demonstrate that noise regularization outperforms other regularization techniques across models and datasets. Its effectiveness, together with its ease of implementation, makes it the preferred method for regularizing high-capacity CDE models like the NFN.

Bibliography

L. Ambrogioni, U. Güçlü, M. A. J. van Gerven, and E. Maris. The Kernel Mixture Network: A Nonparametric Method for Conditional Density Estimation of Continuous Random Variables. *arXiv:1705.07111 [stat]*, May 2017. 22

A. Azzalini. The skew-normal distribution and related multivariate families. *Scandinavian Journal of Statistics*, 32(2):159–188, 2005. 32

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017. 2

C. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag, New York, 2006. ISBN 978-0-387-31073-2. 4

C. M. Bishop. Mixture density networks. Technical report, Citeseer, 1994. 21

C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1): 108–116, 1995. 28

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Apr. 2017. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.2017.1285773. 2, 25, 26

N. Damodar. *Basic Econometrics*. The Mc-Graw Hill, 2004. 6

M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, Mar. 2020. ISBN 978-1-108-45514-5. 24

L. Devroye. The Equivalence of Weak, Strong and Complete Convergence in L_1 for Kernel Density Estimates. *The Annals of Statistics*, 11(3):896–904, Sept. 1983. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1176346255. 5

L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 16

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 16

D. Dua and C. Graff. UCI Machine Learning Repository. 2017. 34

V. Dutordoir, H. Salimbeni, J. Hensman, and M. Deisenroth. Gaussian process conditional density estimation. In *Advances in Neural Information Processing Systems*, pages 2385–2395, 2018. 35

R. Engle. GARCH 101: The Use of ARCH/GARCH Models in Applied Econometrics. *Journal of Economic Perspectives*, 15(4):157–168, Dec. 2001. ISSN 0895-3309. doi: 10.1257/jep.15.4.157. 2

R. F. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4):987–1007, 1982. ISSN 0012-9682. doi: 10.2307/1912773. 6

M. A. T. Figueiredo. Adaptive sparseness for supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1150–1159, Sept. 2003. ISSN 0162-8828. doi: 10.1109/TPAMI.2003.1227989. 25

C. S. Forbes, M. A. Evans, C. Forbes, M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions / Catherine Forbes ...* 2011. ISBN 978-0-470-39063-4. 8

M. Germain, K. Gregor, I. Murray, and H. Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015. 17, 18

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014. 15

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep Learning*, volume 1. MIT press Cambridge, 2016. 23, 24, 28

J. D. Hamilton. *Time Series Analysis*, volume 2. Princeton university press Princeton, NJ, 1994. 6

S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, pages 177–185, 1989. 2, 23

M. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic Variational Inference. *arXiv:1206.7051 [cs, stat]*, June 2012. 27

L. Holmstrom and P. Koistinen. Using additive noise in back-propagation training. *IEEE transactions on neural networks*, 3(1):24–38, 1992. 28

C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*, 2018. 19, 20, 37

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999. 25

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 12, 23

D. P. Kingma and P. Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. *arXiv:1807.03039 [cs, stat]*, July 2018. 2, 17

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, Dec. 2013. 15, 26, 27

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016. 2, 19, 28, 37

A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, pages 950–957, 1992. 23

J. Kukačka, V. Golkov, and D. Cremers. Regularization for Deep Learning: A Taxonomy. *arXiv:1710.10686 [cs, stat]*, Oct. 2017. 23

Q. Li and J. S. Racine. *Nonparametric Econometrics: Theory and Practice*. Princeton University Press, 2007. ISBN 978-0-691-12161-1. 5

I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. *arXiv:1711.05101 [cs, math]*, Nov. 2017. 24

B. B. Mandelbrot. Scaling in financial prices: I. Tails and dependence. *Quantitative Finance*, 1(1): 113–123, Jan. 2001. ISSN 1469-7688. doi: 10.1080/713665539. 6

S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. Monte Carlo Gradient Estimation in Machine Learning. *arXiv preprint arXiv:1906.10652*, 2019. 26

NALP. NALP - National Association for Law Placement | Salary Distribution Curve for the Class of 2009 Shows Relatively Few Salaries Were Close to the Mean. https://www.nalp.org/startingsalarydistributionclassof2009, July 2010. 7

R. M. Neal. *Bayesian Learning for Neural Networks*, volume 118. Springer Science & Business Media, 2012. 2, 24

G. Papamakarios, T. Pavlakou, and I. Murray. Masked Autoregressive Flow for Density Estimation. *arXiv:1705.07057 [cs, stat]*, May 2017. 18

E. Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, Sept. 1962. ISSN 0003-4851, 2168-8990. doi: 10.1214/aoms/1177704472. 5

D. J. Rezende and S. Mohamed. Variational Inference with Normalizing Flows. *arXiv:1505.05770 [cs, stat]*, May 2015. 11, 12, 37

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014. 15, 37

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. 23

M. Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832–837, Sept. 1956. ISSN 0003-4851, 2168-8990. doi: 10.1214/aoms/1177728190. 5

J. Rothfuss, F. Ferreira, S. Boehm, S. Walther, M. Ulrich, T. Asfour, and A. Krause. Noise Regularization for Conditional Density Estimation. *arXiv:1907.08982 [cs, stat]*, July 2019. 3, 28, 33

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016. 11

B. W. Silverman. Using Kernel Density Estimates to Investigate Multimodality. *Journal of the Royal Statistical Society. Series B (Methodological)*, 43(1):97–99, 1981. ISSN 0035-9246. 6

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 24

E. V. Strobl and S. Visweswaran. Dirac Delta Regression: Conditional Density Estimation with Clinical Trials. *arXiv:1905.10330 [cs, stat]*, May 2019. 2

E. G. Tabak and C. V. Turner. A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. ISSN 1097-0312. doi: 10.1002/cpa.21423. 9, 12, 13

E. G. Tabak and E. Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010. 9

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 24

M. Titsias and M. Lázaro-Gredilla. Doubly Stochastic Variational Bayes for non-Conjugate Inference. In *International Conference on Machine Learning*, pages 1971–1979, Jan. 2014. 27

B. L. Trippe. Complex Uncertainty in Machine Learning. page 102, Dec. 2017. 37

B. L. Trippe and R. E. Turner. Conditional Density Estimation with Bayesian Normalising Flows. *arXiv:1802.04908 [stat]*, Feb. 2018. 12, 13

A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012. 35

R. van den Berg, L. Hasenclever, J. M. Tomczak, and M. Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018. 16, 37

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499 [cs]*, Sept. 2016a. 2, 18

A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel Recurrent Neural Networks. *arXiv:1601.06759 [cs]*, Jan. 2016b. 18

L. Weng. Flow-based Deep Generative Models. https://lilianweng.github.io/2018/10/13/flow-based-deep-generative-models.html, Oct. 2018. 15

I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. 34

C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 37

A. Appendix

A.1. Hyperparameter settings

	MDN	KMN	NFN
hidden layer sizes	(32,32)	(32,32)	(32,32)
hidden non-linearity	tanh	tanh	tanh
training epochs	1000	1000	1000
Adam learning rate	0.001	0.001	0.001
K: number of components	20	50	1 affine + 10 radial flows
data normalization	True	True	True
initialization of scales	-	[0.7, 0.3]	-
trainable scales	-	True	-

Table A.1.: The default configuration of the estimators before any regularisation is applied. This configuration was determined through hyperparameter optimization on the simulated datasets.