Investigating the solar atmosphere using machine learning techniques

Study notes

Daniel Zahnd

February 22, 2023 - July 26, 2023

Contents

1 Machine learning

The field of artificial intelligence (AI) is a fascinating, but relatively new field of research dedicated to mimic the process of learning information. In order to define a learning process, another quantity has first to be defined, namely information. [Mitchell, 1997, p.2] defines learning such that a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. If the so-called information entropy serves as a performance measure P, the process of learning can also be described by the statement, that any learning process decreases the information entropy of some system under consideration.

The concepts of learning and information are explored in further detail in the subsequent material below.

1.1 Overview of artificial intelligence and related (sub)-fields

Text.

1.2 Overview of machine learning and related (sub)-fields

The foundational tools of machine learning are so-called neural networks, also called deep neural networks. The basic building block is the single layer perceptron (SLP). By combining several single layer perceptrons to a multi layer perceptron (MLP), a deep neural network is born. Given an input vector $\boldsymbol{x} \in \mathbb{R}^d$ and an output vector $\bar{\boldsymbol{x}} \in \mathbb{R}^w$, the working principle of single and multilayer perceptrons are shown in fig. 1 and fig. 2.

1.3 The perceptron

1.3.1 Single-layer perceptron

In fig. 1, a single-layer perceptron is visualized.

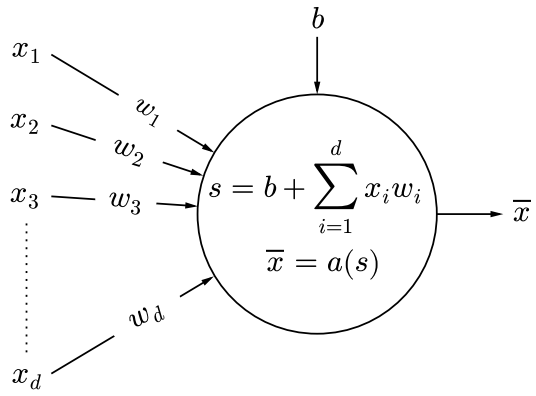


Figure 1: Working principle of a single-layer perceptron; hereby, a represents a nonlinear activation function $a : \mathbb{R} \to \mathbb{R}$, $s \mapsto a(s)$.

1.3.2 Multi-layer perceptron

In fig. 2, a multi-layer perceptron is visualized.

1.4 Deep neural networks

1.4.1 Structure of neural networks

Text.

1.4.2 Training process

Text.

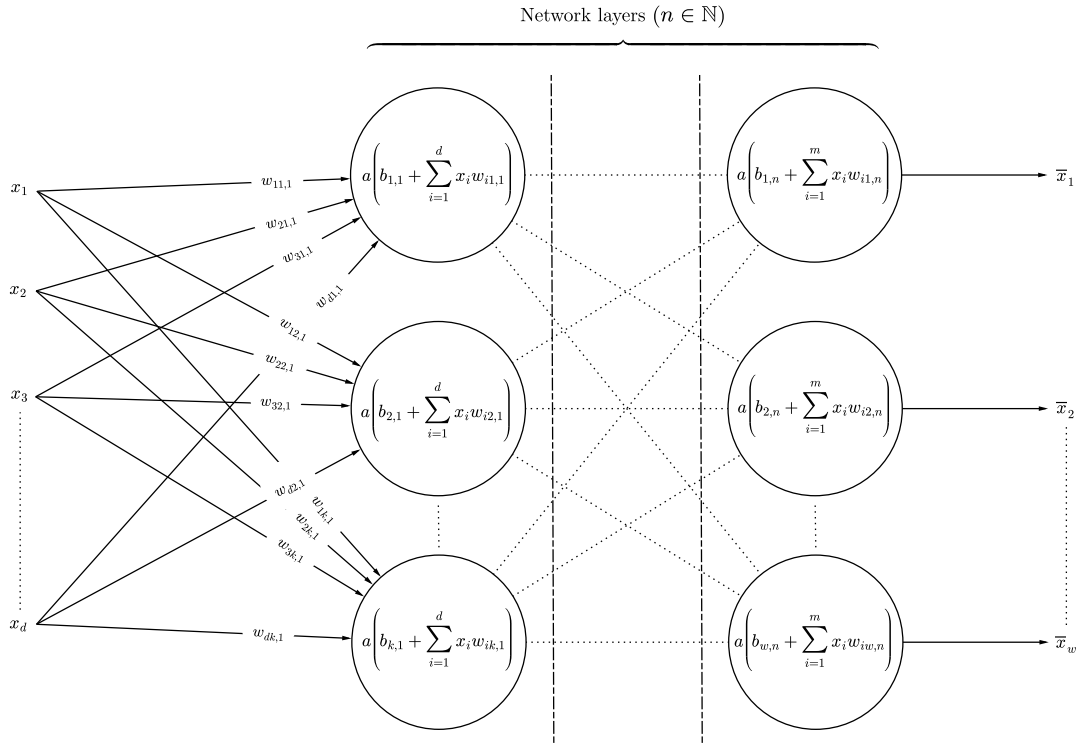


Figure 2: Working principle of a multi-layer perceptron; hereby, a represents a nonlinear activation function $a : \mathbb{R} \to \mathbb{R}$, $s \mapsto a(s)$.

1.5 Deep generative models

Text. Elaborate on GAN's, VAE's and Flows, especially normalizing flows. Comparison of these three techniques. See blog of Lilian Weng.

1.6 Role of CPU's and GPU's in machine learning tasks

So-called CPU's (central processing unit) are those hardware components in a computer, which executes most of the computer's hardware and software instructions. CPU's performs tasks sequentially; therefore a CPU can only handle multiple tasks at the same time if there are multiple cores in the CPU.

So-called GPU's (graphics processing unit) are those hardware components in a computer, which render intensive high-resolution graphics or images. GPU's can perform tasks in parallel processing; therefore a GPU can handle multiple tasks at the same time by splitting up a task in multiple sub-tasks and dividing them among the vast number of processing cores in the GPU.

A CPU typically operates much faster than a GPU, but can handle tasks only sequentially. A GPU however typically consists of a vast number of cores allowing for efficient multitasking. In general therefore, a GPU can handle intensive calculations faster than

a CPU, because it allows for dividing a task in multiple sub-tasks and distributing them among the vast number of processing cores, whereas the CPU cannot do that but can only handle tasks in a sequential way.

For machine learning therefore, it is often the case that a GPU's are required for time-efficient computing.

2 Information theory

2.1 Self-information

The basic intuition behind information theory is the requirement, that learning that an unlikely event has occurred should convey more information than learning, that a likely event has occurred. The quantification of information in computer science relies therefore on three key assumptions/requirements:

(1) Likely events should have low information content. Guaranteed events are required to have no information content.

(2) Unlikely events should have high information content.

(3) Independent events should have additive information; i.e. learning about two independent events of the same likelihood should convey twice as much information as learning about the occurrence of only one of these events.

These requirements are satisfied by the so-called self-information $I(x)$ of a random variable X with probability distribution $p(x)$ defined as

$$I[p(x)] = -\ln[p(x)]\,. \tag{1}$$

By this definition, information $I(U)$ of an unlikely event $x = U$ will be higher than information $I(L)$ of a likely event $x = L$, because $I(U) = -\ln[p(U)] = \ln[1/p(U)] \gg 1$ and $I(L) = -\ln[p(L)] = \ln[1/p(L)] \geq 0$ and therefore $I(U) \gg I(L)$. Furthermore, information about two independent events $x = A$ and $x = B$ are additive, since

$$I(A\cap B) = -\ln[p(A\cap B)] = -\ln[p(A)p(B)] = -(\ln[p(A)]+\ln[p(B)]) = I(A)+I(B). \tag{2}$$

2.2 Information entropy

Information entropy $H(x)$ regarding a random variable X with probability density function $p(x)$ - called Shannon entropy, if the random variable is discrete, whereas called differential entropy, if the random variable is continuous - is defined by the equation

$$H_p[p(x)] = E_{x\sim p}(I[p(x)]) = -E_{x\sim p}\left(\ln[p(x)]\right). \tag{3}$$

Information entropy defined in this way is a measure of the amount of uncertainty in a probability distribution and in a statistical model for that matter. The information entropy will be very low, if $p(x)$ describes mostly very likely events, whereas it will

be very high, if $p(x)$ describes mostly very unlikely events. This can also be stated in the following way: Probability distributions that are nearly deterministic (delta-function shaped) have a low entropy, whereas distributions that are closer to a uniform distribution have high entropy.

2.3 Kullback-Leibler divergence

The Kullback-Leibler divergence is a means to compute how different two probability distributions $p(x)$ and $q(x)$ over the same random variable X are. It is defined as

$$D_{KL}(p||q) = E_{x\sim p}\left[\ln\left(\frac{p(x)}{q(x)}\right)\right] = E_{x\sim p}\left(\ln[p(x)] - \ln[q(x)]\right). \tag{4}$$

As such, the Kullback-Leibler divergence is nothing more than the difference between a so-called cross-entropy $H_p[q(x)]$ and an entropy $H_p[q(x)]$, namely

$$\begin{aligned} D_{KL}(p||q) = H_p[q(x)] - H_p(p(x)) &= -E_{x\sim p}(\ln[q(x)]) + E_{x\sim p}(\ln[p(x)]) \\ &= E_{x\sim p}(\ln[p(x)] - \ln[q(x)]). \end{aligned} \tag{5}$$

The Kullback-Leibler divergence will be low, if $p(x)$ and $q(x)$ are very similar; it will be high, if $p(x)$ and $q(x)$ are very different.

As information entropy is a measure for uncertainty in a probability distribution, divergence is to be interpreted as additional uncertainty accounting for the distance between two probability distributions over the same random variable. If $p(x)$ is the target probability distribution, i.e. the true distribution for the random variable X, and if $q(x)$ is the model probability distribution, i.e. the learned distribution, the Kullback-Leibler divergence provides a measure on how close the model probability distribution $q(x)$ is to the target distribution $p(x)$. Minimizing the Kullback-Leibler divergence is equivalent to finding the best model probability distribution $q(x)$ in terms of optimal resemblance to the target distribution $p(x)$.

3 Normalizing flows

3.1 Introduction

Normalizing flows are useful to learn complex data distributions from available samples, be they experimentally obtained or of a synthesized nature.

The goal of a normalizing flow is to learn a certain probability density function $p_{\boldsymbol{x}}(\boldsymbol{x})$ of some quantity represented by a potentially multivariate random variable $\boldsymbol{x}$. Given samples from $p_{\boldsymbol{x}}(\boldsymbol{x})$ obtained by either experimental or synthetical procedure, the normalizing flow can learn the probability density function by means of finding a diffeomorphism, which maps samples from a standard normal distribution to $p_{\boldsymbol{x}}(\boldsymbol{x})$. Since a diffeomorphism is invertible by definition, a trained normalizing flow can then be used to draw samples from $p_{\boldsymbol{x}}(\boldsymbol{x})$ at will; this can be done by sampling from a standard

normal distribution and subsequent application of the inverse learned diffeomorphism to the generated samples. This will result in a set of samples from $p_{\boldsymbol{x}}(\boldsymbol{x})$.

This technique can be used for various purposes; for example, for inversion problems in (stellar) atmospheric physics, as it is being explored in the subsequent material.

3.2 Change of variable formula in probability theory

The change of variable formula from probability theory can be derived using the substitution formula for integrals. The following theorem is first proven in the univariate case and then generalized to the multivariate case.

Theorem 3.2.1. *Let x and z be continuous random variables with associated probability density functions $p_x(x)$ and $p_z(z)$. Furthermore, let z be the transformation of x by an invertible and strictly increasing continuously differentiable function $f(x)$, such that $z = f(x)$ and $x = f^{-1}(z)$ hold; in this case the change of variable formula from probability theory applies as*

$$p_x(x) = p_z(f(x))\frac{\mathrm{d}f(x)}{\mathrm{d}x}. \tag{6}$$

Proof. First, let with a and b be defined real numbers with $a < b$. By definition of the transformation $f(x)$ let with c and d be defined the real numbers $c = f(a)$ and $d = f(b)$, hence

$$P(a \leq x \leq b) = P(c \leq z \leq d) \tag{7}$$

must hold, where this quantity is defined as $P(a \leq x \leq b) \doteq \int_a^b p_x(x)\,\mathrm{d}x$ and therefore represents the probability mass for an experiment outcome of x to be between a and b. Since x and z are continuous random variables with associated valid probability density functions, one can write

$$P(c \leq z \leq d) = \int_c^d p_z(z)\,\mathrm{d}z = \int_{f(a)}^{f(b)} f_z(z)\,\mathrm{d}z. \tag{8}$$

By the substitution rule for integrals one can substitute in the above integral $z(x) = f(x)$ and

$$\frac{\mathrm{d}z(x)}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} \quad \Leftrightarrow \quad \mathrm{d}z = \mathrm{d}z(y) = \frac{\mathrm{d}f(x)}{\mathrm{d}x}\,\mathrm{d}x. \tag{9}$$

Therefore, the relation

$$P(c \leq z \leq d) = \int_{f(a)}^{f(b)} p_z(z)\,\mathrm{d}z = \int_a^b p_z(f(x))\frac{\mathrm{d}f(x)}{\mathrm{d}x}\,\mathrm{d}x = P(a \leq x \leq b) \tag{10}$$

obtains, thus

$$p_x(x) = p_z(f(x))\frac{\mathrm{d}f(x)}{\mathrm{d}x} \tag{11}$$

must hold, where $\mathrm{d}f(x)/\mathrm{d}x$ acts as a normalizing factor for the new probability density $p_x(x)$. □

Let now $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^d$, $d \in \mathbb{N}$ be continuous multivariate random variables with associated probability density functions $p_{\boldsymbol{x}}(\boldsymbol{x})$ and $p_{\boldsymbol{z}}(\boldsymbol{z})$. Furthermore, let

$$\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) \tag{12}$$

be a diffeomorphism transforming $\boldsymbol{x}$ to $\boldsymbol{z}$. In this case, the formula generalizes to the multivariate case as

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}[\boldsymbol{f}(\boldsymbol{x})] \left| \det \left(\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right| = p_{\boldsymbol{z}}[\boldsymbol{f}(\boldsymbol{x})] \left| \det \begin{pmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_d(\boldsymbol{x})}{\partial x_d} \end{pmatrix} \right|. \tag{13}$$

3.3 General principle of a normalizing flow

3.3.1 Conceptual formulation of a normalizing flow

Consider some vector $\boldsymbol{x} \in \mathbb{R}^d$ with d, $q \in \mathbb{N}$ and a latent vector $\boldsymbol{z} \in \mathbb{R}^d$. The probability densities for these vectors shall be given by

$$\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x}), \quad \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^d. \tag{14}$$

The aim of a normalizing flow is to learn a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}) \tag{15}$$

transforming the probability density function $p_{\boldsymbol{x}}$ to $p_{\boldsymbol{z}}$. Hereby, $\boldsymbol{\phi}(\hat{\mathbf{x}})$ are weights depending on training data $\hat{\mathbf{x}} \in \mathbb{R}^{d \times q}$, which are associated to a deep neural network implementing the diffeomorphic function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$. Given such a function, using the change of variable formula the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ can be calculated as

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})] \left| \det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|. \tag{16}$$

Knowing the transformation $\boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ and given that $\boldsymbol{z} \sim \mathcal{N}(0,1)^d$, one can easily calculate samples from the distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$ by means of sampling $\boldsymbol{z}$ from a d-dimensional standard normal distribution and obtaining the corresponding $\boldsymbol{x}$ plugging $\boldsymbol{z}$ into the inverse function $\boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$, such that $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{z})$.

3.3.2 Requirements for a normalizing flow

Consider a function

$$\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) \tag{17}$$

composed of several functions $\boldsymbol{f}_{(k)}$, $k \in \{1, \ldots, n\}$, $n \in \mathbb{N}$, such that

$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_{(n)} \circ \cdots \circ \boldsymbol{f}_{(1)}(\boldsymbol{x}). \tag{18}$$

In this case, the transformation functions $\boldsymbol{f}_{(k)}$ aswell as the function $\boldsymbol{f}$ as a whole should satisfy several conditions to serve purposefully as a tool to sample from a probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ given samples $\boldsymbol{z}$ obtained from a standard normal distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$. According to [Kobyzev et al., 2021], all functions $\boldsymbol{f}_{(k)}$ should satisfy the following conditions:

(1) All transformation functions $\boldsymbol{f}_{(k)}$ need to be invertible and differentiable. In order to transform one probability density into another density, it is necessary to calculate the Jacobian of the transformation function, as stated by the change of variable theorem. Furthermore, the transformation functions should be invertible, because one would like to sample from a standard normal distribution, plug the obtained values $\boldsymbol{z}$ into the inverse transformation function and thereby obtaining samples $\boldsymbol{x}$. Both of these conditions - invertibility and differentiabiliby - are satisfied by diffeomorphic functions.

(2) The transformation functions $\boldsymbol{f}_{(k)}$ need to be expressive enough to model real data distributions. The normalizing flow composed of these transformation functions needs to learn a real data distribution; hence enough flexibility of the transformation functions is required in order to allow the neural networks composing the flow to learn actual and potentially complicated probability distributions.

(3) The transformation functions $\boldsymbol{f}_{(k)}$ should be computationally efficient. That is, the Jacobian determinant should be calculable in an as efficient as possible way; furthermore also the application of the trained forward and inverse normalizing flow should consume as little as possible time. In order to train and apply the normalizing flow, i.e. the transformation functions composing the normalizing flow, many Jacobian determinants need to be calculated, as required by the change of variable formula. Therefore, transformation functions with a simple Jacobians allowing for quick and easy calculations of the corresponding determinants are preferable.

3.3.3 Normalizing flow with affine coupling layers

Consider a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}), \tag{19}$$

which is defined with the help of a neural network possessing network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}})$ depending on the training data $\hat{\mathbf{x}} \in \mathbb{R}^{d \times q}$ used to train the network. The probability density for $\boldsymbol{x}$ as calculated via $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{z})$ therefore becomes conditional on the network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}})$. Hence for $p_{\boldsymbol{x}}(\boldsymbol{x})$, the notation $p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ is introduced.

With the change of variable theorem, one obtains

$$1 = \int_{\mathbb{R}^d} p_{\boldsymbol{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = \int_{\mathbb{R}^d} p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})] \left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right| \mathrm{d}\boldsymbol{x} = \int_{\mathbb{R}^d} p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \tag{20}$$

from which it follows, that the probability density $p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ can be expressed by means of the diffeomorphism $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$, the Jacobian $\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})/\partial \boldsymbol{x}$ and the probability density $p_{\boldsymbol{z}}(\boldsymbol{z})$, namely

$$p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}) = p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})] \left| \det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|. \tag{21}$$

Since a concatenation of diffeomorphisms is again a diffeomorphism, the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$ can be composed of many diffeomorphisms. Let $\boldsymbol{f}_{\boldsymbol{\phi},(1)}, \ldots, \boldsymbol{f}_{\boldsymbol{\phi},n}$ with $n \in \mathbb{N}$ be a number of diffeomorphisms, such that

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})} = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)}. \tag{22}$$

Introducing the notation $\boldsymbol{x} \doteq \boldsymbol{z}_{(0)}$, $\boldsymbol{z} \doteq \boldsymbol{z}_{(n)}$ and $\boldsymbol{z}_{(k)} \doteq \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ with $k \in \{1, \ldots, n\}$, the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$ can be written as

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{z}_{(0)}) = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)}(\boldsymbol{z}_{(0)}). \tag{23}$$

In fig. 3, an overview of the working principle of a normalizing flow is visualized. Be-

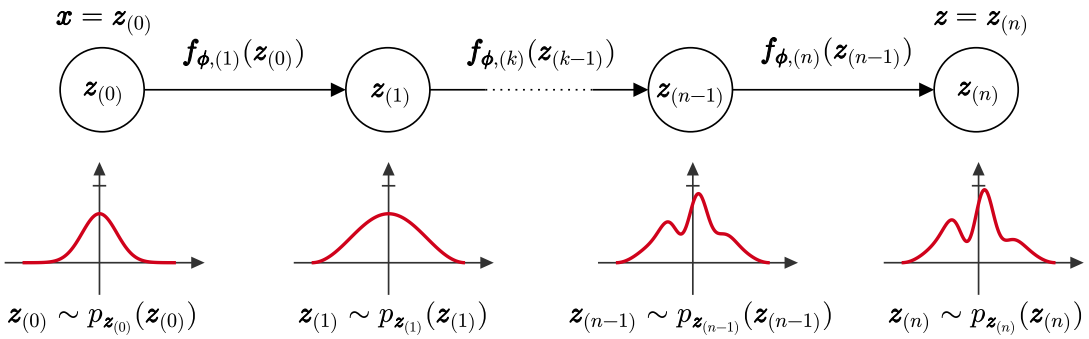


Figure 3: Visualization of the working principle of normalizing flows.

cause $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ is a concatenation of functions $\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ with $k \in \{1, \ldots, n\}$, the determinant of the Jacobian for $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is given by

$$\det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) = \prod_{k=1}^{n} \det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} \right), \tag{24}$$

where the Jacobians on the right-hand side take the form

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} = \begin{pmatrix} \frac{\partial f_{\boldsymbol{\phi},(k),1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \end{pmatrix}, \quad k \in \{1, \ldots, n\}. \tag{25}$$

Every function $\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ for $k \in \{1, \ldots, n\}$ represents one of the n so-called coupling layers constituting the total flow function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$. Every composite function $\boldsymbol{f}_{\boldsymbol{\phi},(k)}$ is

implemented by application of neural networks depending on $\boldsymbol{x}$, such that each composite function is fully invertible and differentiable. The total set of parameters learned by the neural networks is then identified with $\boldsymbol{\phi}(\hat{\mathbf{x}})$, hence defining the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$.

In the case of $\boldsymbol{f}_{\boldsymbol{\phi},(k)}$ being so-called affine coupling layers, the functions can explicitly written by means of two arbitrary functions

$$\boldsymbol{\mu}_{(k)} : \mathbb{R}^m \to \mathbb{R}^{d-m}, \quad \boldsymbol{\sigma}_{(k)} : \mathbb{R}^m \to \mathbb{R}^{d-m}, \quad m \in \{1, \dots, d\}, \quad k \in \{1, \dots, n\} \tag{26}$$

implemented as deep neural networks. That is to say, that the neural networks $\boldsymbol{\mu}_{(k)}$ and $\boldsymbol{\sigma}_{(k)}$ take a vector of dimension m as an input and return a vector of dimension $d-m$ as an output; there can be an arbitrary sequence of network layers and activation functions in between input and output. Every mapping $\boldsymbol{z}_{(k)} = \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ for $k \in \{1, \dots, n\}$ is now defined as an affine transformation, such that

$$f_{\boldsymbol{\phi},(k),l}(\boldsymbol{z}_{(k-1)}) = \begin{cases} z_{(k-1),l} & , \quad l \in \{1, \dots, m\} \\ \mu_{(k),l}(\boldsymbol{z}_{(k-1),1:m}) + e^{\sigma_{(k),l}(\boldsymbol{z}_{(k-1),1:m})} \cdot z_{(k-1),l} & , \quad l \in \{m+1, d\} \end{cases} \tag{27}$$

and

$$f^{-1}_{\boldsymbol{\phi},(k-1),l}(\boldsymbol{z}_{(k)}) = \begin{cases} z_{(k),l} & , \quad l \in \{1, \dots, m\} \\ [z_{(k),l} - \mu_{(k),l}(\boldsymbol{z}_{(k),1:m})] \cdot e^{-\sigma_{(k),l}(\boldsymbol{z}_{(k),1:m})} \cdot z_{(k),l} & , \quad l \in \{m+1, d\} \end{cases}, \tag{28}$$

where $f_{\boldsymbol{\phi},(k)} = (f_{\boldsymbol{\phi},(k),1}, \dots, f_{\boldsymbol{\phi},(k),d})^\top$ and $\boldsymbol{z}_{(k),1:m} = (z_{(k),1}, \dots, z_{(k),m})^\top$.

It is a convenient property of such affine functions $\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ for $k \in \{1, \dots, n\}$, that the Jacobians with respect to $\boldsymbol{z}_{(k-1)}$ are triangular, that is

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} = \begin{pmatrix} \mathbf{1}_m & \mathbf{0}_m \\ \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k),m+1:d}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1),m+1:d}} & \operatorname{diag}\left(e^{\sigma_{(k),m+1:d}(\boldsymbol{z}_{(k-1),1:m})}\right) \end{pmatrix}, \tag{29}$$

where $\mathbf{1}_m$ is a unity matrix and $\mathbf{0}_m$ is a zero matrix of dimension $m \times m$. The lower left quantity in the above matrix is again a Jacobian and takes the form

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k),m+1:d}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1),m+1:d}} = \begin{pmatrix} \frac{\partial f_{\boldsymbol{\phi},(k),m+1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),m+1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),m+1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),m+1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \end{pmatrix}; \tag{30}$$

the lower right quantity however is a diagonal matrix with the elements $e^{\sigma_{(k),l}(\boldsymbol{z}_{(k-1),1:m})}$ for $l \in \{m+1, d\}$ on the diagonal. Therefore, the Jacobian $\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)}) / \partial \boldsymbol{z}_{(k-1)}$ is triangular and hence its determinant is given by the product of the diagonal elements, that is

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right) = \prod_{j=m+1}^{d} e^{\sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m})}. \tag{31}$$

Taking the logarithm of this expression results in a transformation of the product to a sum, such that

$$\log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right)\right] = \sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}) \tag{32}$$

holds. Recall, that the complete normalizing flow is given as a concatenation of affine coupling functions (layers), such that

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{k=1}^{n} \det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right). \tag{33}$$

Taking the logarithm of this expression and inserting the previous result eq. (32), this leads to

$$\begin{aligned}\log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right] &= \sum_{k=1}^{n} \log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right)\right] \\ &= \sum_{k=1}^{n}\sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}).\end{aligned} \tag{34}$$

3.3.4 Training process

Recalling the diffeomorphism $\boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ and the change of variable formula for the multivariate case, the probability density $p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ can be written as found in eq. (21). Taking the natural logarithm of expression eq. (21) and inserting the result eq. (34) into it, one obtains

$$\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})\right] = \log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})]\right) + \left|\sum_{k=1}^{n}\sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m})\right|. \tag{35}$$

Note, that the natural logarithm $\log(a)$ is strictly monotonic increasing for $a > 0$; furthermore, $-\infty < \log(a) \leq 0$ for $0 < a \leq 1$. Therefore, maximizing the so-called likelihood $p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ is equivalent to maximizing $\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})\right]$, which is named the log-likelihood. Maximizing $p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ corresponds to finding optimal network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}})$ for the deep neural networks $\boldsymbol{\mu}_{(k)}$ and $\boldsymbol{\sigma}_{(k)}$ for $k \in \{1, \ldots, n\}$ constituting the flow function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$, such that the most suitable model parameters $\boldsymbol{x}$ given some context $\boldsymbol{y}$ are found. In the case of a stellar atmosphere inversion, $p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ represents a probability distribution for the solutions $\boldsymbol{x}$ of an inversion $\boldsymbol{x} = \boldsymbol{M}^{-1}(\boldsymbol{y})$, where $\boldsymbol{M}$ is the atmosphere model and $\boldsymbol{y}$ are observations.

That is to say, that the normalizing flow, i.e. the neural networks constituting it, are trained by means of minimizing the negative log-likelihood $-\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})\right]$, which is equivalent to maximizing positive log-likelihood and hence also to maximizing the

positive likelihood. Therefore, the loss function to minimize by any gradient-descent and backpropagation algorithm employed can be defined as

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}) &= -\log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})]\right) - \log\left(\left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|\right) \\ &= -\log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})]\right) - \left|\sum_{k=1}^{n}\sum_{j=m+1}^{d}\sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m})\right|.\end{aligned} \tag{36}$$

This loss function is minimized with respect to the neural network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}})$.

3.4 Normalizing flow with context and affine coupling layers

3.4.1 Conceptual formulation and terminology

Consider a model $\boldsymbol{M}(\boldsymbol{x}) = \boldsymbol{y}$, that produces observations $\boldsymbol{y} \in \mathbb{R}^D$, $D \in \mathbb{N}$ based on model parameters $\boldsymbol{x} \in \mathbb{R}^d$, $d\,q \in \mathbb{N}$. Furthermore, consider a latent vector $\boldsymbol{z} \in \mathbb{R}^d$. The probability densities for these vectors are given by

$$\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x}), \quad \boldsymbol{y} \sim p_{\boldsymbol{y}}(\boldsymbol{y}), \quad \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^d. \tag{37}$$

The aim of the normalizing flow is to model the probability density $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$, namely the probability density function of $\boldsymbol{x}$, given a certain observation $\boldsymbol{y}$, thereby defining an inversion of the model $\boldsymbol{M}(\boldsymbol{x}) = \boldsymbol{y}$. The multidimensional quantity $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is a set of parameters learned by a neural network, which is used to implement the normalizing flow.

3.4.2 Construction of a normalizing flow with context and affine coupling layers

Consider a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}), \tag{38}$$

where $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ are weights depending on training data $\hat{\mathbf{x}} \in \mathbb{R}^{d\times q}$ and $\hat{\mathbf{y}} \in \mathbb{R}^{D\times q}$, which are associated to a deep neural network implementing the diffeomorphic function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$. The probability density for $\boldsymbol{x}$ as calculated via $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{z})$ therefore becomes conditional on the context $\boldsymbol{y}$, because the network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ themselves depend on the context $\hat{\mathbf{y}}$ used to train the network. Hence for $p_{\boldsymbol{x}}(\boldsymbol{x})$, the notation $p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ is introduced.

With the change of variable theorem, one obtains

$$1 = \int_{\mathbb{R}^d} p_{\boldsymbol{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = \int_{\mathbb{R}^d} p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|\,\mathrm{d}\boldsymbol{x} = \int_{\mathbb{R}^d} p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}, \tag{39}$$

from which it follows, that the probability density $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ can be expressed by means of the diffeomorphism $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$, the Jacobian $\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})/\partial \boldsymbol{x}$ and the probability density $p_{\boldsymbol{z}}(\boldsymbol{z})$, namely

$$p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y}) = p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})] \left| \det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|. \tag{40}$$

This identity is to be considered as the heart of the normalizing flow technique.

Since a concatenation of diffeomorphisms is again diffeomorphism, the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$ can be composed of many diffeomorphisms. Let $\boldsymbol{f}_{\boldsymbol{\phi},(1)}, \ldots, \boldsymbol{f}_{\boldsymbol{\phi},n}$ with $n \in \mathbb{N}$ be a number of diffeomorphisms, such that

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)}. \tag{41}$$

Introducing the notation $\boldsymbol{x} \doteq \boldsymbol{z}_{(0)}$, $\boldsymbol{z} \doteq \boldsymbol{z}_{(n)}$ and $\boldsymbol{z}_{(k)} \doteq \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ with $k \in \{1, \ldots, n\}$, the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$ can be written as

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{z}_{(0)}) = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)}(\boldsymbol{z}_{(0)}). \tag{42}$$

Because $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})$ is a concatenation of functions $\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ with $k \in \{1, \ldots, n\}$, the determinant of the Jacobian for $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is given by

$$\det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) = \prod_{k=1}^{n} \det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} \right), \tag{43}$$

where the Jacobians on the right-hand side take the form

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} = \begin{pmatrix} \frac{\partial f_{\boldsymbol{\phi},(k),1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \end{pmatrix}, \quad k \in \{1, \ldots, n\}. \tag{44}$$

Every function $\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ for $k \in \{1, \ldots, n\}$ represents one of the n so-called coupling layers constituting the total flow function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$. Every composite function $\boldsymbol{f}_{\boldsymbol{\phi},(k)}$ is implemented by application of neural networks depending on $\boldsymbol{x}$ and $\boldsymbol{y}$, such that each composite function is fully invertible and differentiable. The total set of parameters learned by the neural networks is then identified with $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, hence defining the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$.

In the case of $\boldsymbol{f}_{\boldsymbol{\phi},(k)}$ being so-called affine coupling layers, the functions can explicitly written by means of two arbitrary functions

$$\boldsymbol{\mu}_{(k)} : \mathbb{R}^{m+D} \to \mathbb{R}^{d-m}, \quad \boldsymbol{\sigma}_{(k)} : \mathbb{R}^{m+D} \to \mathbb{R}^{d-m}, \quad m \in \{1, \ldots, d\}, \quad k \in \{1, \ldots, n\} \tag{45}$$

implemented as deep neural networks. That is to say, that the neural networks $\boldsymbol{\mu}_{(k)}$ and $\boldsymbol{\sigma}_{(k)}$ take a vector of dimension $m + D$ as an input and return a vector of dimension $d - m$ as an output; there can be an arbitrary sequence of network layers and activation

functions in between input and output. Every mapping $\boldsymbol{z}_{(k)} = \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ for $k \in \{1, \dots, n\}$ is now defined as an affine transformation, such that

$$f_{\boldsymbol{\phi},(k),l}(\boldsymbol{z}_{(k-1)}) = \begin{cases} z_{(k-1),l} & , \quad l \in \{1, \dots, m\} \\ \mu_{(k),l}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y}) + e^{\sigma_{(k),l}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})} \cdot z_{(k-1),l} & , \quad l \in \{m+1, d\} \end{cases} \tag{46}$$

and

$$f^{-1}_{\boldsymbol{\phi},(k-1),l}(\boldsymbol{z}_{(k)}) = \begin{cases} z_{(k),l} & , \quad l \in \{1, \dots, m\} \\ [z_{(k),l} - \mu_{(k),l}(\boldsymbol{z}_{(k),1:m}, \boldsymbol{y})] \cdot e^{-\sigma_{(k),l}(\boldsymbol{z}_{(k),1:m}, \boldsymbol{y})} \cdot z_{(k),l} & , \quad l \in \{m+1, d\} \end{cases}, \tag{47}$$

where $f_{\boldsymbol{\phi},(k)} = (f_{\boldsymbol{\phi},(k),1}, \dots, f_{\boldsymbol{\phi},(k),d})^\top$ and $\boldsymbol{z}_{(k),1:m} = (z_{(k),1}, \dots, z_{(k),m})^\top$.

It is a convenient property of such affine functions $\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ for $k \in \{1, \dots, n\}$, that the Jacobians with respect to $\boldsymbol{z}_{(k-1)}$ are triangular, that is

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} = \begin{pmatrix} \mathbf{1}_m & \mathbf{0}_m \\ \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k),m+1:d}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1),m+1:d}} & \operatorname{diag}\left(e^{\sigma_{(k),m+1:d}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})}\right) \end{pmatrix}, \tag{48}$$

where $\mathbf{1}_m$ is a unity matrix and $\mathbf{0}_m$ is a zero matrix of dimension $m \times m$. The lower left quantity in the above matrix is again a Jacobian and takes the form

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k),m+1:d}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1),m+1:d}} = \begin{pmatrix} \frac{\partial f_{\boldsymbol{\phi},(k),m+1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),m+1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),m+1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),m+1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \end{pmatrix}; \tag{49}$$

the lower right quantity however is a diagonal matrix with the elements $e^{\sigma_{(k),l}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})}$ for $l \in \{m+1, d\}$ on the diagonal. Therefore, the Jacobian $\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)}) / \partial \boldsymbol{z}_{(k-1)}$ is triangular and hence its determinant is given by the product of the diagonal elements, that is

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right) = \prod_{j=m+1}^{d} e^{\sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})}. \tag{50}$$

Taking the logarithm of this expression results in a transformation of the product to a sum, such that

$$\log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right)\right] = \sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y}) \tag{51}$$

holds. Recall, that the complete normalizing flow is given as a concatenation of affine coupling functions (layers), such that

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{k=1}^{n} \det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right). \tag{52}$$

Taking the logarithm of this expression and inserting the previous result eq. (51), this leads to

$$\begin{aligned}\log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right] &= \sum_{k=1}^{n} \log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right)\right] \\ &= \sum_{k=1}^{n}\sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y}).\end{aligned} \tag{53}$$

3.4.3 Training process

Recalling the diffeomorphism $\boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})$ and the change of variable formula for the multivariate case, the probability density $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ can be written as found in eq. (40). Taking the natural logarithm of expression eq. (40) and inserting the result eq. (53) into it, one obtains

$$\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})\right] = \log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\right) + \left|\sum_{k=1}^{n}\sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})\right|. \tag{54}$$

Note, that the natural logarithm $\log(a)$ is strictly monotonic increasing for $a > 0$; furthermore, $-\infty < \log(a) \leq 0$ for $0 < a \leq 1$. Therefore, maximizing the so-called likelihood $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ is equivalent to maximizing $\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})\right]$, which is named the log-likelihood. Maximizing $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ corresponds to finding optimal network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ for the deep neural networks $\boldsymbol{\mu}_{(k)}$ and $\boldsymbol{\sigma}_{(k)}$ for $k \in \{1, \ldots, n\}$ constituting the flow function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$, such that the most suitable model parameters $\boldsymbol{x}$ given some context $\boldsymbol{y}$ are found. In the case of a stellar atmosphere inversion, $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ represents a probability distribution for the solutions $\boldsymbol{x}$ of an inversion $\boldsymbol{x} = \boldsymbol{M}^{-1}(\boldsymbol{y})$, where $\boldsymbol{M}$ is the atmosphere model and $\boldsymbol{y}$ are observations.

That is to say, that the normalizing flow, i.e. the neural networks constituting it, are trained by means of minimizing the negative log-likelihood $-\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})\right]$, which is equivalent to maximizing positive log-likelihood and hence also to maximizing the positive likelihood. Therefore, the loss function to minimize by any gradient-descent and backpropagation algorithm employed can be defined as

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}, \boldsymbol{y}) &= -\log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\right) - \log\left(\left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|\right) \\ &= -\log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\right) - \left|\sum_{k=1}^{n}\sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})\right|.\end{aligned} \tag{55}$$

This loss function is minimized with respect to the neural network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$.

3.4.4 Calculation of synthesized posterior samples used for training a normalizing flow

In order to be able to train a normalizing flow implementing a transformation

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}), \tag{56}$$

if can be a commodity to know the true posterior distribution $p(\boldsymbol{x}|\boldsymbol{y})$, in order to be able to compare it to the posterior distribution $p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ learned by the normalizing flow; this can be done by application of the Bayes theorem. As to calculate the true posterior distribution by means of the model $\boldsymbol{M}(\boldsymbol{x}) = \boldsymbol{y}$ linking together the so-called context $\boldsymbol{y}$ with the input $\boldsymbol{x}$ to the normalizing flow, one can proceed as follows.

First, $\boldsymbol{x}$ is assumed to follow a specific probability density $p(\boldsymbol{x})$, for example a uniform distribution. Next, one can sample $L \in \mathbb{N}$ datapoints $\{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(L)}\}$ from the prior $p(\boldsymbol{x})$. Using the model $\boldsymbol{M}$, for every $\boldsymbol{x}_{(k)}$ the corresponding value $\boldsymbol{y}_{(k)}$ can be calculated as $\boldsymbol{y}_{(k)} = \boldsymbol{M}(\boldsymbol{x}_{(k)})$, leading to the likelihood distribution $p(\boldsymbol{y}|\boldsymbol{x})$. Using the Bayes theorem and the law of total probability, one can calculate $p(\boldsymbol{x}_{(k)}|\boldsymbol{y}_{(k)})$ as

$$p(\boldsymbol{x}_{(k)}|\boldsymbol{y}_{(k)}) = \frac{p(\boldsymbol{y}_{(k)}|\boldsymbol{x}_{(k)})p(\boldsymbol{x}_{(k)})}{\sum_{j=1}^{L} p(\boldsymbol{y}_{(k)}|\boldsymbol{x}_{(j)})p(\boldsymbol{x}_{(j)})}, \tag{57}$$

leading to the probability distribution $p(\boldsymbol{x}|\boldsymbol{y})$ for $k \in \{1, \ldots, L\}$ and $L \to \infty$.

4 Proof of concept for normalizing flows

4.1 Moons

4.1.1 Conceptual formulation

In this experiment, a normalizing flow using affine coupling layers was tested on data $\boldsymbol{x} = (x_1, x_2)^\top \in \mathbb{R}^2$ given by a probability density function $p_{\boldsymbol{x}}(\boldsymbol{x})$ representing two moons. The goal was to map the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ representing the moons distribution to a standard normal distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, i.e. to find a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})} : \mathbb{R}^2 \to \mathbb{R}^2, \quad p_{\boldsymbol{x}}(\boldsymbol{x}) \sim \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}) \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^2. \tag{58}$$

To this end, a normalizing flow to implement the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$ was created as described in section 3.3.3. Herewith, $\hat{\mathbf{x}} \in \mathbb{R}^{d \times q}$ denotes the training data, where $d = 2$ and $q \in \mathbb{N}$. It is to be remembered, that the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ can be written as being conditional on the network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}})$. Hence, the notation $p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ is adopted to indicate, that for a trained normalizing flow, $\boldsymbol{x}$ can be written as $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{z})$ and that the density $p_{\boldsymbol{x}}(\boldsymbol{x})$ as sampled by application of the inverse normalizing flow to normally distributed samples $\boldsymbol{z}$ is therefore dependent upon the network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}})$.

4.1.2 Results

The subsequent results and figures were obtained[1] using a training sample size of 100000, a batch size of 512, a hidden layer size of 512, a coupling layer amount of 10, a learning rate of 0.001, a scheduling rate of 0.999 and 25 epochs to train the flow. The training

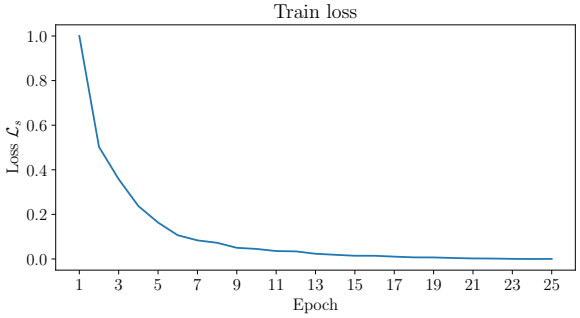


Figure 4: Visualization of the training process in terms of losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

process is visualized by fig. 4, where the loss $\mathcal{L}$ is scaled to $\mathcal{L}_s$ as

$$\mathcal{L}_s = \frac{\mathcal{L} - \min(\mathcal{L})}{\max(\mathcal{L} - \min(\mathcal{L}))}. \tag{59}$$

The performance after training can be seen as shown in fig. 5. It is evident, that the normalizing flow clearly has learned most of the defining properties of the probability distribution for $\boldsymbol{z}$. While the presented results are not perfect, it can safely be assumed that nearly perfect results would obtain if the epoch number for the training process would be chosen as going to infinity. Perfect hereby means, that the flow $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$ maps samples $\boldsymbol{x}$ from the distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$ to samples $\boldsymbol{z}$ from the standard normal distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ without any mismaps.

[1]The code which produced the presented results for the moons example is available at `https://github.com/danielzahnd/nf-moons-example`.

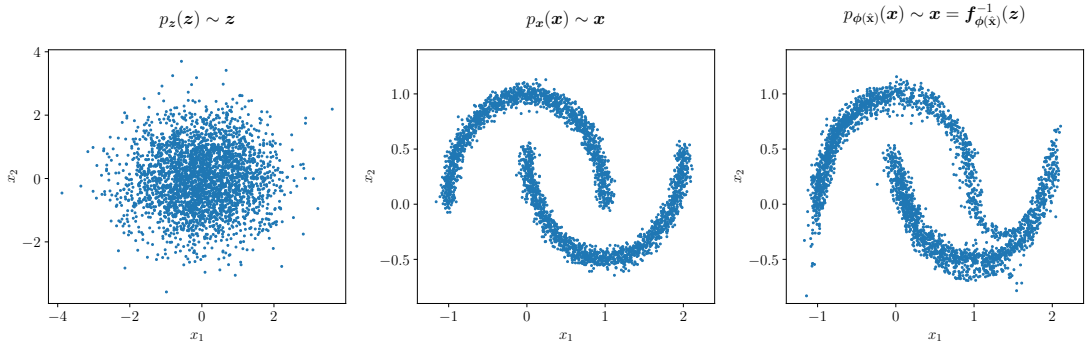


Figure 5: Results for a normalizing flow implementing the moons example.

4.2 Linear regression

4.2.1 Conceptual formulation

In this experiment, a normalizing flow using affine coupling layers was tested on data $\boldsymbol{x} = (x_1, x_2)^\top = (a, b)^\top \in \mathbb{R}^2$ and associated context $\boldsymbol{y} = (y_1, \ldots, y_{10})^\top \in \mathbb{R}^{10}$ given by probability density functions $p_{\boldsymbol{x}}(\boldsymbol{x})$ and $p_{\boldsymbol{y}}(\boldsymbol{y})$. Hereby, the context $\boldsymbol{y}$ is generated based on $\boldsymbol{x}$ by means of a model

$$\boldsymbol{M} : \mathbb{R}^2 \to \mathbb{R}^{10}, \quad \boldsymbol{x} \mapsto \boldsymbol{y} = \boldsymbol{M}(\boldsymbol{x}), \tag{60}$$

where the model $\boldsymbol{M}$ is defined as

$$y_j = M_j(\boldsymbol{x}) = x_1 u_j + x_2 = a u_j + b_j, \quad j \in \{1, \ldots, 10\} \tag{61}$$

with $\boldsymbol{u} = (u_1, \ldots, u_{10})^\top \in \mathbb{R}^{10}$ being a vector containing 10 equally spaced numbers between -5 and 5.

The goal was to map the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ representing the distributions of the slopes and intercepts of an affine function to a standard normal distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, i.e. to find a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^2 \to \mathbb{R}^2, \quad p_{\boldsymbol{x}}(\boldsymbol{x}) \sim \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}) \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^2. \tag{62}$$

To this end, a normalizing flow with context was created as described above and in section 3.4, in order to implement the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$. Herewith, $\hat{\mathbf{x}} \in \mathbb{R}^{d \times q}$ and $\hat{\mathbf{y}} \in \mathbb{R}^{D \times q}$ denote the training data, where $d = 2$, $D = 10$ and $q \in \mathbb{N}$. It is to be remembered, that the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ can be written as being conditional on $\boldsymbol{y}$, since the two are intertwined by $\boldsymbol{x} = \boldsymbol{M}^{-1}(\boldsymbol{y})$. Furthermore and because this conditional probability is to be learned by the normalizing flow $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$, the notation $p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ is adopted to indicate, that for a trained normalizing flow, $\boldsymbol{x}$ can be written as $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{z})$ and that the density $p_{\boldsymbol{x}}(\boldsymbol{x})$ as sampled by application of the inverse normalizing flow to normally distributed samples $\boldsymbol{z}$ is therefore dependent upon the network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$.

4.2.2 Results

The following results and figures were obtained[2] using a training sample size of 90000, a batch size of 512, a hidden layer size of 128, a coupling layer amount of 8, a learning rate of 0.001, a scheduling rate of 0.999 and 25 epochs to train the flow. In fig. 6, the training

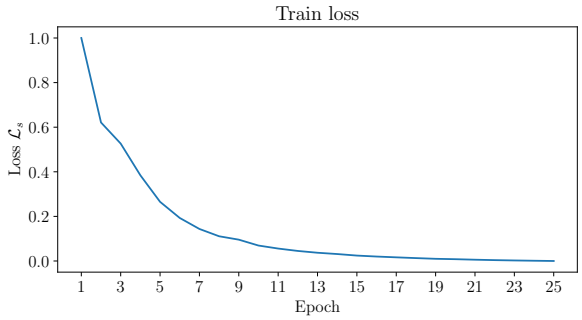


Figure 6: Visualization of the training process in terms of losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (59).

In fig. 7, the latent densities for the slope and intercept parameters can be seen; these density plots show how the training samples look like, if they are inserted into the normalizing flow function; that is to say, if $\boldsymbol{x}_{(j)}$, $j \in \{1, \dots, L\}$, $L \in \mathbb{N}$ denote training samples, then one obtains samples $\boldsymbol{z}_{(j)}$ by means of calculating

$$\boldsymbol{z}_{(j)} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}_{(j)}), \tag{63}$$

which should be standard normally distributed. As is readily seen, the mean and stan-

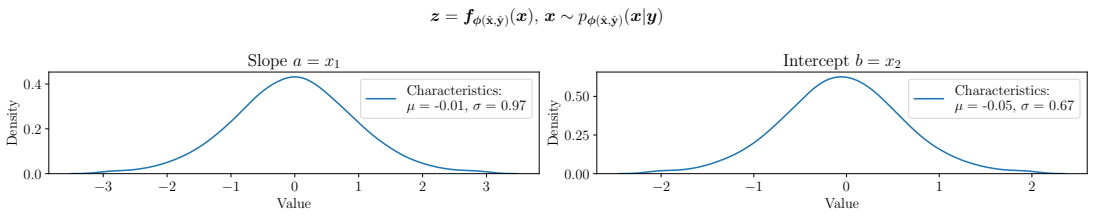


Figure 7: Calculated target densities.

dard deviations for both the slope and intercept parameters are close to 0 and 1, as they should be. The standard deviation for the latent density of the intercept parameter b is however somewhat off of 1, as it should be; this could likely resolved by iteratively optimizing the training settings.

[2]The code which produced the presented results for the linear regression example is available at `https://github.com/danielzahnd/nf-linear-regression-example`.

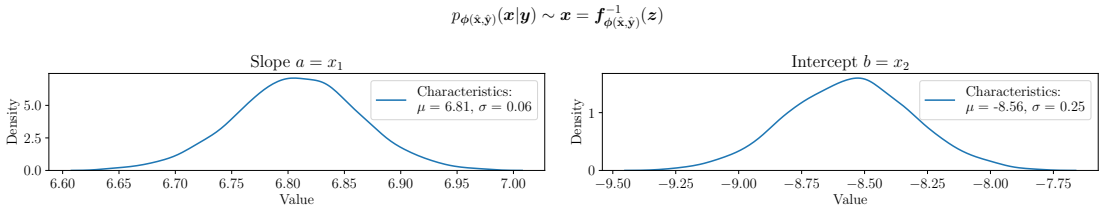
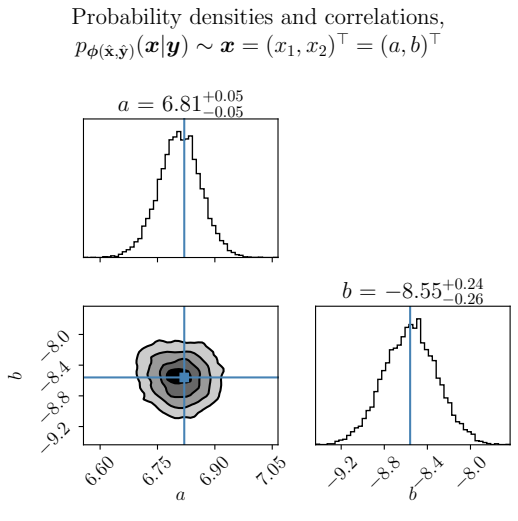


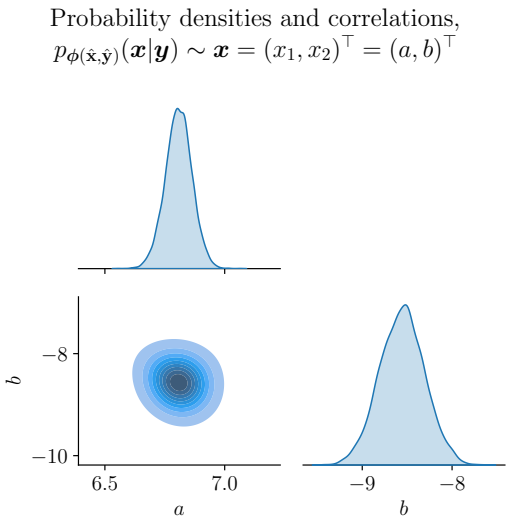
Figure 8: Calculated base densities.

Moreover, in fig. 8, the calculated densities for the slope and intercept parameters are depicted for a particular test observation $\boldsymbol{y}_{test}$ given to the normalizing flow as context. In particular, $\boldsymbol{y}_{test}$ consists of 10 values calculated by means of the above defined linear model eq. (60) based on test parameters $\boldsymbol{x}_{test} = (x_1, x_2)^\top = (a, b)^\top = (6.82, -8.56)$.

Furthermore, fig. 9 provides a view of the distributions aswell as of correlations for both the slope and intercept parameters. The corner and pairs plots show nicely, that



(a) Corner plot for the calculated densities of slope and intercept parameters using the `corner` package.



(b) Pairs plot for the calculated densities of slope and intercept parameters using the `seaborn` package.

Figure 9: Corner and pairs plots for the calculated densities of slope a and intercept b.

the normalizing flow indeed was successfully trained to perform linear regressions. First of all, the created Gaussian noise on observations used to train the normalizing flow is reflected in the Gaussian-shaped densities for the slope and intercept parameters. Secondly, the mean values and the associated standard deviations show, that the true values for a and b indeed lie within the range covered by one standard deviation around the mean of the considered parameter a or b.

4.3 Feature extraction

4.3.1 Conceptual formulation

In these experiments, a normalizing flow using affine coupling layers was tested on data $\boldsymbol{x} = (x_1, \ldots, x_10)^\top \in \mathbb{R}^{10}$ and associated context $\boldsymbol{y} = (y_1, \ldots, y_{240})^\top \in \mathbb{R}^{240}$ given by probability density functions $p_{\boldsymbol{x}}(\boldsymbol{x})$ and $p_{\boldsymbol{y}}(\boldsymbol{y})$. Hereby, the context $\boldsymbol{y}$ are spectra of the sun taken by IRIS satellite. From these spectra, altogether 10 features for each spectrum can be extracted by means of a program implementing a function

$$\boldsymbol{M} : \mathbb{R}^{240} \to \mathbb{R}^{10}, \quad \boldsymbol{y} \mapsto \boldsymbol{x} = \boldsymbol{M}(\boldsymbol{y}). \tag{64}$$

The goal was to map the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ representing the distributions of the extracted features to a standard normal distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, i.e. to find a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^{10} \to \mathbb{R}^{10}, \quad p_{\boldsymbol{x}}(\boldsymbol{x}) \sim \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}) \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^{10}. \tag{65}$$

To this end, a normalizing flow with context was created as described above and in section 3.4, in order to implement the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$. Here, $\hat{\mathbf{x}} \in \mathbb{R}^{d\times q}$ and $\hat{\mathbf{y}} \in \mathbb{R}^{D\times q}$ denote the training data, where $d = 10$, $D = 240$ and $q \in \mathbb{N}$. It is to be remembered, that the probability density $p_{\boldsymbol{x}}(\boldsymbol{x})$ can be written as being conditional on $\boldsymbol{y}$, since the two are intertwined by $\boldsymbol{x} = \boldsymbol{M}(\boldsymbol{y})$. Furthermore and because this conditional probability is to be learned by the normalizing flow $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$, the notation $p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})$ is adopted to indicate, that for a trained normalizing flow, $\boldsymbol{x}$ can be written as $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{z})$ and that the density $p_{\boldsymbol{x}}(\boldsymbol{x})$ as sampled by application of the inverse normalizing flow to normally distributed samples $\boldsymbol{z}$ is therefore dependent upon the network parameters $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$.

Given the similarity of the feature extraction task to performing inversions on a stellar atmosphere, a successful outcome of corresponding experiments is to be judged as a powerful proof of concept for the application of normalizing flows to inversions of the sun's atmosphere.

4.3.2 Results: Test 1

As a first test for the normalizing flow implemented as described above, the flow was trained on all available data; that is to say, that a train and test split of the data at hand was created, whereby the flow was subsequently trained on the train split of the data. The trained flow was then tested by means of propagating a spectrum from the test split through the trained normalizing flow. It was then tested, if the predicted feature distribution for the given test spectrum matched the exact feature values $\boldsymbol{x}$ as calculated by $\boldsymbol{x} = \boldsymbol{M}(\boldsymbol{y})$.

The following results and figures were obtained[3] using a training sample size of 39952, a batch size of 512, a hidden layer size of 64, a coupling layer amount of 6, a learning rate of 0.001, a scheduling rate of 0.999 and 60 epochs to train the flow.

[3]The code which produced the presented results for the first test of the feature extraction example is available at `https://github.com/danielzahnd/nf-feature-extraction-example-1`.

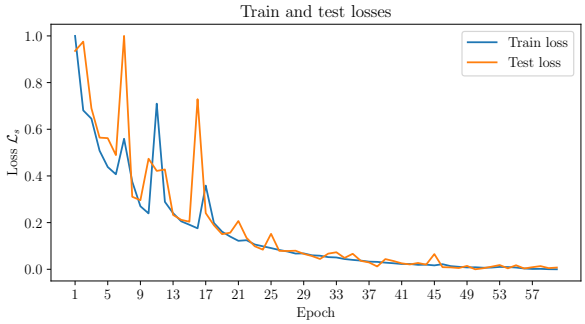


Figure 10: Visualization of the training process in terms of train and test losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

In fig. 10, the training process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (59).

In fig. 11, the latent densities for the feature parameters can be seen. As it is evident from the figure, the latent densities are close to 0 in terms of the mean value and close to 0 with respect to the standard deviation; so the latent densities can be considered close to standard normally distributed. Iteratively optimizing the training settings for the normalizing flow could lead to better results in terms more standard normally distributed latent densities for the features.

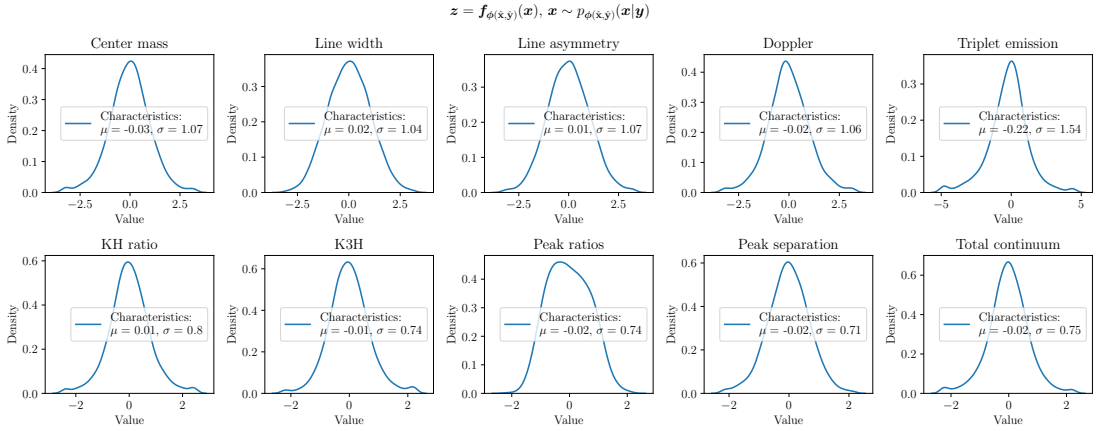


Figure 11: Calculated target densities.

Moreover, in fig. 12, the calculated densities and correlations of feature parameters are depicted for a particular test observation $\boldsymbol{y}_{test}$ given to the normalizing flow as context. In particular, $\boldsymbol{y}_{test}$ belongs to the test set of the data only used for testing; which is to say, that the normalizing flow has no knowledge of the test observation. The blue lines in the mentioned plot indicate the true values. As it is evident from said figure, the normalizing flow predicts feature distributions agreeing with the true values for the

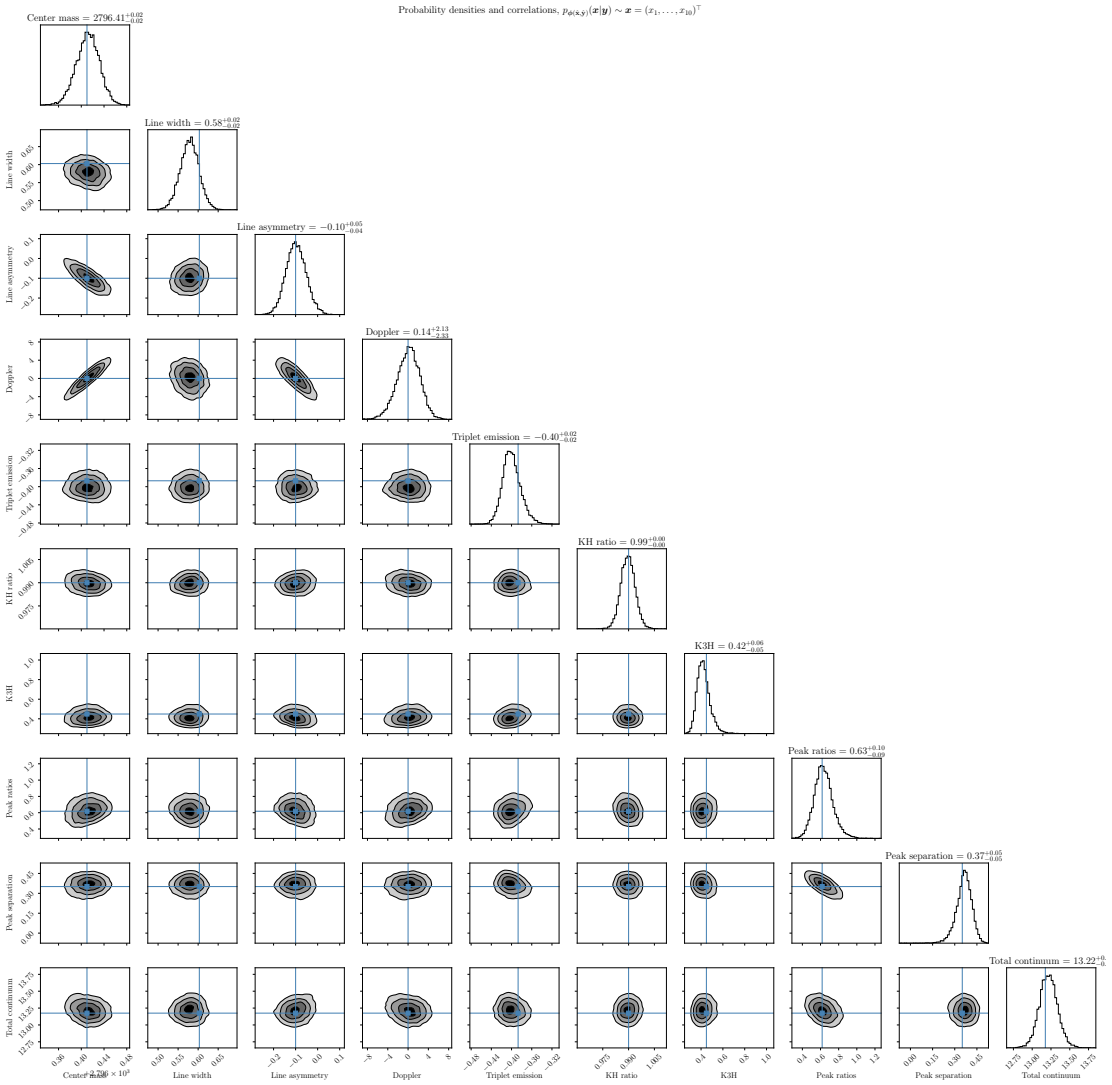


Figure 12: Corner plot for the calculated densities of feature parameters using the `corner` package.

features very well for most of the features. There only seem to be some inaccuracies with the line width and triplet emission features, but iteratively optimizing the training settings and the structure of the normalizing flow is likely to improve results even for these two mentioned features.

4.3.3 Results: Test 2

As a second test for the normalizing flow implemented as described above, the flow was trained on an artificially created dataset of spectra and corresponding features. This dataset was created by means of first calculating an average spectrum based on the

dataset used in the first test mentioned in section 4.3.2. To this average spectrum, random Gaussian noise was added, thus creating a dataset of Gaussian distributed spectra centered around the average spectrum. This was the dataset used to train the normalizing flow on. After training, the normalizing flow was tested on the average spectrum; this is to say, that the average spectrum was propagated through the normalizing flow, which allows for checking, whether the predicted feature distribution for the average spectrum is indeed Gaussian or not. Based on the Gaussian distributed spectra used to train the flow, a Gaussian distribution of the features densities as predicted by the normalizing flow should be expected.

The following results and figures were obtained[4] using a training sample size of 44671, a batch size of 512, a hidden layer size of 64, a coupling layer amount of 6, a learning rate of 0.001, a scheduling rate of 0.999 and 60 epochs to train the flow.

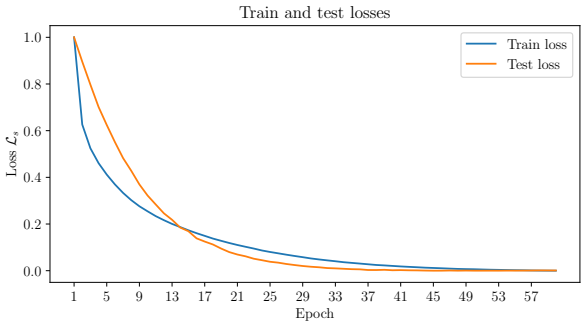


Figure 13: Visualization of the training process in terms of train and test losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

In fig. 13, the training process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (59).

In fig. 14, the latent densities for the feature parameters can be seen. As it is apparent from the figure, the latent densities are close to being standard normally distributed. Iteratively optimizing the training settings for the normalizing flow could lead to better results in terms more standard normally distributed latent densities for the features.

Moreover, in fig. 15, the calculated densities and correlations of feature parameters are depicted for a particular test observation $\boldsymbol{y}_{test}$ given to the normalizing flow as context; in this case, $\boldsymbol{y}_{test}$ is identical to the average spectrum. The blue lines in the mentioned plot indicate the true values. As it is evident from said figure, the normalizing flow predicts feature distributions agreeing with the true values for the features very well for most of the features. There only seem to be some inaccuracies with the line width and triplet emission features, but iteratively optimizing the training settings and the structure of the normalizing flow is likely to improve results even for these two mentioned features. As for the total continuum feature, there seem to be two peaks in

[4]The code which produced the presented results for the first test of the feature extraction example is available at `https://github.com/danielzahnd/nf-feature-extraction-example-2`.

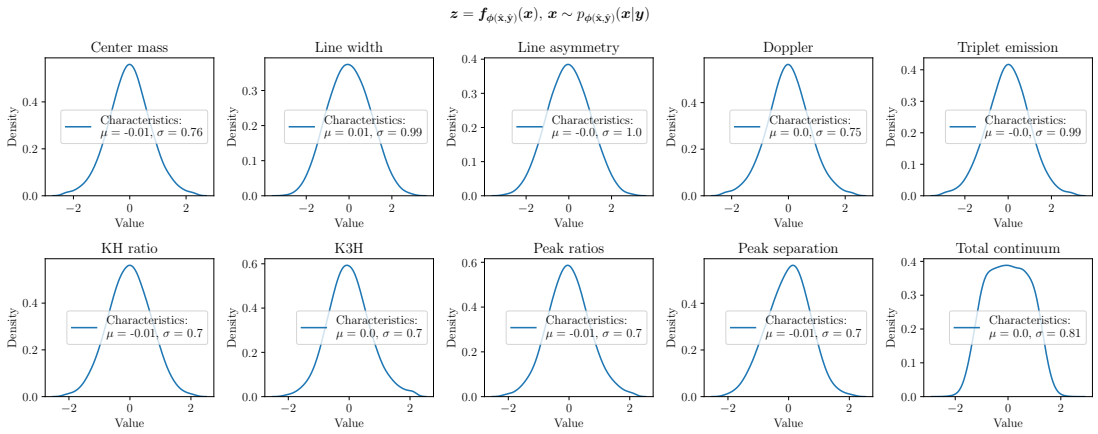


Figure 14: Calculated target densities.

the density distribution, the true values for the features corresponding to the average spectrum situated nicely between those peaks, as it would be expected.

4.3.4 Results: Test 3

As a third test for the normalizing flow implemented as described above, the flow was trained on an a dataset containing spectra and corresponding features of two very different categories. This dataset was created by means of first clustering the available data into several clustering algorithm[5]. After that, the two most different clusters of spectra and corresponding features were combined to one dataset, which was subsequently split into a train and test dataset. The normalizing flow was subsequently trained on the train split of the data. The trained flow was then tested by means of propagating the centroid spectra for both clusters of spectra contained in the train dataset through the trained normalizing flow. I was then checked, whether the predicted feature distributions for these two centroid spectra matched the exact feature values $\boldsymbol{x}$ as calculated by $\boldsymbol{x} = \boldsymbol{M}(\boldsymbol{y})$.

The following results and figures were obtained[6] using a training sample size of 4018, a batch size of 512, a hidden layer size of 256, a coupling layer amount of 6, a learning rate of 0.001, a scheduling rate of 0.999 and 35 epochs to train the flow.

In fig. 16, the training process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (59).

In fig. 17, the latent densities for the feature parameters can be seen. As it is apparent from the figure, the latent densities are close to being standard normally distributed. Iteratively optimizing the training settings for the normalizing flow would likely lead to better results in terms more standard normally distributed latent densities for the

[5]In this case, the `KMeans` function of the `sklearn.cluster` package was used.

[6]The code which produced the presented results for the first test of the feature extraction example is available at `https://github.com/danielzahnd/nf-feature-extraction-example-3`.

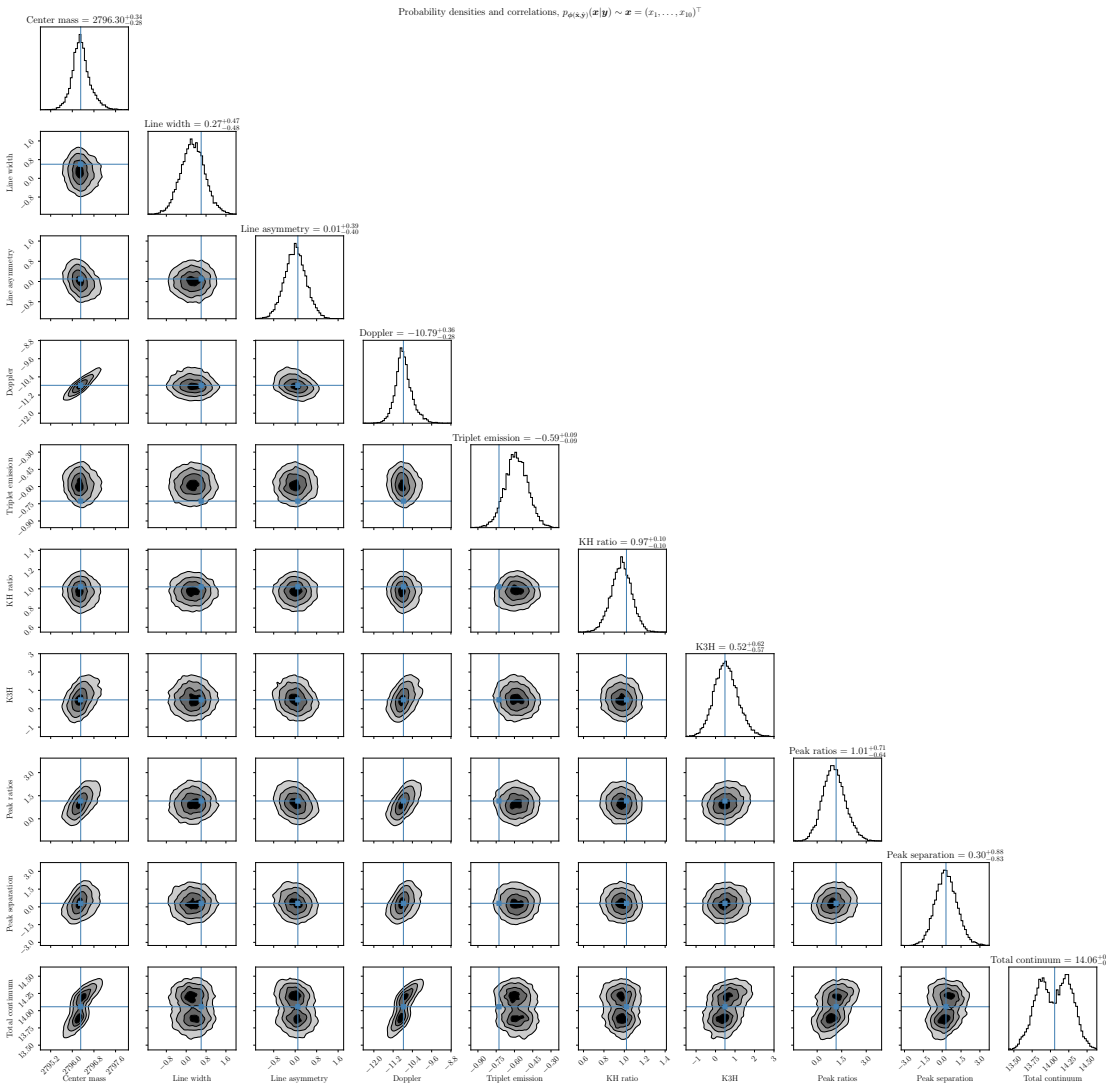


Figure 15: Corner plot for the calculated densities of feature parameters using the `corner` package.

features.

Furthermore, in fig. 18, the calculated densities and correlations of feature parameters are depicted for two test observations given to the normalizing flow as context; in this case, the centroid spectra of both categories of spectra used to train the normalizing flow on were used as test observations. The blue lines in the mentioned plot indicate the true values for the feature distributions, whereas orange/grey differentiate, whether a feature distribution corresponds to either one or the other centroid spectrum. As it is evident from said figure, the normalizing flow predicts the feature distributions for both centroid spectra to agree with the true values well for only about half of the features. This could

Figure 16: Visualization of the training process in terms of train and test losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

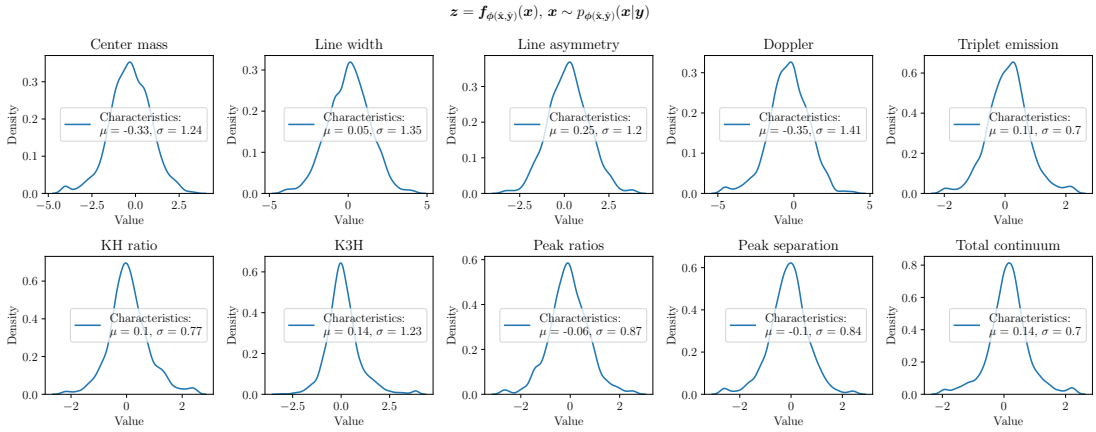


Figure 17: Calculated target densities.

however be resolved by means of using a larger dataset to train the normalizing flow on or by iteratively optimizing the structure of the flow and the training settings of it. In general however, the normalizing flow seems to be able to clearly differentiate, if an input spectrum belongs to one or the other category of spectra.

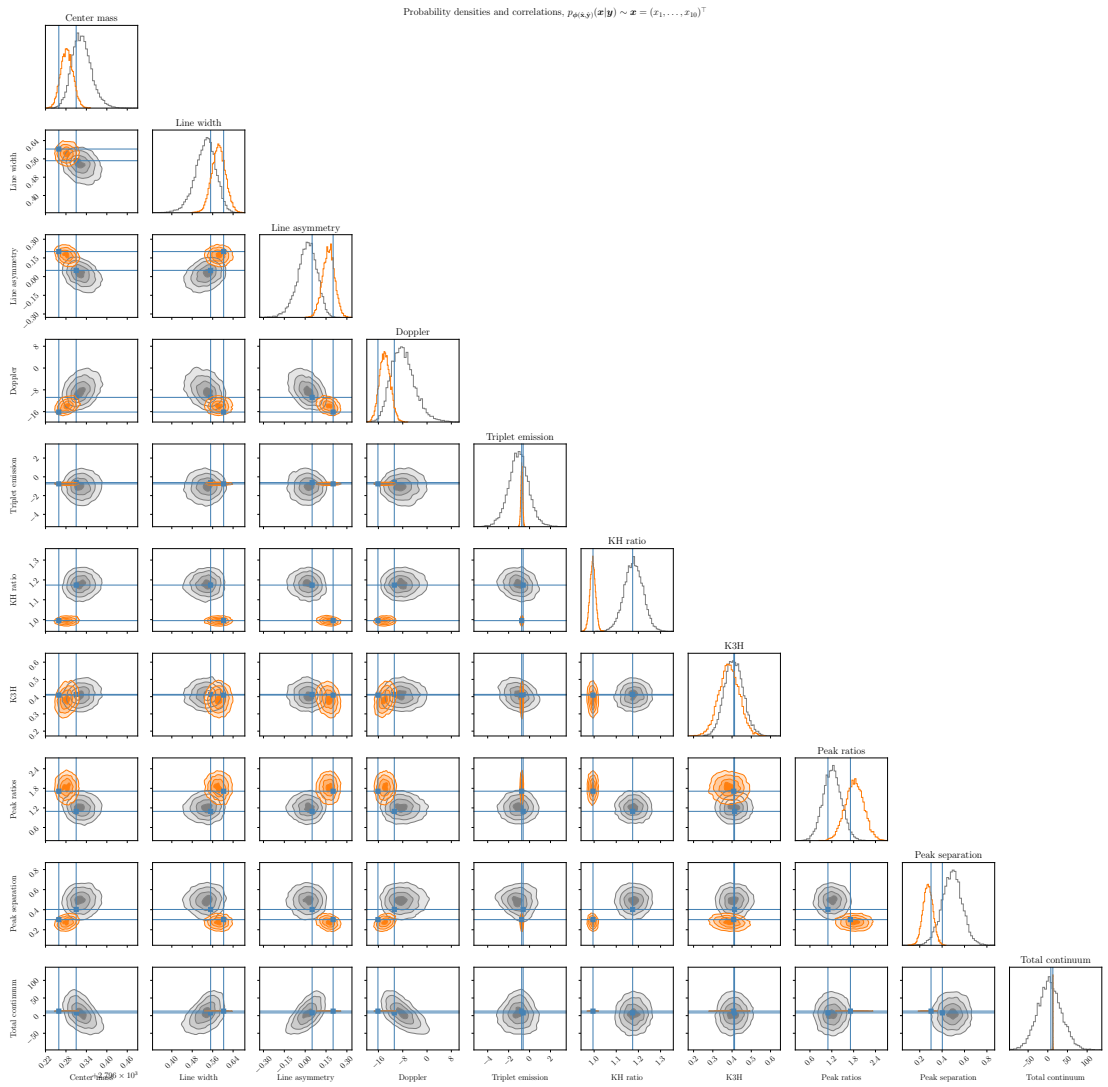


Figure 18: Corner plot for the calculated densities of feature parameters using the `corner` package.

5 Solar physics

5.1 Radiative transfer

Text.

5.2 Stellar models

Text.

5.3 Stellar inversions

Consider some observation $\boldsymbol{y}$ of a stellar atmosphere. Assume, that there is a model $\boldsymbol{M}$ with associated parameters $\boldsymbol{x}$, such that the model is given by the relationship $\boldsymbol{M} = \boldsymbol{M}(\boldsymbol{x}) = \boldsymbol{y}$.

So if one knows the parameter values $\boldsymbol{x}_0$ of the stellar atmosphere model $\boldsymbol{M}$, then observations $\boldsymbol{y}_0$ can be generated based on the parameter values $\boldsymbol{x}_0$ by means of the relation $\boldsymbol{y}_0 = \boldsymbol{M}(\boldsymbol{x}_0)$; this is called the forward model. However, if one only has observations $\boldsymbol{y}_0$ and the stellar model $\boldsymbol{M}$, one would like to find out the model parameters θ_0 associated to these observations; this is called the backward model and can be achieved by means of applying the relation $\boldsymbol{x}_0 = \boldsymbol{M}^{-1}(\boldsymbol{y}_0)$. As one can see, this requires a so-called inversion of the atmosphere model $\boldsymbol{M}$, which can lead to degeneracy in the model parameters $\boldsymbol{x}_0$, meaning that more than one configuration of $\boldsymbol{x}_0$ can lead to the same observations $\boldsymbol{y}_0$. A normalizing flow helps to disentangle this degeneracy by means of introducing a probability density function for each model parameter, such that the more likely parameter configurations leading to some observations $\boldsymbol{y}_0$ can be distinguished from less likely parameter configurations leading to the same observations.

5.4 Solving a stellar inversion problem

Consider a stellar model $\boldsymbol{M}(\boldsymbol{x})$. Given observations $\boldsymbol{y}_0$, one would like to infer the stellar parameters $\boldsymbol{x}_0$ associated to these observations by applying the relation

$$\boldsymbol{x}_0 = \boldsymbol{M}^{-1}(\boldsymbol{y}_0). \tag{66}$$

Commonly, such inversion problems are carried out using a gradient search minimization algorithm which basically functions as follows. First, a function which should be minimized is defined; in our case

$$\boldsymbol{F}(\boldsymbol{x}) \doteq \boldsymbol{M}^{-1}(\boldsymbol{y}_0) - \boldsymbol{x}. \tag{67}$$

By computing the gradient of $\boldsymbol{F}(\boldsymbol{x}) = (F_1(\boldsymbol{x}), \ldots, F_n\boldsymbol{x})^\top$, $n \in \mathbb{N}$ with respect to the parameters $\boldsymbol{x}$, that is for every $F_i \in \boldsymbol{F} = (F_1, \ldots, F_n)^\top$ the vector $\boldsymbol{\nabla}_{\boldsymbol{x}} F_i(\boldsymbol{x})$ is calculated, one iteratively approaches a minimum of $\|\boldsymbol{F}\|$, if $\boldsymbol{x}$ is iteratively changed in the direction of negative gradient of $\boldsymbol{F}(\boldsymbol{x})$.

But these types of algorithms are computationally costly, if one wants to infer some sort of distribution of model parameters based on more than just one observation; one gradient search only converges to one possible solution of the inversion problem. Here the normalizing flows technique comes into play, which provides a computationally more efficient way to do inversions and furthermore allows for the possibility to construct a probability density of possible solutions to the inversion problem.

5.5 Milne-Eddington atmosphere

Text.

5.6 Stokes parameters in stellar models

Text.

6 Structure of thesis

Suggested structure:

(1) Introduction (general introduction to stellar models, inversions and normalizing flows)

(2) Solar physics

   (a) Stellar models
   (b) Inversions

(3) Machine learning

   (a) Overview
   (b) Basics of deep learning (perceptron, multilayer perceptron, convolutional layers, etc.)
   (c) Deep generative models (GAN's, VAE's and Normalizing flows)
   (d) Normalizing flows

(4) Applications

   (a) Proof of concept
   (b) Findings on data of the SST (swedish solar telescope)

References

[Kobyzev et al., 2021] Kobyzev, I., Prince, S. J. D., and Brubaker, M. A. (2021). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill international editions. McGraw-Hill, New York, NY, international ed. edition.