University of Bern

Master thesis

Physics

Investigation of the solar atmosphere using machine learning techniques

Author:
Daniel Zahnd

Supervisor:
Prof. Dr. Lucia Kleint

September 18, 2023

Abstract

English abstract.

Contents

Chapter 1

Introduction

Text.

Chapter 2

Artificial intelligence

The field of artificial intelligence (AI) is relatively recent field of research dedicated to mimic the process of learning information. Hereby, inspiration for first principles of a sub-field of artificial, the so-called deep learning, comes from the brain structure of mammals.

As AI tries to mimic the process of learning, it remains to be answered what learning actually is. But it In order to define how a learning process is constituted, another quantity has to be defined first, 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, which will be defined in a later stage, 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.

2.1 Information theory

The basic intuition behind information theory is the consideration, that the occurrence of an unlikely event should convey more information than the occurrence of a likely event. The quantification of information in the field of information theory therefore relies on three key 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.

2.1.1 Self-information

The above mentioned requirements are satisfied by the so-called self-information $\mathcal{I}[p(x)]$ defined as

$$\mathcal{I}[p(x)] = -\ln\left[p(x)\right], \tag{2.1}$$

with X being a random variable and $p(x)$ the probability density function thereof.

By this definition, information $\mathcal{I}[p(x = U)]$ of an unlikely event $x = U$ will be higher than information $\mathcal{I}[p(x = L)]$ of a likely event $x = L$; this can be demonstrated by assessing the self-information contents of both events. Because U is said to be an unlikely event, $p(U) \approx 0$ will hold and therefore $\mathcal{I}[p(x = U)] = -\ln[p(U)] = \ln[1/p(U)] \gg 1$. Since L is a likely event, the probability $p(L)$ of L occurring is estimated by $p(L) \approx 1$ and the self-information $\mathcal{I}[p(x = L)]$ will therefore be very low, because $\mathcal{I}[p(x = L)] = -\ln[p(L)] = \ln[1/p(L)] \approx 0$ holds. Moreover, information about two independent events $x = A$ and $x = B$ is additive, as the calculation

$$\begin{aligned} \mathcal{I}[p(A \cap B)] &= -\ln[p(A \cap B)] = -\ln[p(A)p(B)] \\ &= -(\ln[p(A)] + \ln[p(B)]) = \mathcal{I}[p(A)] + \mathcal{I}[p(B)] \end{aligned} \tag{2.2}$$

shows it to be the case.

2.1.2 Information entropy

Information entropy $H_p[p(x)]$ 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

$$H_p[p(x)] = E_{x\sim p}(\mathcal{I}[p(x)]) = -E_{x\sim p}\left(\ln[p(x)]\right). \tag{2.3}$$

Information entropy defined in this way is a measure of the amount of uncertainty in a probability distribution. 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 mainly 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.

The field of artificial intelligence and especially its sub-field machine learning are concerned with learning the available self-information from the data. That is to say, the information entropy initially present within the machine learning entity is iteratively reduced by means of training it on a dataset, thus resulting in learned information on the part of the trained artificial intelligence.

2.2 Overview of artificial intelligence

According to [Amini, 2023, p.7], the field of artificial intelligence can be broadly described as any computational technique, that enables a computer to engange with information in a reasonable way, such as to mimic how a human would process the available data or information. Machine learning is more specialized sub-field of artificial intelligence, which is concerned with any routine, that gives a computer the ability to learn a task without being explicitly programmed for that particular task. Deep learning, which is ultimately about reflecting and rebuilding a simple representation the human brain working principle, is an even more specialized sub-field of artificial intelligence, which is itself a sub-field of machine learning.

The foundational tool of deep learning is the so-called neural network, also known as deep neural network. The basic building block of a neural network is the single layer perceptron (SLP). By combining several single layer perceptrons to a multi layer perceptron (MLP), a deep neural network can be constructed. Given an input vector $\boldsymbol{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$ and an output vector $\bar{\boldsymbol{x}} = (\bar{x}_1, \ldots, \bar{x}_w)^\top \in \mathbb{R}^w$ with $d, w \in \mathbb{N}$, the working principle of single layer perceptron is shown in fig. 2.1, similarly for a multi layer preceptron in fig. 2.2.

2.3 Neural networks

2.3.1 Single layer perceptron

In fig. 2.1, a single layer perceptron is visualized. The quantity $\boldsymbol{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$ with $d \in \mathbb{N}$ is an arbitrary input vector. The single layer perceptron generates a single output $\bar{x}$. First, all input vector

components x_i are multiplied with initially arbitrary weights $w_i \in \mathbb{R}$ for all $i \in \{1, \ldots, d\}$ and subsequently summed up to some sum $\tilde{s} = \sum_{i=1}^{d} x_i w_i$; then, an arbitrary bias value $b \in \mathbb{R}$ is added to the sum $\tilde{s}$ such as to arrive at a new sum $s = b + \tilde{s}$. Finally, an output scalar $\bar{x} = a(s)$ is generated by means of pushing the beforehand attained sum s through a so-called activation function $a : \mathbb{R} \to \mathbb{R}, s \mapsto a(x)$, which is nothing more than some arbitrary nonlinear function, which introduces nonlinearities to the otherwise linear operations; this enables the perceptron to learn nonlinear correlations.

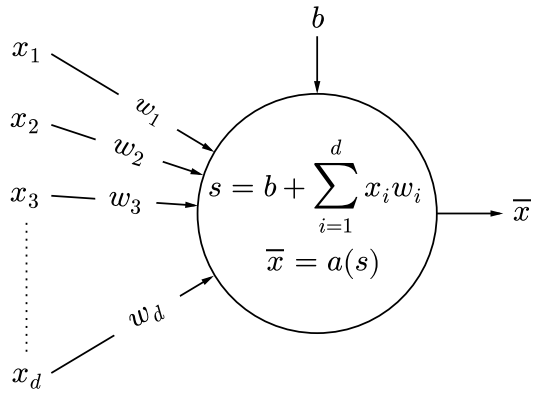


Figure 2.1: Working principle of a single layer perceptron; hereby, a represents a nonlinear activation function $a : \mathbb{R} \to \mathbb{R}$, $s \mapsto a(s)$.

2.3.2 Multi layer perceptron

A multi layer perceptron is the simplest form of a neural network. In fig. 2.2, such a multi layer perceptron is visualized. Again, $\boldsymbol{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$ with $d \in \mathbb{N}$ is an arbitrary input vector. Compared to the single layer perceptron, the multi layer perceptron however generates a vectorial output $\bar{\boldsymbol{x}} = (\bar{x}_1, \ldots, \bar{x}_w)^\top \in \mathbb{R}^w$ with $w \in \mathbb{N}$. Now, all input components x_i are connected to more than one perceptron; thus all perceptrons directly connected to the input components compose a so-called network layer, the first layer. There can be one or more network layers, generally there are $n \in \mathbb{N}$ layers in a neural network. At each perceptron of a certain, fixed network layer, a single output is generated as with the single layer perceptron. All of these single outputs together compose a new vector, which serves as the input vector for the next network layer composed of perceptrons. This general working principle is better explained visually, consider therefore fig. 2.2.

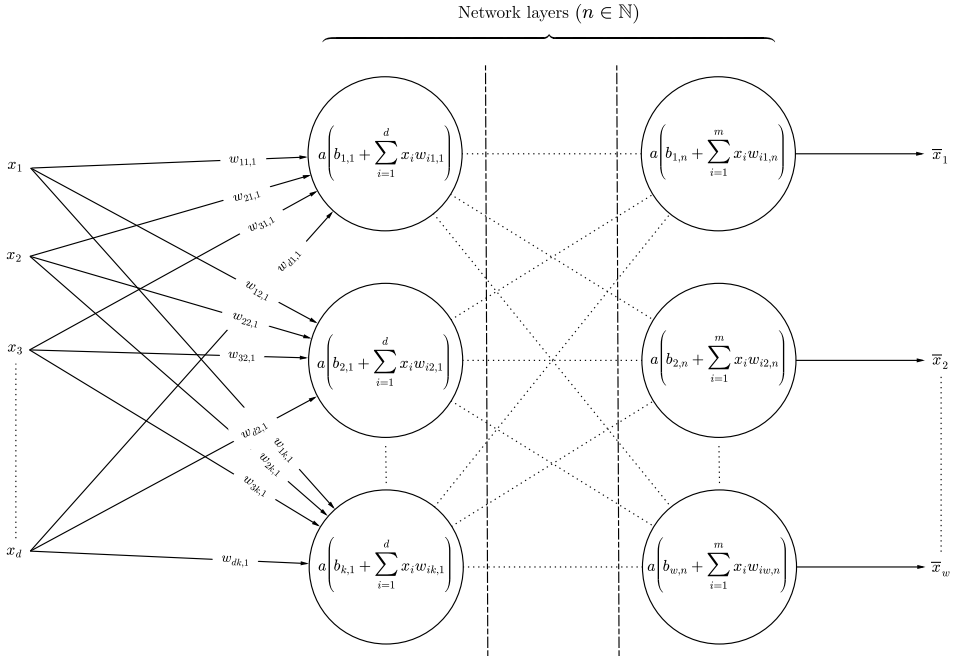


Figure 2.2: Working principle of a multi layer perceptron; hereby, a represents a nonlinear activation function $a : \mathbb{R} \to \mathbb{R}$, $s \mapsto a(s)$.

2.3.3 Training process for neural networks

In order for a neural network to learn anything, it has to be conditioned by data, whose interpretation is known. This process is generally called a training process and requires a mathematical framework that allows for optimization. During the training process of a neural network, one would like to achieve the best possible performance of the neural network in interpreting the training data correctly, that is to say, that the input data $\boldsymbol{x}$ correctly maps to the known output data $\bar{\boldsymbol{x}}$.

The mathematical framework for such an optimization is the so-called gradient descent routine. **Elaborate on gradient descent, write out formulae and supply a nice picture.**

2.4 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.

Chapter 3

Solar physics

All stars, and herewith also the sun, are sources of gigantic amounts of energy. This energy is set free by means of atomic fusion processes happening within the star and is conveyed to an observer through electromagnetic or particle radiation. In order to infer properties of radiation source, e.g. a star, the properties of the emitted radiation need to be observed and analyzed using the theoretical framework of physics.

In the case of solar atmosphere research, there are three main tools for investigation available to the physicist; first, the theory of radiative transfer, which is concerned with the propagation of electromagnetic radiation in some medium. Second, there is spectroscopy, which is about how electromagnetic radiation interacts with matter as a function of wavelength. Third, polarimetry is necessary, which addresses how and why electromagnetic radiation is polarized as it is.

3.1 Radiative transfer

Because electromagnetic radiation is the dominant radiation source in solar atmosphere research, radiative transfer is integral to investigating the solar atmosphere.

3.1.1 Quantities in radiative transfer

In the theory of radiative transfer, the definition of several measurable quantities is necessary. If some quantity is a called a spectral quantity, this means that the quantity is measured at a certain wavelength λ or frequency ν; spectral quantity will be denoted by a subscript λ or ν.

The radiant energy E with units $[E] = \mathrm{J}$ is defined as the energy

radiated away from some source. The radiant flux Φ with units $[\Phi] = \mathrm{Js}{-}1 = \mathrm{W}$ however is defined as the radiant energy per unit time and is equal to the spectral flux integrated over all wavelengths. Furthermore, the spectral flux Φ_λ having units $[\Phi_\lambda] = \mathrm{Wm}^{-1}$ is the radiant flux per unit wavelength. Additionally, the radiance I with units $[I] = \mathrm{Wsr}^{-1}\mathrm{m}^{-2}$ is the radiant flux per unit solid angle and per unit projected area; similarly the spectral radiance I_λ with units $\mathrm{Wsr}^{-1}\mathrm{m}^{-3}$ is the radiance per unit wavelength. Finally, the irradiance F with units $[F] = \mathrm{Wm}^{-2}$ is given by the radiant flux per unit projected area; it is also known as the flux density and is equal to the radiance integrated over all solid angles of a hemisphere. Associated to the radiance, there is also the spectral irradiance F_λ, with is nothing but the radiance per unit wavelength and therefore with units $[F_\lambda] = \mathrm{Wm}^{-3}$.

3.1.2 Radiative transfer equation

The radiative transfer equation is usually given in terms of spectral irradiance I_λ. The basic two forms of the radiative transfer equation shall now be briefly derived, following the material in [Rutten, 2015, pp.27-40].

Consider as a first step fig. 3.1. A beam of spectral irradiance I_λ passes through a slab of matter with thickness $\mathrm{d}z$, along a path s with length $\mathrm{d}s$. The question now is, how the spectral irradiance I_λ after the slab can be calculated. In order to quantify the spectral irradiance after passing through the slab of matter, two contributions due to electromagnetic interaction between radiation and matter within the slab have to be considered, namely extinction and emission. Extinction is loss of radiative energy along a path s due to scattering or absorption and it is accounted for by the extinction coefficient α_λ with units $[\alpha_\lambda] = \mathrm{m}^{-1}$. Emission is gain of radiative energy along a path s due to spontaneous emission of photons caused by decay of excited states in molecules or atoms within the slab of matter; it is quantified by the emission coefficient j_λ having units $[j_\lambda] = \mathrm{Wsr}^{-1}\mathrm{m}^{-4}$.

The difference $\mathrm{d}I_\lambda$ in spectral irradiance along the path of propagation must be the sum of the extinction and emission contributions. Considering the units of the extinction and emission coefficients, these contributions can be written as $-\alpha_\lambda I_\lambda \,\mathrm{d}s$ for the loss due to extinction and as $j_\lambda \,\mathrm{d}s$ for the gain due to emission. Herewith, one arrives at

$$\mathrm{d}I_\lambda = j_\lambda \,\mathrm{d}s - \alpha_\lambda I_\lambda \,\mathrm{d}s \quad \Leftrightarrow \quad \frac{\mathrm{d}I_\lambda}{\mathrm{d}s} = j_\lambda - \alpha_\lambda I_\lambda. \tag{3.1}$$

Defining the optical pathlength τ_λ by means of $\mathrm{d}\tau_\lambda \doteq \alpha_\lambda \,\mathrm{d}s$ and further-

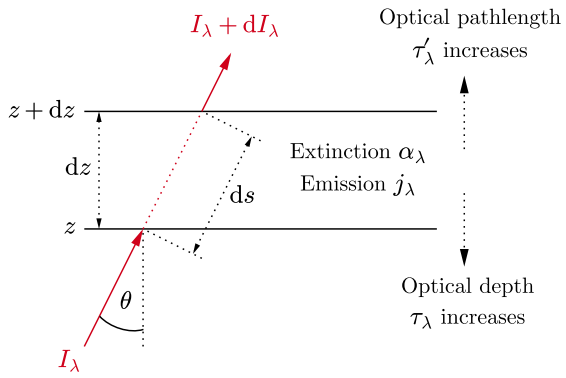


Figure 3.1: Illustration for the derivation of the radiative transfer equation. $[I_\lambda] = \mathrm{W\,sr^{-1}\,m^{-3}}$ denotes the spectral radiance measured along a ray of inclination θ with respect to a horizontal plane in the coordinate system under consideration.

more the so-called source function S_λ as $S_\lambda \doteq \frac{j_\lambda}{\alpha_\lambda}$, this equation can also be written as

$$\frac{\mathrm{d}I_\lambda(\tau_\lambda)}{\mathrm{d}\tau_\lambda} = S_\lambda(\tau_\lambda) - I_\lambda(\tau_\lambda), \tag{3.2}$$

which is called the transport equation in differential form. Defining the optical depth τ'_λ as $\mathrm{d}\tau'_\lambda \doteq -\alpha_\lambda\,\mathrm{d}z = -\alpha_\lambda \cos(\theta)\,\mathrm{d}s = \cos(\theta)\,\mathrm{d}\tau_\lambda$, the transport equation can also be written in terms of optical depth rather than optical pathlength, namely

$$\cos(\theta)\frac{\mathrm{d}I_\lambda(\tau'_\lambda)}{\mathrm{d}\tau'_\lambda} = I_\lambda(\tau'_\lambda) - S_\lambda(\tau'_\lambda). \tag{3.3}$$

The change in sign is due to the fact, that the optical depth τ'_λ is measured from the point of view of an observer looking against the ray propagation direction, while the optical pathlength τ_λ measures the distance of propagation in the direction of the ray. Both of these formulations for the transfer equation can also be written in an integral form, as it is demonstrated as follows. First, consider thoroughly fig. 3.2.

In order to derive integral forms of the transfer equation with respect to the optical pathlength τ_λ, the derivative

$$\frac{\mathrm{d}}{\mathrm{d}\tau_\lambda}\left[e^{\tau_\lambda} I_\lambda(\tau_\lambda)\right] = e^{\tau_\lambda}\left[I_\lambda(\tau_\lambda) + \frac{\mathrm{d}I_\lambda(\tau_\lambda)}{\mathrm{d}\tau_\lambda}\right] = e^{\tau_\lambda} S_\lambda(\tau_\lambda) \tag{3.4}$$

is first calculated. Finding the outward spectral radiance $I^+_\lambda(\tau_\lambda)$ at a given optical pathlength τ_λ requires integration of eq. (3.4) from 0 to τ_λ,

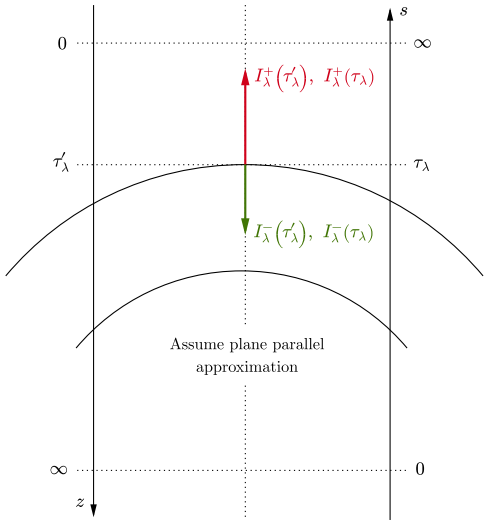


Figure 3.2: Illustration of coordinate systems for optical pathlength τ_λ parametrized by s and optical depth τ'_λ parametrized by z. Note, that the optical pathlength axis does not necessarily need to be parallel to the optical depth axis, the factor $\cos(\theta)$ corrects for the case where these axes are not parallel. The curved lines represent layers of a stellar atmosphere, which are locally assumed as plane parallel.

namely

$$\int_0^{\tau_\lambda} \frac{\mathrm{d}}{\mathrm{d}t}\left[e^t I_\lambda^+(t)\right]\,\mathrm{d}t = e^{\tau_\lambda} I_\lambda^+(\tau_\lambda) - I_\lambda^+(0) = \int_0^{\tau_\lambda} e^t S_\lambda(t)\,\mathrm{d}t. \tag{3.5}$$

From this expression it follows by multiplication of the above equation with $e^{-\tau_\lambda}$ and rearrangement of terms, that the outward spectral radiance at a certain optical pathlength τ_λ from the origin of the coordinate system under consideration is given by

$$I_\lambda^+(\tau_\lambda) = I_\lambda^+(0)e^{-\tau_\lambda} + \int_0^{\tau_\lambda} e^{t-\tau_\lambda} S_\lambda(t)\,\mathrm{d}t, \tag{3.6}$$

which is what is commonly called the formal solution of the transfer equation with respect to optical pathlength. If one wants to calculate the inward spectral radiance $I_\lambda^-(\tau_\lambda)$ however, integration of eq. (3.4) is required to be carried from ∞ to τ_λ, such that

$$-\int_{\tau_\lambda}^{\infty} \frac{\mathrm{d}}{\mathrm{d}t}\left[e^t I_\lambda^-(t)\right]\,\mathrm{d}t = e^{\tau_\lambda} I_\lambda^-(\tau_\lambda) = -\int_{\tau_\lambda}^{\infty} e^t S_\lambda(t)\,\mathrm{d}t \tag{3.7}$$

results. Multiplying this equation by $e^{-\tau_\lambda}$ results in the inward spectral

radiance at a given optical pathlength τ_λ is calculable as

$$I_\lambda^-(\tau_\lambda) = -\int_{\tau_\lambda}^{\infty} e^{t-\tau_\lambda} S_\lambda(t)\,\mathrm{d}t. \tag{3.8}$$

For both formulations of $I_\lambda^+(\tau_\lambda)$ and $I_\lambda^-(\tau_\lambda)$ the boundary conditions

$$I_\lambda^-(\infty) = 0, \quad e^t S_\lambda(t) \xrightarrow{t\to\infty} 0 \tag{3.9}$$

were used, which mean that at the position $\tau_\lambda = \infty$ there is no radiation directed towards the source moreover that the source function $S_\lambda(t)$ decreases faster with increasing optical pathlength than e^{-t}.

The integral representations of the transfer equation with respect to the optical depth τ'_λ can also be obtained in a similar way; for this purpose, the derivative

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\tau'_\lambda}\left[\cos(\theta)e^{-\frac{\tau'_\lambda}{\cos(\theta)}} I_\lambda(\tau'_\lambda)\right] &= e^{-\frac{\tau'_\lambda}{\cos(\theta)}}\left[\cos(\theta)\frac{\mathrm{d}I_\lambda(\tau'_\lambda)}{\mathrm{d}\tau'_\lambda} - I_\lambda(\tau'_\lambda)\right] \\ &= -e^{-\frac{\tau'_\lambda}{\cos(\theta)}} S_\lambda(\tau'_\lambda)\end{aligned} \tag{3.10}$$

is needed. Obtaining an expression for the outward spectral radiance $I_\lambda^+(\tau'_\lambda)$ at a given optical depth τ'_λ requires integration of eq. (3.10) from τ'_λ to ∞, namely

$$\begin{aligned}\int_{\tau'_\lambda}^{\infty} \frac{\mathrm{d}}{\mathrm{d}t}\left[\cos(\theta)e^{\frac{t}{\cos(\theta)}} I_\lambda^+(t)\right]\,\mathrm{d}t &= -\cos(\theta)e^{-\frac{\tau'_\lambda}{\cos(\theta)}} I_\lambda^+(\tau'_\lambda) \\ &= -\int_{\tau'_\lambda}^{\infty} e^{-\frac{t}{\cos(\theta)}} S_\lambda(t)\,\mathrm{d}t.\end{aligned} \tag{3.11}$$

Dividing by $\cos(\theta)$ and multiplying by $e^{\frac{\tau'_\lambda}{\cos(\theta)}}$ yields

$$I_\lambda^+(\tau'_\lambda,\theta) = \frac{1}{\cos(\theta)}\int_{\tau'_\lambda}^{\infty} e^{\frac{\tau'_\lambda - t}{\cos(\theta)}} S_\lambda(t)\,\mathrm{d}t \tag{3.12}$$

as the expression for the outward spectral radiance at a given optical depth τ'_λ. Moreover, if the inward spectral radiance $I_\lambda^-(\tau'_\lambda)$ is to be calculated, integration of eq. (3.10) is required from τ'_λ to zero. Written down, this amounts to

$$\begin{aligned}-\int_0^{\tau'_\lambda} \frac{\mathrm{d}}{\mathrm{d}t}\left[e^{-\cos(\theta)\frac{t}{\cos(\theta)}} I_\lambda^-(t)\right]\,\mathrm{d}t &= -\cos(\theta)e^{-\frac{\tau'_\lambda}{\cos(\theta)}} I_\lambda^-(\tau'_\lambda) \\ &= \int_0^{\tau'_\lambda} e^{-\frac{t}{\cos(\theta)}} S_\lambda(t)\,\mathrm{d}t.\end{aligned} \tag{3.13}$$

Finally, dividing again by $-\cos(\theta)$ and multiplying by $e^{\frac{\tau'_\lambda}{\cos(\theta)}}$ leads to the inward spectral radiance at a given optical depth τ'_λ as

$$I^-_\lambda(\tau'_\lambda, \theta) = -\frac{1}{\cos(\theta)} \int_0^{\tau'_\lambda} e^{\frac{\tau'_\lambda - t}{\cos(\theta)}} S_\lambda(t)\, \mathrm{d}t. \tag{3.14}$$

Again, for both expressions $I^+_\lambda(\tau'_\lambda, \theta)$ and $I^-_\lambda(\tau'_\lambda, \theta)$ boundary conditions had to be applied, namely

$$I^-_\lambda(0, \theta) = 0, \quad e^{-t} S_\lambda(t) \xrightarrow{t \to \infty} 0. \tag{3.15}$$

These conditions mean, that at the location $\tau'_\lambda = 0$, there is no radiation directed towards the source; furthermore the source function $S_\lambda(t)$ is required not grow faster than e^t as optical depth τ'_λ increases.

3.1.3 Solutions for a homogeneous medium

Text.

3.1.4 The Eddington-Barbier approximation

Text.

3.1.5 The Milne-Eddington approximation

Text.

3.2 Spectroscopy

Text.

3.3 Polarimetry

Text.

3.3.1 Zeeman effect

Text.

3.4 Stokes parameters in stellar models

Text.

3.5 Stellar atmosphere models

3.5.1 Stellar atmosphere 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.

3.5.2 Solving a stellar atmosphere 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

$$\boldsymbol{x}_0 = \boldsymbol{M}^{-1}(\boldsymbol{y}_0). \tag{3.16}$$

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

$$\boldsymbol{F}(\boldsymbol{x}) \doteq \boldsymbol{M}^{-1}(\boldsymbol{y}_0) - \boldsymbol{x}. \tag{3.17}$$

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.

3.5.3 Calculation of synthesized datasets using a stellar atmosphere model

In the case, where data $\boldsymbol{x} \in \mathbb{R}^d$ is conditioned by context $\boldsymbol{y} \in \mathbb{R}^D$, it can be of interest to find a mathematical model implementing a transformation $\boldsymbol{y} = \boldsymbol{M}(\boldsymbol{x})$. In order to be able to train a normalizing flow implementing a transformation

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}), \tag{3.18}$$

if can be a commodity to know the true posterior distribution $p(\boldsymbol{x}|\boldsymbol{y})$, thus enabling a comparison 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

$$p(\boldsymbol{x}_{(k)}|\boldsymbol{y}_{(k)}) = \frac{p(\boldsymbol{y}_{(k)}|\boldsymbol{x}_{(k)})p(\boldsymbol{x}_{(k)})}{\sum_{j=1}^{L} p(\boldsymbol{y}_{(k)}|\boldsymbol{x}_{(j)})p(\boldsymbol{x}_{(j)})}, \tag{3.19}$$

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

Chapter 4

Normalizing flows

4.1 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.

4.2 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 4.2.0.1. *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*

$$p_x(x) = p_z(f(x))\frac{\mathrm{d}f(x)}{\mathrm{d}x}. \tag{4.1}$$

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

$$P(a \leq x \leq b) = P(c \leq z \leq d) \tag{4.2}$$

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

$$P(c \leq z \leq d) = \int_c^d p_z(z)\,\mathrm{d}z = \int_{f(a)}^{f(b)} f_z(z)\,\mathrm{d}z. \tag{4.3}$$

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

$$\frac{\mathrm{d}z(x)}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} \quad \Leftrightarrow \quad \mathrm{d}z = \mathrm{d}z(y) = \frac{\mathrm{d}f(x)}{\mathrm{d}x}\,\mathrm{d}x. \tag{4.4}$$

Therefore, the relation

$$\begin{aligned} P(c \leq z \leq d) &= \int_{f(a)}^{f(b)} p_z(z)\,\mathrm{d}z \\ &= \int_a^b p_z(f(x))\frac{\mathrm{d}f(x)}{\mathrm{d}x}\,\mathrm{d}x = P(a \leq x \leq b) \end{aligned} \tag{4.5}$$

obtains, thus

$$p_x(x) = p_z(f(x))\frac{\mathrm{d}f(x)}{\mathrm{d}x} \tag{4.6}$$

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

$$\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) \tag{4.7}$$

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

$$\begin{aligned} p_{\boldsymbol{x}}(\boldsymbol{x}) &= p_{\boldsymbol{z}}[\boldsymbol{f}(\boldsymbol{x})] \left| \det \left(\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right| \\ &= p_{\boldsymbol{z}}[\boldsymbol{f}(\boldsymbol{x})] \left| \det \begin{pmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_d(\boldsymbol{x})}{\partial x_d} \end{pmatrix} \right|. \end{aligned} \tag{4.8}$$

4.3 General principle of a normalizing flow

4.3.1 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

$$\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x}), \quad \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^d. \tag{4.9}$$

The aim of a normalizing flow is to learn a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}) \tag{4.10}$$

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

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})] \left| \det \left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|. \tag{4.11}$$

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 $\boldsymbol{x} = \boldsymbol{f}^{-1}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{z})$.

4.3.2 Requirements for a normalizing flow

Consider a function

$$\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) \tag{4.12}$$

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

$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_{(n)} \circ \cdots \circ \boldsymbol{f}_{(1)}(\boldsymbol{x}). \tag{4.13}$$

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.

4.3.3 Normalizing flows with conditioning data

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

$$\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x}), \quad \boldsymbol{y} \sim p_{\boldsymbol{y}}(\boldsymbol{y}), \quad \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^d. \tag{4.14}$$

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.

If there is no conditioning data $\boldsymbol{y}$ available, the normalizing flow can directly be applied to learn the probability distribution $p_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x})$ based on standard normally distributed data $z \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(0,1)^d$. In this case, the conditioning data $\boldsymbol{y}$ can be omitted in all formulae pertaining to the current section of this document.

4.3.4 Coupling layer normalizing flows

Consider a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}), \tag{4.15}$$

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

$$\begin{aligned} 1 = \int_{\mathbb{R}^d} p_{\boldsymbol{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} &= \int_{\mathbb{R}^d} p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})] \left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right| \mathrm{d}\boldsymbol{x} \\ &= \int_{\mathbb{R}^d} p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}, \end{aligned} \tag{4.16}$$

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

$$p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y}) = p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})] \left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|. \tag{4.17}$$

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

Since a concatenation of diffeomorphisms is again a 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

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)}. \tag{4.18}$$

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

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{z}_{(0)}) = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)}(\boldsymbol{z}_{(0)}). \tag{4.19}$$

In fig. 4.1, an overview of the working principle of a coupling layer normalizing flow is visualized. Because $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})$ is a concatenation of functions

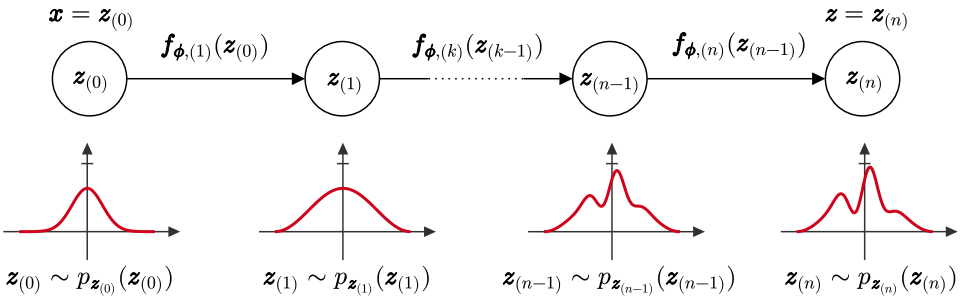


Figure 4.1: Visualization of the working principle of coupling layer normalizing flows.

$\boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})$ 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

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{k=1}^{n} \det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right), \tag{4.20}$$

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

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} = \begin{pmatrix} \frac{\partial f_{\boldsymbol{\phi},(k),1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \end{pmatrix} \tag{4.21}$$

with $k \in \{1, \ldots, n\}$. 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 $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}$.

4.4 Affine coupling layer normalizing flow

4.4.1 Construction of an affine coupling layer normalizing flow

Consider a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}), \tag{4.22}$$

where all quantities are given as defined in section 4.3.4.

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

$$\boldsymbol{\mu}_{(k)} : \mathbb{R}^{m+D} \to \mathbb{R}^{d-m}, \quad \boldsymbol{\sigma}_{(k)} : \mathbb{R}^{m+D} \to \mathbb{R}^{d-m} \tag{4.23}$$

with $m \in \{1, \ldots, d\}$ and $k \in \{1, \ldots, n\}$ 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 $k \in \{1, \ldots, n\}$ is now defined as an affine transformation, such that

$$f_{\boldsymbol{\phi},(k),l}(\boldsymbol{z}_{(k-1)}) = z_{(k-1),l}, \tag{4.24}$$

$$f^{-1}_{\boldsymbol{\phi},(k-1),l}(\boldsymbol{z}_{(k)}) = z_{(k),l} \tag{4.25}$$

for $l \in \{1, \ldots, m\}$ and

$$f_{\boldsymbol{\phi},(k),l}(\boldsymbol{z}_{(k-1)}) = \mu_{(k),l}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y}) + e^{\sigma_{(k),l}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y})} \cdot z_{(k-1),l}, \tag{4.26}$$

$$f^{-1}_{\boldsymbol{\phi},(k-1),l}(\boldsymbol{z}_{(k)}) = [z_{(k),l} - \mu_{(k),l}(\boldsymbol{z}_{(k),1:m}, \boldsymbol{y})] \cdot e^{-\sigma_{(k),l}(\boldsymbol{z}_{(k),1:m}, \boldsymbol{y})} \tag{4.27}$$

for $l \in \{m+1, \ldots, d\}$, where $\boldsymbol{f}_{\boldsymbol{\phi},(k)} = (f_{\boldsymbol{\phi},(k),1}, \ldots, f_{\boldsymbol{\phi},(k),d})^\top$ and $\boldsymbol{z}_{(k),1:m} = (z_{(k),1}, \ldots, z_{(k),m})^\top$.

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

that is

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}} = \begin{pmatrix} \mathbf{1}_m & \mathbf{0}_m \\ \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k),m+1:d}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1),m+1:d}} & \operatorname{diag}\left(e^{\sigma_{(k),m+1:d}(\boldsymbol{z}_{(k-1),1:m},\boldsymbol{y})}\right) \end{pmatrix}, \tag{4.28}$$

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

$$\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k),m+1:d}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1),m+1:d}} = \begin{pmatrix} \frac{\partial f_{\boldsymbol{\phi},(k),m+1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),m+1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),m+1}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),m+1}} & \cdots & \frac{\partial f_{\boldsymbol{\phi},(k),d}(\boldsymbol{z}_{(k-1)})}{\partial z_{(k-1),d}} \end{pmatrix}; \tag{4.29}$$

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

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right) = \prod_{j=m+1}^{d} e^{\sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m},\boldsymbol{y})}. \tag{4.30}$$

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

$$\log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right)\right] = \sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y}) \tag{4.31}$$

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

$$\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{k=1}^{n} \det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right). \tag{4.32}$$

Taking the logarithm of this expression and inserting the previous result eq. (4.31), this leads to

$$\begin{aligned} \log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right] &= \sum_{k=1}^{n} \log\left[\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi},(k)}(\boldsymbol{z}_{(k-1)})}{\partial \boldsymbol{z}_{(k-1)}}\right)\right] \\ &= \sum_{k=1}^{n} \sum_{j=m+1}^{d} \sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m}, \boldsymbol{y}). \end{aligned} \tag{4.33}$$

4.4.2 Training process for affine coupling layer normalizing flows

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. (4.17). Taking the natural logarithm of expression eq. (4.17) and inserting the result eq. (4.33) into it, one obtains

$$\log\left[p_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}|\boldsymbol{y})\right] = \log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\right) + \left|\sum_{k=1}^{n}\sum_{j=m+1}^{d}\sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m},\boldsymbol{y})\right|. \quad (4.34)$$

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

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x},\boldsymbol{y}) &= -\log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\right) - \log\left(\left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|\right) \\ &= -\log\left(p_{\boldsymbol{z}}[\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x})]\right) - \left|\sum_{k=1}^{n}\sum_{j=m+1}^{d}\sigma_{(k),j}(\boldsymbol{z}_{(k-1),1:m},\boldsymbol{y})\right|.\end{aligned} \quad (4.35)$$

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

4.5 Piecewise rational quadratic spline normalizing flow

4.5.1 Construction of a piecewise rational quadratic spline normalizing flow

Consider a diffeomorphism

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} = \boldsymbol{f}_{\boldsymbol{\phi},(n)} \circ \cdots \circ \boldsymbol{f}_{\boldsymbol{\phi},(1)} : \mathbb{R}^d \to \mathbb{R}^d, \quad \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}), \quad (4.36)$$

where all quantities are given as defined in section 4.3.4.

In the case of $\boldsymbol{f}_{\boldsymbol{\phi},(k)}$ being so-called piecewise rational quadratic coupling layers, these functions can be given by analytical expressions involving arbitrary functions

$$\boldsymbol{\mu}_{(k)} : \mathbb{R}^{m+D} \to \mathbb{R}^{(d-m)(3K-1)} \quad (4.37)$$

with $m \in \{1, \ldots, d\}$, $k \in \{1, \ldots, n\}$ and $K \in \mathbb{N}$, which are implemented by deep neural networks.

4.5.2 Training process for piecewise rational quadratic spline normalizing flows

Text.

Chapter 5

Proof of concept: Affine coupling layer normalizing flows

5.1 Moons

5.1.1 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

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})} : \mathbb{R}^2 \to \mathbb{R}^2, \quad p_{\boldsymbol{x}}(\boldsymbol{x}) \sim \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}(\boldsymbol{x}) \sim \mathcal{N}(0,1)^2. \tag{5.1}$$

To this end, a normalizing flow to implement the function $\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}})}$ was created as described in section 4.4. 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 $\boldsymbol{\phi}(\hat{\mathbf{x}})$.

5.1.2 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 process is visualized by

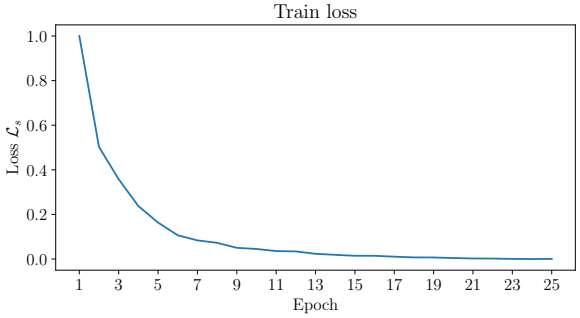


Figure 5.1: Visualization of the training process in terms of losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

fig. 5.1, where the loss $\mathcal{L}$ is scaled to $\mathcal{L}_s$ as

$$\mathcal{L}_s = \frac{\mathcal{L} - \min(\mathcal{L})}{\max(\mathcal{L} - \min(\mathcal{L}))}. \tag{5.2}$$

The performance after training can be seen as shown in fig. 5.2. 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`.

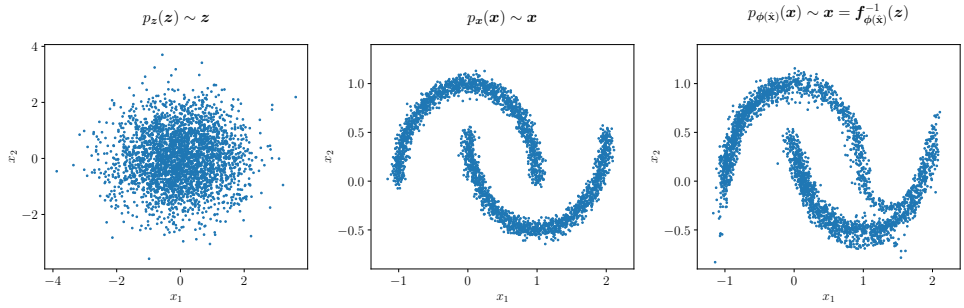


Figure 5.2: Results for a normalizing flow implementing the moons example.

5.2 Linear regression

5.2.1 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

$$\boldsymbol{M} : \mathbb{R}^2 \to \mathbb{R}^{10}, \quad \boldsymbol{x} \mapsto \boldsymbol{y} = \boldsymbol{M}(\boldsymbol{x}), \tag{5.3}$$

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

$$y_j = M_j(\boldsymbol{x}) = x_1 u_j + x_2 = a u_j + b_j, \quad j \in \{1, \ldots, 10\} \tag{5.4}$$

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

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^2 \to \mathbb{R}^2, \quad p_{\boldsymbol{x}}(\boldsymbol{x}) \sim \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}) \sim \mathcal{N}(0,1)^2. \tag{5.5}$$

To this end, a normalizing flow with context was created as described above and in section 4.4, 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 $\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$.

5.2.2 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. 5.3, the training process of the

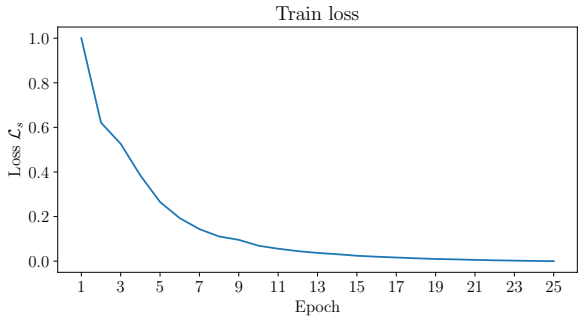


Figure 5.3: Visualization of the training process in terms of losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (5.2).

In fig. 5.4, 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 $\boldsymbol{x}_{(j)}$, $j \in \{1, \ldots, L\}$, $L \in \mathbb{N}$ denote training samples, then one obtains samples $\boldsymbol{z}_{(j)}$ by means of calculating

$$\boldsymbol{z}_{(j)} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}}, \hat{\mathbf{y}})}(\boldsymbol{x}_{(j)}), \tag{5.6}$$

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

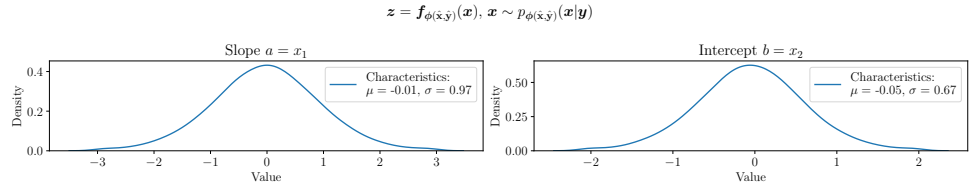


Figure 5.4: Calculated target densities.

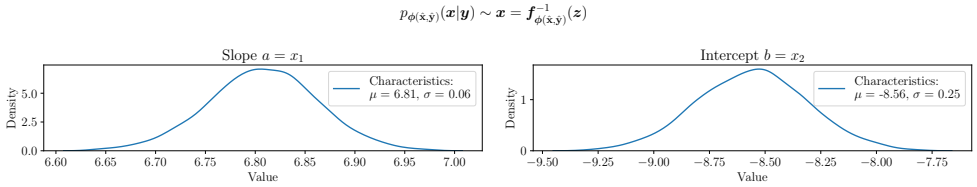
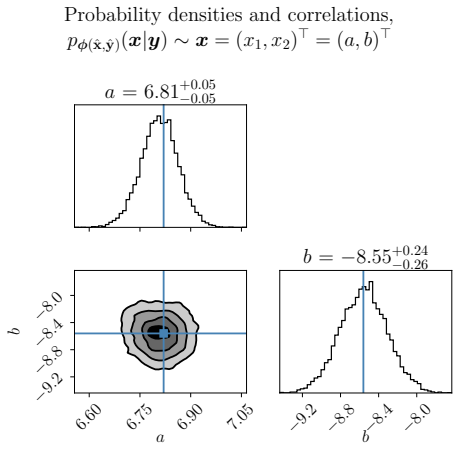


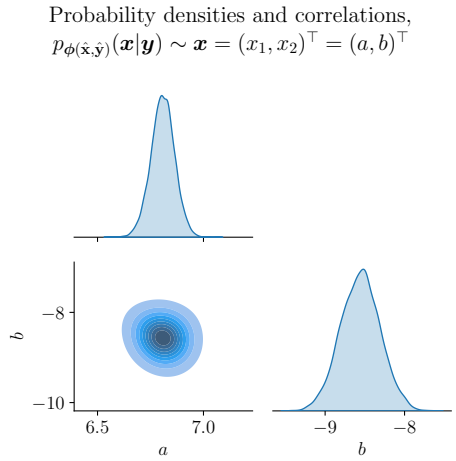
Figure 5.5: Calculated base densities.

Moreover, in fig. 5.5, 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. (5.3) based on test parameters $\boldsymbol{x}_{test} = (x_1, x_2)^\top = (a, b)^\top = (6.82, -8.56)$.

Furthermore, fig. 5.6 provides a view of the distributions aswell as of correlations for both the slope and intercept parameters. The corner and



(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 5.6: Corner and pairs plots for the calculated densities of slope a and intercept b.

pairs plots show nicely, that 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.

5.3 Feature extraction

5.3.1 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

$$\boldsymbol{M} : \mathbb{R}^{240} \to \mathbb{R}^{10}, \quad \boldsymbol{y} \mapsto \boldsymbol{x} = \boldsymbol{M}(\boldsymbol{y}). \tag{5.7}$$

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

$$\boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})} : \mathbb{R}^{10} \to \mathbb{R}^{10}, \quad p_{\boldsymbol{x}}(\boldsymbol{x}) \sim \boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}(\hat{\mathbf{x}},\hat{\mathbf{y}})}(\boldsymbol{x}) \sim \mathcal{N}(0,1)^{10}. \tag{5.8}$$

To this end, a normalizing flow with context was created as described above and in section 4.4, 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.

5.3.2 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.

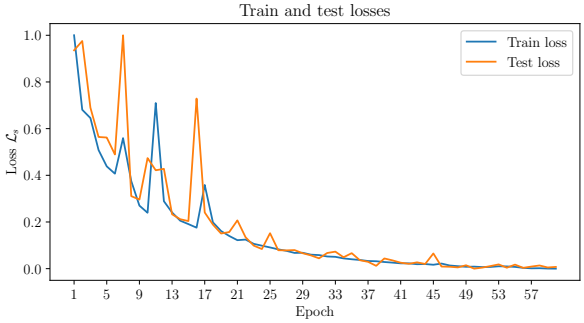


Figure 5.7: Visualization of the training process in terms of train and test losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

In fig. 5.7, the training process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (5.2).

In fig. 5.8, 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.

Moreover, in fig. 5.9, the calculated densities and correlations of feature parameters are depicted for a particular test observation $\boldsymbol{y}_{test}$ given

[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`.

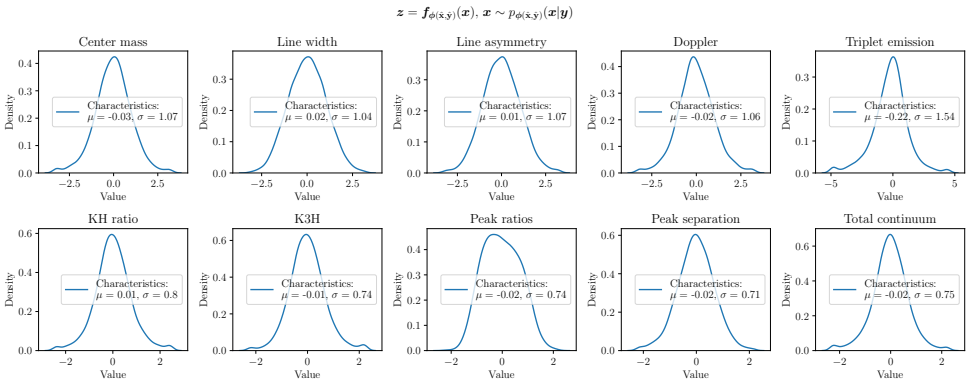


Figure 5.8: Calculated target densities.

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 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.

5.3.3 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 5.3.2. 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.

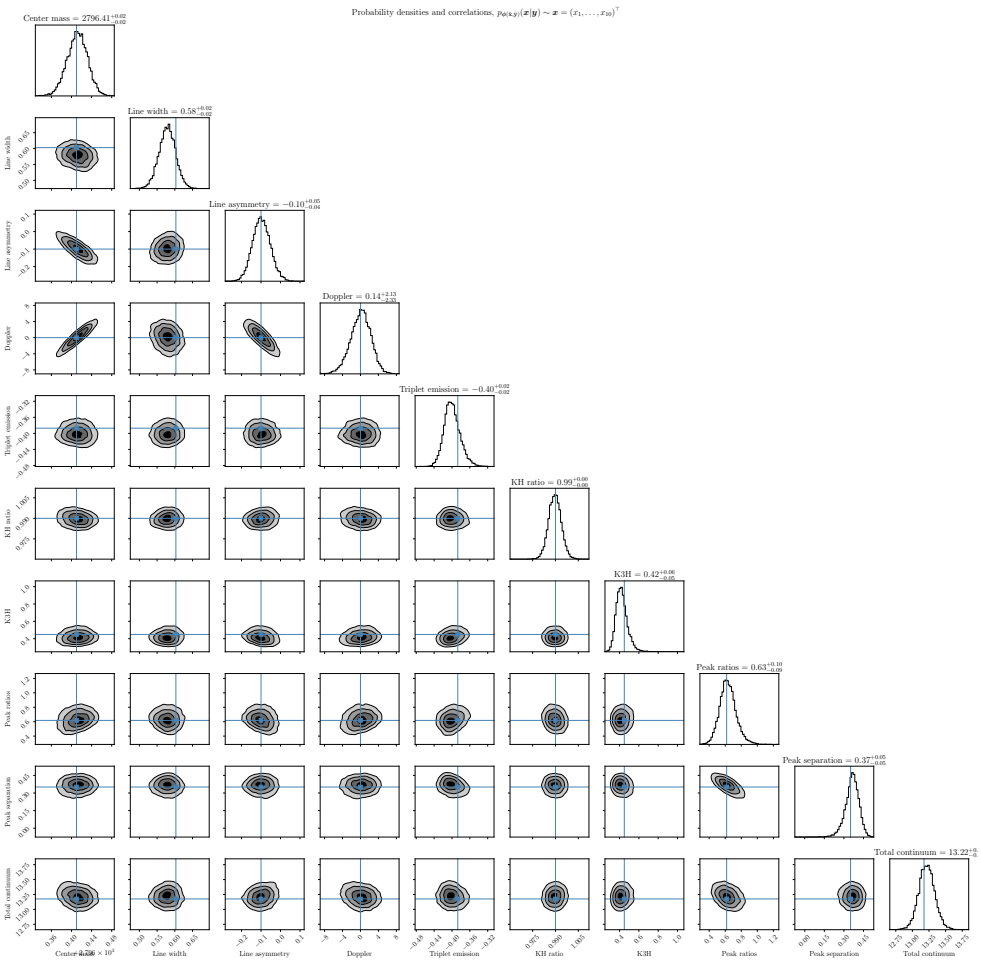


Figure 5.9: Corner plot for the calculated densities of feature parameters using the `corner` package.

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.

In fig. 5.10, the training process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (5.2).

In fig. 5.11, the latent densities for the feature parameters can be seen. As it is apparent from the figure, the latent densities are close to

[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`.

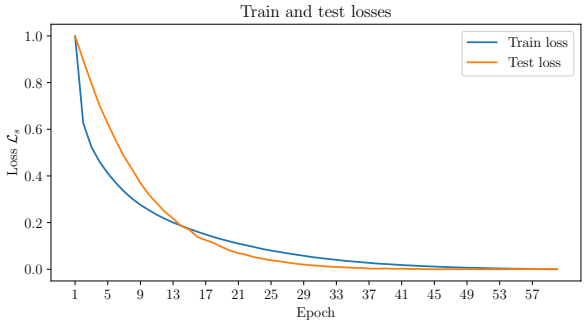


Figure 5.10: Visualization of the training process in terms of train and test losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

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.

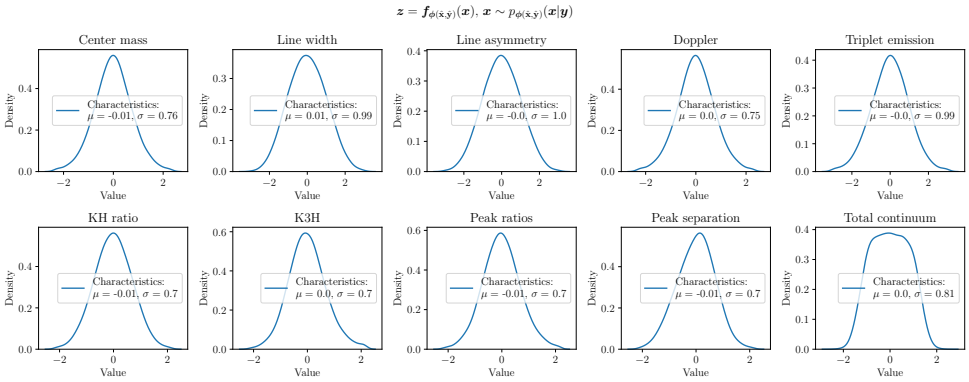


Figure 5.11: Calculated target densities.

Moreover, in fig. 5.12, 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 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 the density

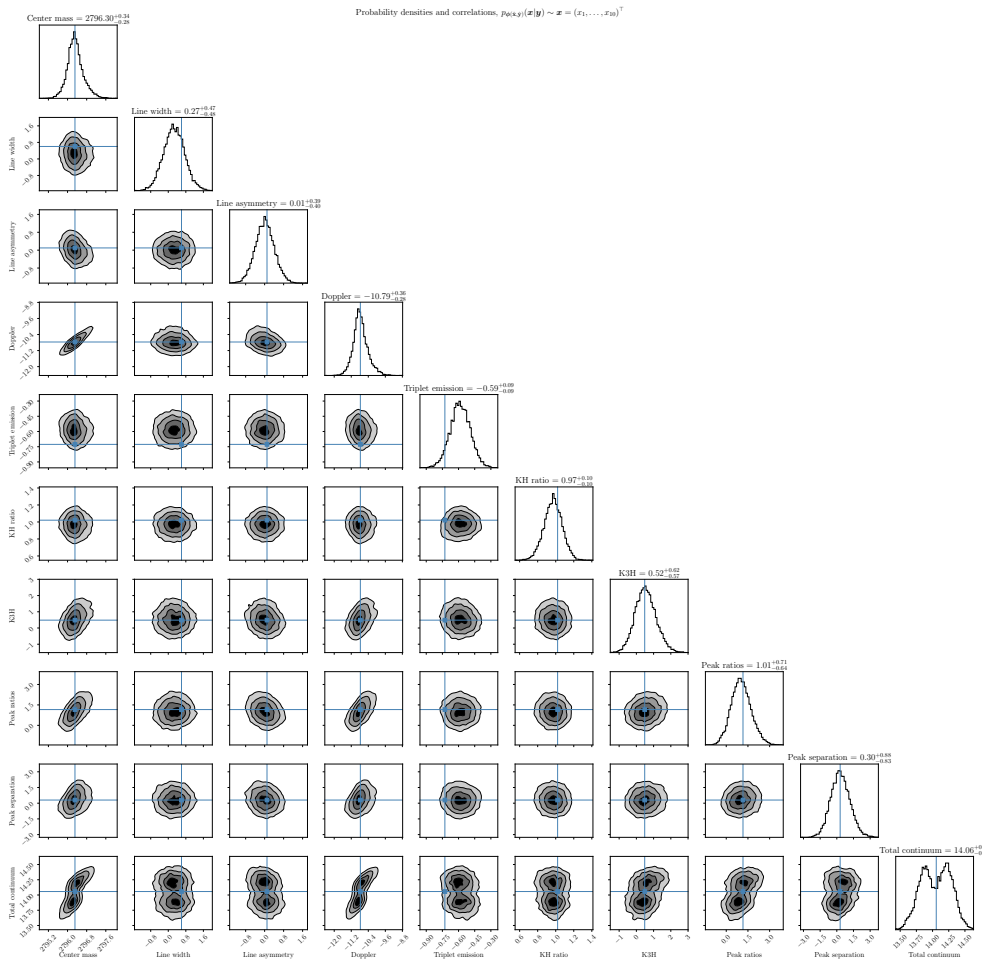


Figure 5.12: Corner plot for the calculated densities of feature parameters using the `corner` package.

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

5.3.4 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 corre-

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

sponding 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.

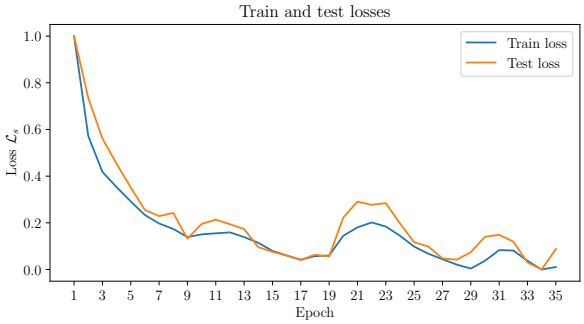


Figure 5.13: Visualization of the training process in terms of train and test losses vs. epochs. Note, that the data on the vertical axis was scaled beforehand.

In fig. 5.13, the training process of the normalizing flows is visualized in terms of a scaled loss $\mathcal{L}_s$ calculated according to eq. (5.2).

In fig. 5.14, 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 would likely lead to better results in terms more standard normally distributed latent densities for the features.

Furthermore, in fig. 5.15, 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

[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`.

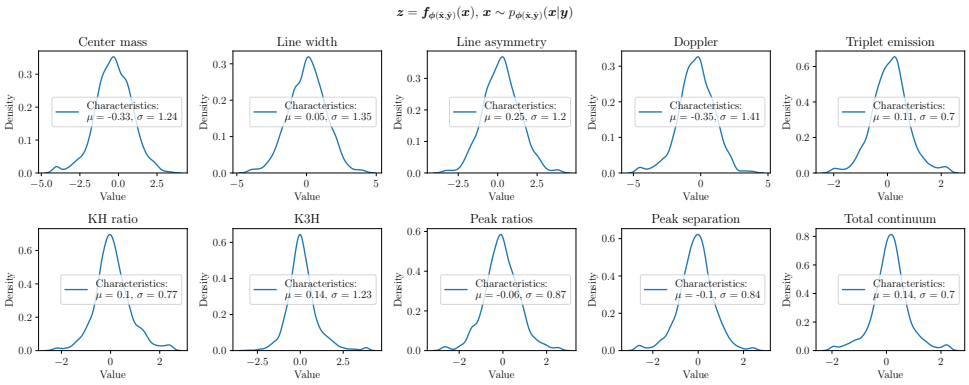


Figure 5.14: Calculated target densities.

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 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.

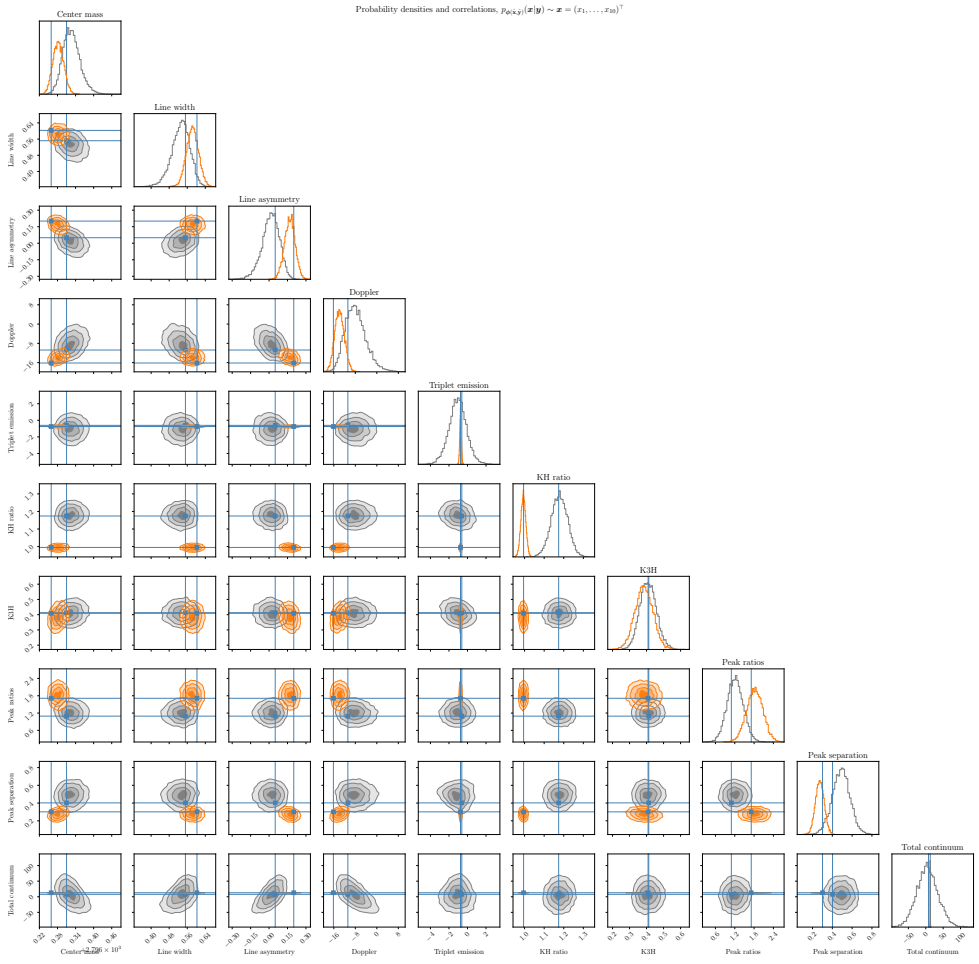


Figure 5.15: Corner plot for the calculated densities of feature parameters using the `corner` package.

Bibliography

[Amini, 2023] Amini, A. (2023). *Introduction to Deep Learning - Lecture notes*. Massachusetts Institute of Technology.

[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.

[Rutten, 2015] Rutten, R. J. (2015). *Introduction to astrophysical radiative transfer: Lecture notes*.