Ruprecht-Karls-Universität Heidelberg
Institut für Informatik
Lehrstuhl für Artificial Intelligence for Programming

Masterarbeit

Signature of warm dark matter in the cosmological density fields extracted using Machine Learning

Name:	Ander Artola
Matrikelnummer:	Matrikelnummer des Autors
Betreuer:	Name des Betreuers
Datum der Abgabe:	dd.mm.yyyy

Ich versichere, dass ich diese Bachelor-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Heidelberg, dd.mm.yyyy

Zusammenfassung

Dies ist eine Zusammenfassung der Arbeit.

Abstract

This is the abstract.

Contents

1 Introduction **1**

1.1 Cosmological preliminaries 1

1.2 Dark matter 3

1.2.1 Evidence for the inclusion of dark matter in the cosmological model 3

1.2.2 Manifold dark matter candidates 3

1.2.3 Constraining mechanisms for dark matter 3

1.3 The intergalactic medium 3

1.3.1 The Lyman-alpha forest as a probe of the IGM 3

2 Simulating the Lyman-alpha forest: The Sherwood simulation suite **6**

2.1 Prelude on cosmological simulation 6

2.2 Obtaining mock Lyman-alpha skewers from cosmological simulations . . 6

2.3 Statistical analysis of the effect of dark matter in the flux and density fields 6

2.4 Peculiar velocities and optical depth-weighted quantities 6

3 Deep Learning the Lyman-alpha forest **8**

3.1 Introduction and motivation for the use of Deep Learning 8

3.2 Fundamentals of (Bayesian) Neural Networks 14

3.2.1 Dataset generation, data augmentation and overfitting. 17

3.2.2 Deep learning architecture 20

3.2.3 Prediction uncertainty and Bayesian models 22

3.2.4 Hyperparameter selection 24

3.2.5 Loss function and training 26

3.3 Workflow implementation: Recovering IGM conditions from the Lyman-alpha forest 29

3.4 Recovered field statistics and uncertainties 34

3.5 Model interpretability and limitations 35

4 Constraining Warm Dark Matter at the density level **37**
4.1 Inference pipeline: from Lyman-alpha skewers to WDM constraints . . . 37
4.2 Inference testing on Sherwood spectra under realistic observational conditions . . . 39
4.2.1 Untrained DM models . . . 39
4.2.2 Realistic UVES observational conditions . . . 41
4.3 Inference on alternative hydrodynamical codes . . . 42
4.4 Comparison of the inference pipeline against Information Maximising Neural Networks . . . 42
4.4.1 Information Maximising Neural Networks . . . 42
4.4.2 IMNN training and non-linear summaries . . . 43
4.4.3 Summarising a Gaussian signal . . . 44
4.4.4 IMNN inference results on WDM masses . . . 46
4.5 WDM constraints from SQUAD DR1 observational data . . . 49

5 Conclusions **50**

Bibliography **51**

1 Introduction

1.1 Cosmological preliminaries

The currently accepted cosmological model describes space-time as a 4-dimensional Lorentzian manifold equipped with the Robertson-Walker metric [1]

$$\mathrm{d}s^2 = c^2\mathrm{d}t^2 - a(t)^2\left(\frac{\mathrm{d}r^2}{1-kr^2} + r^2\mathrm{d}\Omega^2\right), \tag{1.1}$$

with c the speed of light in vacuum, a the scale factor, k a curvature parameter and $\mathrm{d}\Omega$ the angular volume element in spherical coordinates. The scale factor is taken to be unity at the present time. At time t, a physical (proper) distance l_{phy} is then related to a comoving distance l_{com} by

$$l_{\mathrm{phy}} = a(t)l_{\mathrm{cov}}. \tag{1.2}$$

The physical distance at time t between an observer at $r = 0$ and a point at r is then

$$l_{\mathrm{phy}} = a(t)\int_0^r \frac{\mathrm{d}r}{\sqrt{1-kr^2}} = a(t)\chi(r). \tag{1.3}$$

The Robertson-Walker metric implies that for a radial luminous signal emitted at time t_e and received at time t_0, we have

$$\mathrm{d}s^2 = 0 \implies \frac{\mathrm{d}t_0}{a(t_0)} = \frac{\mathrm{d}t_e}{a(t_e)}. \tag{1.4}$$

As a consequence, the received frequency is redshifted according to

$$1 + z = \frac{\lambda_0}{\lambda_e} = \frac{\nu_e}{\nu_0} = \frac{a(t_0)}{a(t_e)}, \tag{1.5}$$

where z is the redshift.

The time-dependence of physical distances in equation (1.2) implies that an object whose comoving distance χ to an observer is constant recedes by following the Hubble

flow according to

$$v(t) = \dot{a}(t)\chi = \frac{\dot{a}}{a} a\chi = H(t) l_{\rm phy}, \tag{1.6}$$

where $H(t)$ is known as the Hubble factor. equation (1.6) is known a Hubble's law. At present time, $H(t_0) = H_0$ is referred to as Hubble'2 constant. For historical reasons, it is common to work with the reduced Hubble constant $h = H_0[\mathrm{km/s/Mpc}]/100$. Note that, according to equation (1.3), and using the Robertson-Walker metric for a radial light signal, we obtain

$$\mathrm{d}\chi = \frac{c\mathrm{d}t}{a} \implies \chi = \int_a^1 \frac{\mathrm{d}a}{a\dot{a}} = \int_0^z \frac{c\mathrm{d}z}{H(z)}. \tag{1.7}$$

As a consequence, the proper line element satisfies

$$\mathrm{d}\chi = \frac{c\mathrm{d}z}{H(z)} = \frac{dl}{a(t)} \implies \frac{\mathrm{d}l}{\mathrm{d}z} = \frac{c}{(1+z)H(z)}, \tag{1.8}$$

which will be useful when integrating quantities along a line of sight. When working with such sightlines in spectroscopy, it is often advantageous to work with velocity units instead of redshifts (or proper distances). Differentiating equation (1.6) and considering a slow varying Hubble factor around a mean redshift $\overline{z}$, we obtain the following useful expression:

$$\mathrm{d}v = H(\overline{z})\mathrm{d}l = H(\overline{z})\frac{c\mathrm{d}z}{(1+\overline{z})H(\overline{z})} = \frac{c\mathrm{d}z}{1+\overline{z}}. \tag{1.9}$$

The evolution of the scale factor (and hence of the redshift) with time is completely determined by the energy content of the universe through Einstein's field equation, which is known as Friedmann's equation in this context

$$H^2 = H_0^2 \left(\Omega_{\rm M}(1+z)^3 + \Omega_{\rm R}(1+z)^3 + \Omega_\Lambda + \Omega_K(1+z)^2\right) = H_0^2 E(z)^2, \tag{1.10}$$

where the density parameters Ω are related to the physical densities of the components according to

$$\begin{aligned} \Omega_{\rm M} &= \frac{8\pi G}{3H_0^2}\rho_{M0} \\ \Omega_{\rm R} &= \frac{8\pi G}{3H_0^2}\rho_{R0} \\ \Omega_\Lambda &= \frac{8\pi G}{3H_0^2}\rho_\Lambda \\ \Omega_{\rm K} &= -\frac{k}{H_0^2} \end{aligned} \tag{1.11}$$

In equation (1.11), ρ_{M} denotes the matter density of the universe, ρ_{R} the radiation density, and ρ_Λ the dark energy component. In the following, the values used for the cosmological parameters are $\Omega_m = 0.308, \Omega_\Lambda = 0.692, h = 0.678, \Omega_b = 0.0482, \sigma_8 = 0.829$ and $n = 0.961, \Omega_{\mathrm{K}} \approx 0$, as obtained from CMB measurements by the Planck Collaboration [2]. With the previous cosmological parameters, the matter and cosmological constant are equal when

$$\Omega_{\mathrm{M}}(1+z)^3 = \Omega_\Lambda \implies z \approx 0.3. \tag{1.12}$$

In consequence, at the redshift of interest for this work, $z \sim 4-5$, the universe is well-described as a matter dominated universe.

1.2 Dark matter

1.2.1 Evidence for the inclusion of dark matter in the cosmological model

1.2.2 Manifold dark matter candidates

1.2.3 Constraining mechanisms for dark matter

1.3 The intergalactic medium

1.3.1 The Lyman-alpha forest as a probe of the IGM

Let us now describe how an intergalactic cloud (with no peculiar velocity) along the line of sight of a quasar affects its spectrum, allowing for a powerful probing mechanism of the IGM. Consider the situation illustrated in figure 1.1, where a QSO at redshift z_{QSO} emits photons, and consider the propagation of an emitted photon with rest-frame frequency ν_e. Those photons are redshifted and are absorbed at z_α by a neutral hydrogen absorber with local number density $n(z_\alpha)$ producing an absorption feature in the flux at the rest-frame Lyman-α resonance $\lambda_\alpha \approx 1215$Å. The unabsorbed photons are then redshifted and are detected by an observer at $z = 0$ and a frequency ν_o. The relationship between the frequencies mentioned above is then:

$$\nu_o = \frac{\nu_e}{1+z_e} = \frac{\nu_\alpha}{1+z_\alpha} \tag{1.13}$$

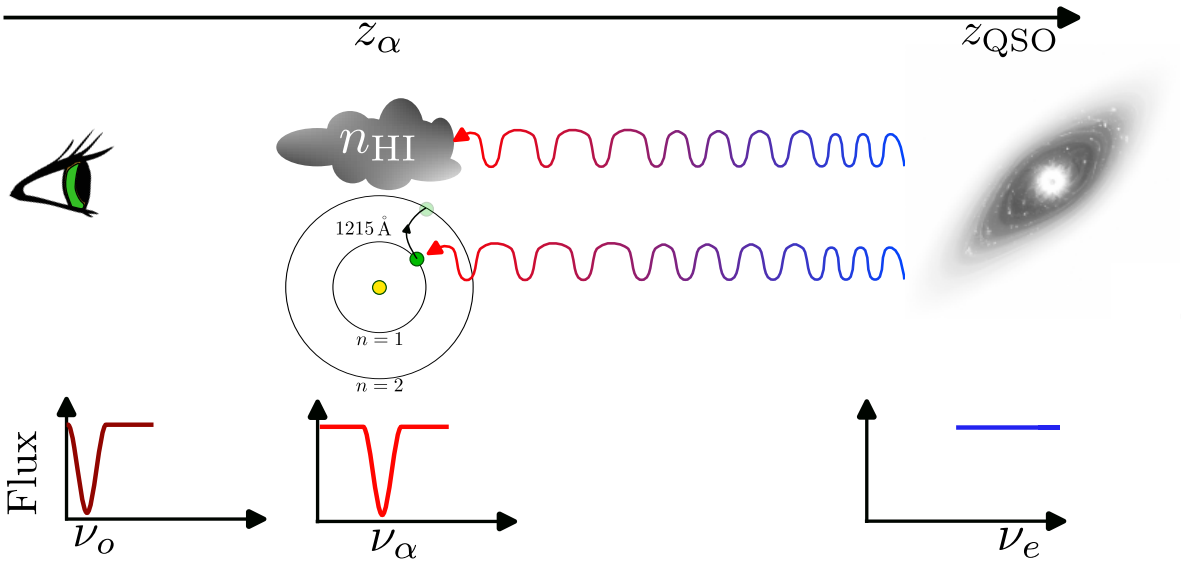


Figure 1.1: Illustration of the Lyman-α absorption by neutral hydrogen at $z = z_\alpha$ in the line of sight of a QSO at $z = z_{\mathrm{QSO}}$. In the observer's rest frame, the observed frequency is ν_o. The associated frequency emitted by the QSO is ν_e.

We are interested in studying the effect of the Lyman-α absorbed at z_α. The observed flux attenuation at the observed frequency ν_0 is then expressed as $\exp(-\tau_\alpha)$, with τ_α the Lyman-α opacity at the observed frequency, which depends on the observer's density and the Lyman-α cross-section $\sigma_\alpha(\nu)$. Observe now that since the Lyman-α cross-section is strongly peaked at the resonance ν_α, but can have a non-zero width, a nearby neutral hydrogen cloud might absorb photons at a redshift different to z_α that would have contributed to the observed flux at frequency ν_o. With this consideration, we integrate over the line of sight to obtain the Lyman-α opacity at the observed frequency

$$\tau_\alpha(\nu_o) = \int_o^{z_{\mathrm{QSO}}} n_{\mathrm{HI}}(z)\sigma_\alpha[\nu_o(1+z)]\mathrm{d}z. \tag{1.14}$$

If we now take $\sigma_\alpha(\nu)$ to be a Dirac delta centered at the resonance ν_α and we integrate equation (1.14) by using 1.8 we obtain

$$\tau_\alpha(\nu_o) \approx \frac{cn_{\mathrm{HI}}(z_\alpha)\sigma_\alpha}{H_0\Omega_{\mathrm{m}}^{1/2}(1+z)^{1/3}}, \tag{1.15}$$

where now $\sigma_\alpha = 4.5 \times 10^{-18}\mathrm{cm}^2$ is to total Lyman-α cross-section. equation (1.15) is know as the Gunn-Peterson approximation for the Lyman-α opacity of the IGM [3]. equation (1.15) demonstrates that quasar spectra are a useful probe of the intergalactic neutral hydrogen density.

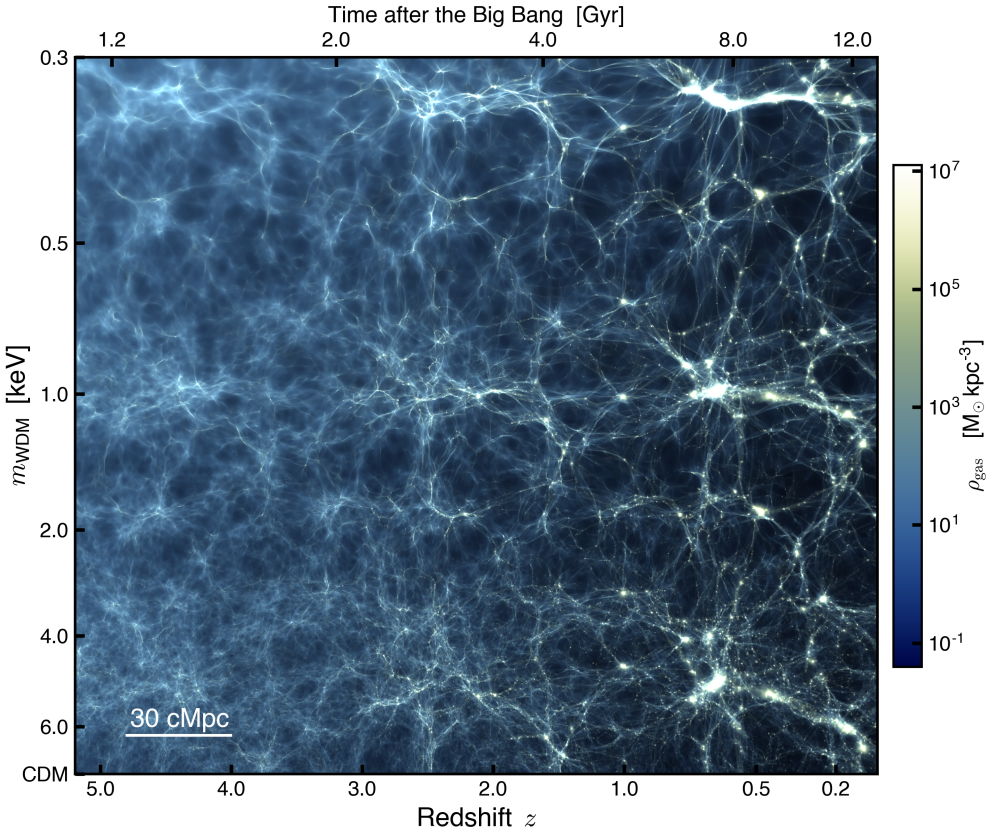


Figure 1.2: Baryonic density plot of the IGM as a function of redshift, and the WDM model mass. On the horizontal axis, the time evolution shows how gravity collapses dense regions into structures. On the vertical axis, the WDM free-streaming length suppress small-scale clustering. Extracted from [6].

The Gunn-Peterson Lyman-α opacity can then be used to estimate the average neutral hydrogen fraction $x_{\rm HI}$. The evolution of the observed Lyman-α optical depth indicates that the IGM is highly ionized at $z \lessapprox 5.5$, [4], [5].

discuss villa plot and pdf and ps plot

ADD notations about overdensityes, relatio nship with omega etc IGM typical density, temp, temp-density relationship, UVB,phoion equiv Gun Peterson test, and reionization constraints contribution to UVB? ADD IN THE INTRO SOME ABOUT GR AND METRIC ADD SECTION 2.8.2 OF BOOk ADD BOLTON ARTICLE WITH UVB AND INFO

2 Simulating the Lyman-alpha forest: The Sherwood simulation suite

2.1 Prelude on cosmological simulation

2.2 Obtaining mock Lyman-alpha skewers from cosmological simulations

2.3 Statistical analysis of the effect of dark matter in the flux and density fields

DISCUSS plots with color bar of pdf and ps of Density discuss the boeara discrepancy

2.4 Peculiar velocities and optical depth-weighted quantities

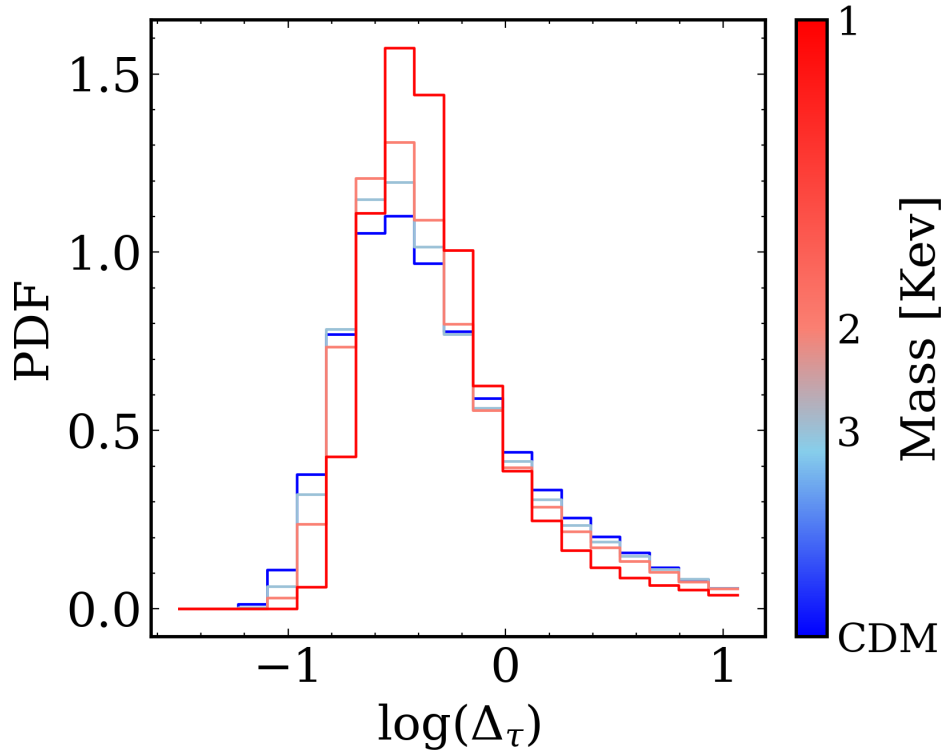


Figure 2.1: The Δ_τ probability distriburion function (PDF) for different WDM models in the SHERWOOD suite.

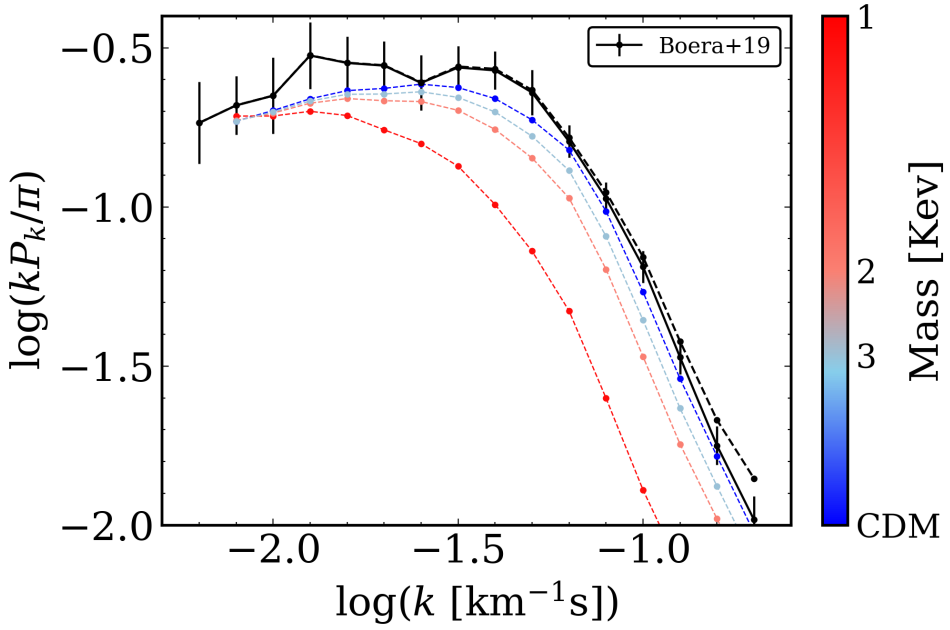


Figure 2.2: The Lyman-α flux power spectrum for different WDM models in the SHERWOOD simulation suite. For reference, we also plot the observed PS by [7].

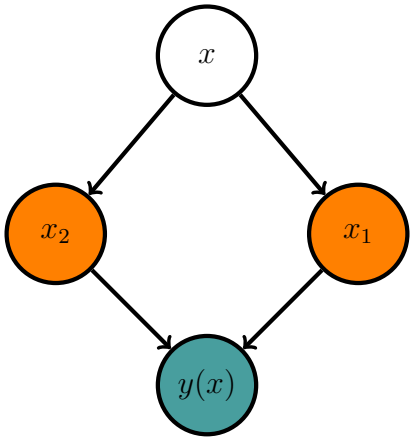
3 Deep Learning the Lyman-alpha forest

3.1 Introduction and motivation for the use of Deep Learning

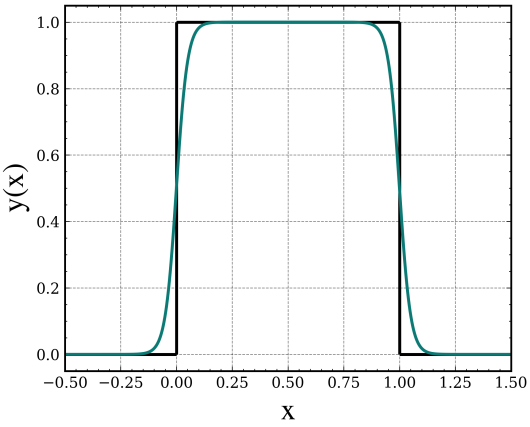
Fundamentally, a neural network is a directed and acyclic computational graph [8]. It represents a set of operations that transform input data input an output. In the simplest case of a fully connected network, the nodes of this graph are represented by neurons. Neurons are organized on successive layers, making the information flow from a layer to the following one. In the most basic scenario, the neurons in a given layer are linearly connected to the neurons in the previous layer. Each neuron then adds a bias to the result of the computation and applies a non-linear function to the result, which determines the activation state of the neuron. Consider the graph shown in figure 3.1a, where the input neuron has a value x and the output neuron has a value $y(x)$. The intermediate layers have values

$$\begin{cases} x_1 = \sigma(\alpha_1 x + \beta_1) \\ x_2 = \sigma(\alpha_2 x + \beta_2) \end{cases}, \tag{3.1}$$

where α_i are the linear weights, β_i the biases, and σ is a non-linear activation function. Typical choices include tanh on ReLU (Rectifier Linear Unit) given by $\mathrm{RELu}(x) = \max(0, x)$. Graphs such as the one shown in figure 3.1a as know as *fully connected layers*. Much more complex architectures have of course been investigated. Depending on the specific problem and dataset, we can incorporate a priori knowledge of the problem in the design of the network. For instance, in Computer Vision, the use of *convolutional layers* especially target at identifying key features in images has proven to be extremely successful [9]. In the analysis of time series, Long short-term memory (LSTM) neurons allow the network to “remember" information from previous inputs [10].



(a) Graph for a simple MLP with a hidden layer (in orange) and an output neuron (in cyan).



(b) Unit impulse (in black) and the output of the MLP shown in cyan approximating the impulse.

Figure 3.1: A simple multilayer perceptron (MLP) with a single hidden layer and a tanh activation function is able to approximate a unit pulse function. From left to right and top to bottom, the three biases are $\{0, 20, 0\}$ and the four weights are $\{20, -20, 1/2, 1/2\}$.

From the theoretical standpoint, neural networks are universal approximations (for sufficiently well-behaved functions), which makes them especially appealing in the modelling of complex systems. A concrete results is as follows [11]:

Theorem 1 (Universal approximation theorem)**.** *If* $f\colon \mathbb{R}^n \to \mathbb{R}^m$ *is a Lebesgue p-integrable function and* $\varepsilon > 0$*, then there exists a fully connected ReLU network* $F\colon \mathbb{R}^n \to \mathbb{R}^m$ *such that*

$$\int_{\mathbf{R}^n} ||f(x) - F(x)||^p \, \mathrm{d}x < \varepsilon.$$

Theorem 1 can be expanded to include tight bounds on the depth or width of the network, which then depend on n and m. Note that all the complexity of a fully connected neural network is generated by the non-linear activation function. With a linear activation function, a fully connected networks would be an affine transformation, which cannot approximate arbitrary non-linear functions.

This theoretical result can be easily visualized be understanding how a simple fully connected network (a multilayer perceptron, MLP) can approximate a unit impulse function. It is then enough to recall that a linear combinations of step functions can approximate any integrable function. In figure 3.1a we show a MLP consisting of an input neuron, and output neuron and a hidden layer with two neurons with a tanh activation function. From left to right and top to bottom, we set the three biases as

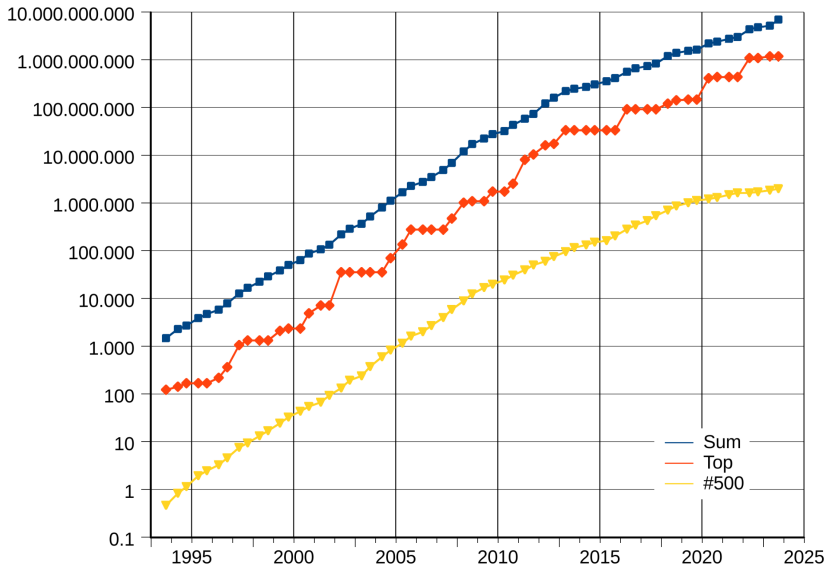


Figure 3.2: Evolution of the largest supercomputers in the TOP500 list in recent years (x-axis). The y-axis shows the peak performance in GFLOPS for the first-ranked computer (red), the last-ranked computer (yellow), and the cumulative power for the top-500 computers (blue). Source: Wikipedia.

$\{0, 20, 0\}$ and the four weights as $\{20, -20, 1/2, 1/2\}$. The resulting MLP then produces the approximation shown in figure 3.1b of the unit impulse over $[0, 1]$. By expanding the number of neurons in the hidden layer we could approximate impulse functions of arbitrary width and height and centered at every arbitrary value. The high degree of expressivity of neural networks makes them particularly suited to parametrically approximate complex non-linear relationships between variables that would otherwise be intractable.

Theorem 1 establishes a theoretical aspect of neural networks supporting their utility, but it does not provide a method (nor does it guarantee the existence of any) to find networks that approximate a particular function of interest. The process of fitting a statistical model (such as a neural network) to a dataset of interest in know as *supervised learning*, an constitutes one of the main aspect of this work, as we will explain in the rest if this section. In recent years, the main points have allowed the rapid expansion of machine learning, marking a new era in the use of large parametric networks for real-world problems:

- **Hardware development.** The rapid (quasi-exponential)growth in computer power in recent years, as exemplified by Moore's law [12] means that extremely

deep and complex models can now be trained and effectively used. As a reference, the language models LLAMA by Meta can have over 65 billion parameters [13]. To handle this amount of data, High Performance Computing (HPC) infrastructures (such as supercomputers) are needed. Figure 3.2 shows the rapid evolution for the top supercomputers in the world, as ranked by the TOP500 list[1]. In 1993, the Fujitsu Numerical Wind Tunnel in Japan topped the list with 124 GFLOPS. In 2023, the list is topped by Frontier in the United States, with over 1000 PFLOPS, which corresponds to an 8000-fold improvement. Progress in hardware technology as also been remarkable. For instance, Graphics Processing Unit (GPU) are now common when training machine learning models. As a last example, in May 2017, Google introduced an architecture known as Tensor Processing Unit (TPU) especially designed to accelerate neural network operations [14]. HPC also enables the generation of synthetic datasets from simulations, that can then be used as training datasets. We will leverage this idea in the rest of this work.

- **Computationally-efficient algorithms.** Together with hardware development, we have seen a rapid evolution of computational and mathematical algorithms in the field of statistical learning that have enabled the efficient utilization of HPC resources. The epitome of such algorithm might be *backpropagation* [15] which is the most common algorithm used in the training phase to update the weights when deploying a neural network. Together with backpropagation, a growing set of optimizers for deep learning problems have been developed. A popular choice is Adam, which was originally published in 2017 [16].

- **Availability of large datasets.** The advent of next-generation instruments and data-collections systems provides the scientific community with increasingly large datasets. Prominent examples in the field of astronomy include Gaia [17], whose third data release (DR3) includes 10TB of data for 1.46 billion sources[2], or the Euclid telescope, expected to deliver 850 GB of compressed data per day[3]. At last, in figure 3.3 we show the evolution in the number of SNe discoveries per year. The figure as been obtained from [18].

[1] `https://top500.org`
[2] `https://www.cosmos.esa.int/web/gaia/dr3`
[3] `https://sci.esa.int/web/euclid/-/46661-mission-operations`

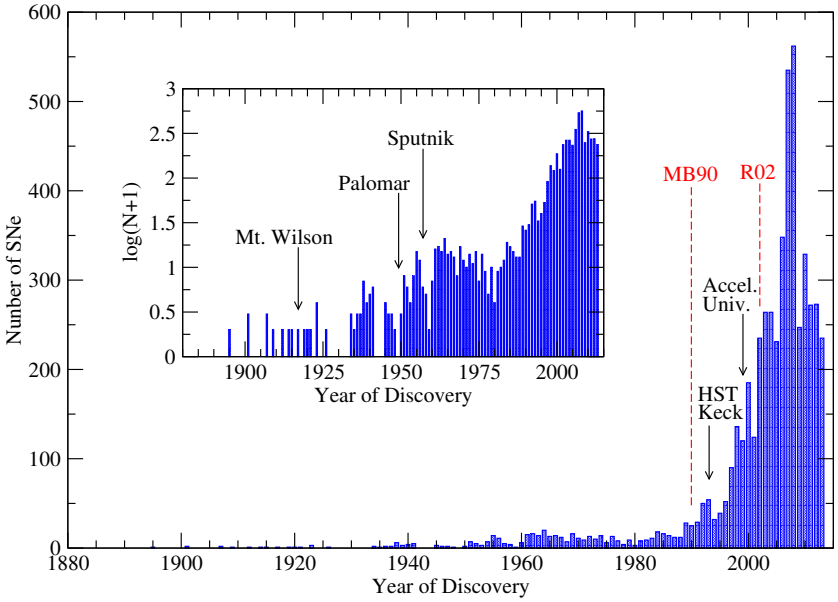


Figure 3.3: Histogram showing the number of SNe discovered each year as given by the Asiago Supernova Catalogue.

Let us demonstrate the relevance, pertinence, and range of applicability of machine learning methods in astronomy and cosmology by examining recent endeavours documented in the literature. While not exhaustive, this examination provides a broad overview of previous efforts in the field, and illustrates some recent successes of machine learning-oriented approaches in data-driven problems.

The abundance of data from surveys covering large regions of the sky aimed at targeting QSOs makes supervised statistical techniques a particularly attractive data analysis technique. For reference, the main quasar sample in the data release number 16 for the Extended Baryon Oscillation Spectroscopic Survey (eBOSS) contains 434820 targets with redshifts in the range $0.8 < z < 2.2$ [19]. The prediction of the intrinsic Lyman-α emission line from high redshift QSOs is a non-trivial problem that has important implication in the study of IGM damping wings and the reionization of the later [20]. Reconstruction techniques often rely on the correlation of the Lyman-α peak with other observable lines and information redward of the Lyman-α line. Machine learning approaches are then suitable to connect the unattenuated information redward of the Lyman-α peak with its intrinsic profile. In *Blind QSO reconstruction challenge: Exploring methods to reconstruct the Lyα emission line of QSOs* [21], the authors perform an in-depth comparison of different state-of-the-art techniques based on statistical learning. The authors blindly evaluate their performance on two QSOs samples randomly

extracted from X-Shooter and BOSS with $3.5 < z < 4.5$, in such a way that selecting samples already used in the training data-set was avoided. The various techniques range from principal component analysis (PCA) approaches, such as [22], to deep learning networks [23]. The authors conclude that the better performing pipelines consistently rely on machine learning approaches. The authors caution against overreliance on machine learning techniques due to their potential lack of statistical uncertainty, which is one of the main aspect that we will develop in this work.

Parameter inference on WDM using deep learning has already been tentatively explored in the recent literature. The paper *Inferring Warm Dark Matter Masses with Deep Learning* [24], which is especially relevant for this work, demonstrates that neural networks can be used to recovered WDM parameters from observed field density images. The authors present a suite of 1500 cosmological N-body simulations with varied WDM mass in the range 2.5 to 30 KeV. Field density images of size $25\mathrm{h}^{-1}\mathrm{Mpc}$, with varied image resolution, simulation resolution and redshift (in the range $0 \leq z \leq 5$) are extracted from the simulation runs. The images are augmented usual standard techniques (such as image rotation) and then incorporated into training datasets. The authors use a Convolutional Neural Network (CNN) trained to directly predict WDM masses based on an input density field image. Their fiducial convolutional network trained with the set of highest resolution image is able to accurately recover WDM masses with an accuracy of ± 1 KeV for models up to 10 KeV. After this threshold value, the network is no longer able to recover the true mass as predicts an approximately constant mass, see figure 3.4. Note that in the architecture presented by the authors, the network is also trained to predict an uncertainty estimate. Another interesting insight offered by this paper refers by the capacity of neural network to make use of the full information contained on a field distribution (in this case, the density field). By training a network on a full density field image the models learns the relevant properties of the field that can lead to an accurate parameter prediction. The authors train another model only on summary statistic, in particular, on the density power spectrum, and compare its performance with their fiducial model trained on the full images. The model trained only on the power spectra shows a significantly degraded performance, with higher uncertainties and accurate prediction only up to 5.5 KeV. This illustrate how deep learning techniques can help harvesting the full information present in a field (or generally, in a complex input tensor).

Another recent use of deep learning to analyse IGM data can be found in the paper *LyαNNA: A Deep Learning Field-level Inference Machine for the Lyman-α Forest* [25],

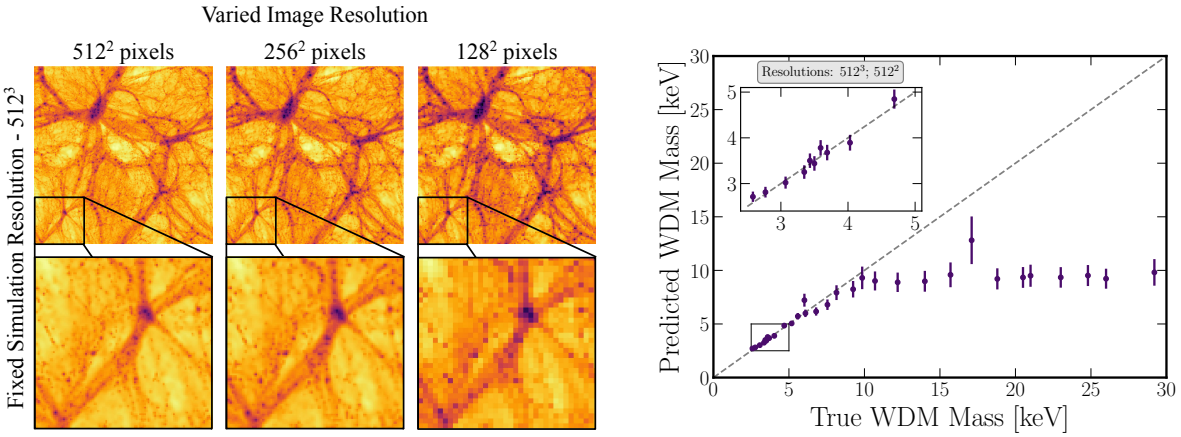


Figure 3.4: Figure extracted from [24]. The left panel shows a sample image density field used in the training data by the authors, with varied image resolution. The right panel shows a sample a predicted WDM masses versus the true WDM mass of the simulation for their fiducial neural network, which can accurately recover the WDM model within a 1 KeV accuracy up to 10 KeV.

where authors harvest the field level potential of residual convolutional networks to perform inference on the thermal parameters of the IGM, namely T_0, the temperature at mean density, and γ, the slope of the temperature-density relation. Their model is trained on simulation boxes with side-length 120 Mpc from which 10^5 sightlines are extracted and processed to produce mock Lyman-α spectra. The simulation boxes are run with different thermal parameters, by sampling 121 (T_0, γ) combinations in the parameter space. The network is trained on 24000 labeled spectra from the mix of thermal models, and the architecture is designed to predict a mean value for the thermal parameters as well as an estimate for the parameter covariance matrix. Figure 3.5 shows the scatter in the point predictions for (T_0, γ) for a set of 4000 unseen test spectra. The true parameter values, shown as dashed lines, are recovered by the average of the point predictions, shown as the dark green cross. The authors also perform a comparison with an inference pipeline based on the traditional transmitted flux power spectrum, and find that the posterior constraint using the machine learning field-level approach are 5.65 times tighter.

3.2 Fundamentals of (Bayesian) Neural Networks

A very general regression problem in statistical learning [26] arises when we observe a quantity Y that is assumed to depend on an independent variable X through a relation

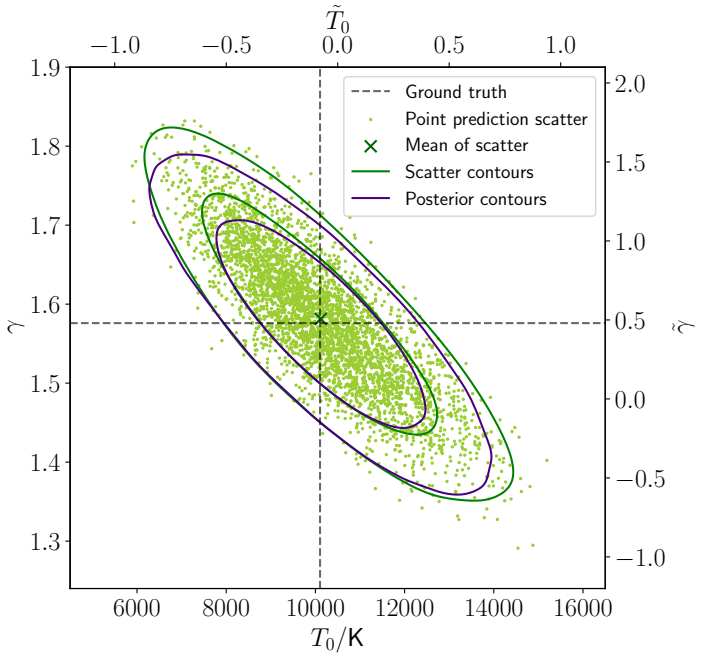


Figure 3.5: Figure extract from [25] showing the performance of the neural network recovering thermal parameters on a set of unseen skewers. The true parameter values, shown as dashed lines, are recovered by the average of the point predictions, shown as the dark green cross.

of the form

$$Y = f(X) + \mathcal{N}(0, \sigma), \tag{3.2}$$

where f is a function defining the relation $Y = Y(X)$ and $\mathcal{N}(0, \sigma)$ is an error term modeling a zero-mean Gaussian noise. The problem then consists of estimating f given a sample of observations $\{X_i, Y_i\}_i$ to obtain a functional form $\hat{f}$ that can then be evaluated to obtain predictions. Following the examples in section 3.1, X might be the flux spectra redward of the Lyman$-\alpha$ peak, and Y might then be the intrinsic shape of the Lyman$-\alpha$ peak in the quasar spectra. The noise then is produced by different sources, for instance, associated with the instrumental devices. We could then use the obtained $\hat{f}$ to predict to intrinsic Lyman$-\alpha$ peak of a QSO, $\hat{f}(X)$, when we only have information about the spectrum redward of it, X. As it is expected, the quality of our prediction depends on how accurate is the approximation $\hat{f}$ with respect to f, but also on how noisy our data is. In fact, we see that, if the true quantity associated with X is Y:

$$(Y - \hat{f}(X))^2 = (f(x) + \mathcal{N}(0, \sigma) - \hat{f}(X))^2.$$

Taking expected values and noting that the only stochastic component here is the noise, we obtain

$$\mathbb{E}[(Y - \hat{f}(X))^2] = (f(X) - \hat{f}(X))^2 + \sigma^2. \tag{3.3}$$

In equation (3.3), the first term of the right-hand side represents the error produced by approximating f by $\hat{f}$, while the second term is a theoretical limit imposed by the noise properties.

Parametric methods represent a powerful statistical learning tool to approximate the target relation f between variables. Methods such as linear regression, which approximates $Y = f(X)$ using a linear functional form on the parameter values, offer limited flexibility in terms of approximating complex relations but allow for a high degree of interpretability of the model. On the other end of the spectrum, deep learning models offer a large degree of expressivity (see 3.1), but interpretation of individual parameters is often not possible. Increasing the number of parameters can also cause the model to learn from spurious structures in the finite data sample, or the learn from possible noise correlation. This process, known as *over-fitting*, can significantly impact the predictive performance of the model when predicting on unseen data. Multiple techniques have been explored to mitigate over-fitting [27], we will adopt some of them in this work and explain them in the following sections.

In this section we will discuss the basic workflow involving a deep learning model in a general and abstract scenario. We will restrict ourselves to the topics that are relevant for this work, and later explain in detail how we implement this workflow for our problem. The general workflow when working with a deep learning statistical model under supervised learning includes the following phases:

1. Collecting/data and processing it to generate a training dataset. This requires specific domain expertise to assess the quality and representability of the data.

2. Designing a deep learning architecture, i.e., a computational graph that depends on a set of parameters and that generates target outputs from the input data.

3. Using the available data to train the model. This is done by optimizing the model parameters by minimizing a selected loss function.

4. Assessing the performance of the network a validation dataset, previously unseen.

5. Using the model on real unseen data to make predictions.

3.2.1 Dataset generation, data augmentation and overfitting.

In a general real-world setting only a finite dataset is available to us, for instance, obtained from observational procedures. That is, we have a set of pairs $\mathcal{D} = \{X_i, Y_i\}_{i=1}^{i=N}$ of N observations from an input-output pairs, distributed according to some distribution $F(X, Y)$:

$$(X_i, Y_i) \sim F, i = 1, ..., N. \tag{3.4}$$

Of course in general, we don't have access to the distribution of this population (i.e., the function f describing the data relation $Y = f(X)$), otherwise we would already have a perfectly accurate model, and we would not to invoke any statistical learning tool. The challenge is then to use our sample $\mathcal{D}$ to infer properties of the population. Since we don't have access to the generating function f, we cannot assess the general performance of the model on arbitrary input data. For this purpose, the training dataset $\mathcal{D}$ is typically split into two disjoint subsets $\mathcal{D} = \mathcal{D}_T \sqcup \mathcal{D}_V$, where $\mathcal{D}_T$ represents the subset of the data used for training, and $\mathcal{D}_V$ represents the subset of the data used to validate the model and assess its performance. The central idea behind this split is to be able to validate the model on data that has not been used in the training process,

allowing for an estimation of the model's generalization to the population given by:

$$\varepsilon = \mathbb{E}_F[\mathcal{L}(Y, \hat{f}(X|\mathcal{D}_T))], \tag{3.5}$$

where $\mathcal{L}$ is a loss function. The exact form in which the $\mathcal{D}$ is split needs chosen a priori, with the aim of having a representative sample of the population. A common easy-to-implement strategy is to randomly select the samples in $\mathcal{D}$ that will be part of $\mathcal{D}_V$ and distributing them according to a ratio, that is commonly taken to in the range $60-80\%$, meaning that a larger percentage of $\mathcal{D}$ is dedicated to training. In this case, an unbiased estimator of ε is

$$\hat{\varepsilon} = \frac{1}{|\mathcal{D}_V|} \sum_{i=1}^{|\mathcal{D}_V|} \mathcal{L}(Y_i, \hat{f}(X_i)), \tag{3.6}$$

where the sum runs over the validation dataset. More complex strategies that take into account the topology of the data exist and allow for an optimal training-valiation split [28]. The validation subset can also be sequentially cycled through all the available samples in a series of methods called *cross-validation* [29]. Cross-validation uses all available data in a validation stage by retraining a model on different disjoint splits of the data. This allows for a more precise evaluation of the model.

If the training subset $\mathcal{D}_T$ is not representative of the population, then the global error in equation (3.5) will be large, and the model will not be able to learn the general properties of the data. This can cause the model to overfit and learn from spurious correlation in the data and noise, and can be encouraged by having an excessive number of free parameters. Overfitting is not an intrinsic characteristic of deep-learning models, can can also be seen for instance in simple polynomial regression when the number of "independent" points exceeds the order of the fitting polynomial. Figure 3.6 illustrates the simple example of polynomial regression overfitting the training dataset as the order of the polynomial increases. Note how, by construction, the fit becomes increasingly accurate on the training data, but looses generalization power on unseen points.

The training data $\mathcal{D}_T$ can be artificially extended using *data augmentation* techniques [30] to generate new training points from existing points. Data augmentation aims at presenting the model with data that maintains intrinsic properties of the original data but varies other features that should not affect the prediction. This approach can be understood to be inspired by symmetry considerations[4] and is particularly useful in computer vision tasks. Some common data augmentation techniques for image processing

[4] A strawberry is a strawberry regardless of its orientation

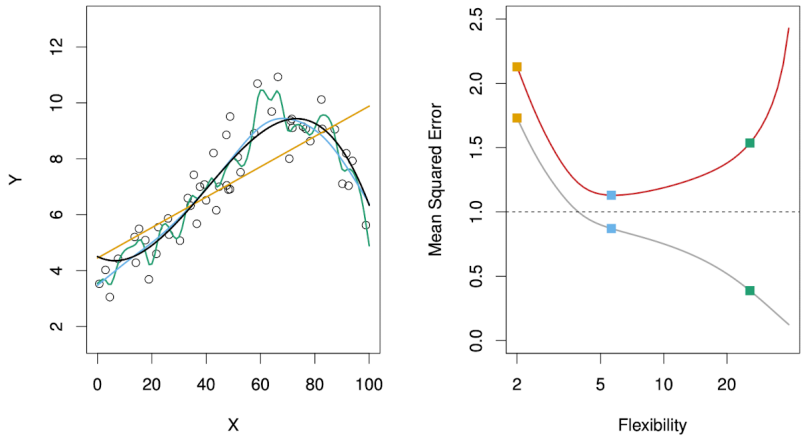


Figure 3.6: The left panel shows the data points (with added noise) generated from a function f in black. Three generative polynomial models are fitted to the data: linear regression in orange, and two smoothing splines in red and blue. The right panel shows the fitting error as a function of the polynomial degree, i.e., the flexibility of the model. The gray curve shows the error on the training dataset, which is monotonically decreasing. The red curve shows the error on the validation dataset, which initially decreases but then grows as the model overfits the training data. Figure extracted from [26].

include:

- Basic geometric transformations such as rotation, translation or flipping an image (or field) are efficiently implemented. Cropping can also be easily implemented, but changes the dimensionality of the data.

- Basic color space manipulation in colored images, including isolating a single color channel, or modifying the brightness of an image.

- Noise injection during training is particularly useful in making neural networks robust against noisy and corrupted data. Note that this can be relevant if the data we are expecting to use the trained model on has noisy (for example instrumental data from a spectrograph). The noise can be randomly drawn from independent distributions, or drawn from a noise model if we have additional insights on how to model it.

- Convolutional operations with kernels, such as the Gaussian kernel to apply a limited spatial resolution (blurring), or the Sobel kernel for related to edge detection.

3.2.2 Deep learning architecture

The network architecture defines the parametrization of the function that approximates the true underlying relation between input and output data points. In this section we briefly explore some basic building blocks used when constructing a deep learning network. A neural network (NN) ϕ is a computational graph with an input tensor X and an output tensor Y and parametrized by a set of values ω that define a functional model

$$Y = \phi(X|\omega) \equiv \phi_\omega(X). \tag{3.7}$$

For simple linear topologies a NN can be specified given a set of connected *layers* that carry out the elementary operations. These layers can be grouped in *blocks* to form subnetworks inside a NN. Three of the simplest layer architectures are *feedforward* layers (also known as *dense* layers), convolutional layers, and residual layers. Since they will be of use in our machine learning workflow, we will discuss their exact structure.

- In the simplest feedforward network, such as the one illustrated in figure 3.1a, all layers are linearly connected with an additional activation function, leading to a model of the form

$$\begin{aligned} X &= \boldsymbol{l}_0, \\ \boldsymbol{l}_i &= s_i(\boldsymbol{W}_i \boldsymbol{l}_{i-1} + \boldsymbol{b}_i) \quad \forall i \in [1, N], \\ Y &= \boldsymbol{l}_N, \end{aligned} \tag{3.8}$$

  where l_i are the layers, W_i the weights, b_i the biases, σ_i the activation functions and N the number of layers.

- Convolutional layers are especially useful in signal processing and computer vision problem. They incorporate multiple convolution operations on the input array. The parameters of the layers define the exact way the convolution is performed. Consider for illustration purposes the case of an input 2D array of size (N, N). The convolutional layer is internally parametrized by a kernel ω of shape (m, m) such that the input X to the layer is transformed as

$$Y_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} X_{i+a,j+b}. \tag{3.9}$$

  The output size will be $(N-m+1, N-m+1)$. Lastly, a user-specified non-linearity is applied. The implementation of convolutional layers in machine learning APIs

is usually done by specifying a kernel size and a set of auxiliary parameters. For instance, a single layer can have multiple independent kernels, can apply padding to the input array before convolving, etc.

- Residual layers implement a skipping connection within another layer, typically a convolutional one. A skipping connection adds the input tensor to the output tensor:

$$Y = T(X) + X, \tag{3.10}$$

where T is other neural layer. A relevant property, both from the theoretical and practical viewpoints, is that the derivative of the output tensor Y with respect X to X always include an identity term that tends to prevent it from being zero. This allows the information to flow easily in deep architectures, speeding up training and limiting overfitting [31].

- Batch-normalization layers [32] address the training difficulty encountered when the input tensors vary significantly from a sample to another. These differences cause different network weights to be adjusted simultaneously in opposite directions, which ultimately impairs training. Batch-normalization layers simply aim at normalization each tensor input feature when we feed a batch of samples to a network. A batch-normalization layer as 2 internal parameters γ, β for each input features, that are applied as a linear transformation after normalization. If we consider an input feature x (that is, a component of X), and a bath of samples $\{x_1, ..., x_m\}$, then a batch-normalization layer is implemented as:

$$\begin{aligned} \mu &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \\ \widehat{x}_i &= \frac{x_i - \mu}{\sqrt{\sigma^2}} \\ \mathbf{BN}_{\gamma,\beta}(x_i) &\equiv \gamma\widehat{x}_i + \beta \end{aligned} \tag{3.11}$$

- Max pooling layers are usually added after convolutional layers to downsample, add translational invariance and hence make to network more robust against the presence of features in different spatial positions. Similar to a convolution, max pooling is done by sliding a kernel windows onto the input array, but now we select

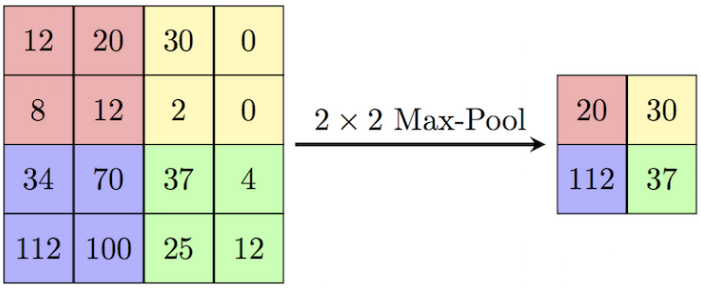


Figure 3.7: Max pooling operation with kernel size $(2, 2)$

the maximum value within the window.

3.2.3 Prediction uncertainty and Bayesian models

In classical supervised deep learning, a neural network with weights θ is trained on some dataset $\mathcal{D}$ to produce a minimal cost estimator $\hat{\theta}$ for a predefined loss function. From the statistical point of view, this is a point estimate for each one of the network parameter. Point-estimate networks might lack explainability and generalize in overconfident ways to unseen data, and there is no obvious mechanism for such models to express their ignorance in such cases.

This is an analogous situation to classical parameter inference in statistic. From this perspective, the *frequentist* approach to parameter estimation can be compared to point-estimate networks. However, *Bayesian* statistics [33] has flourished in the last decades and is becoming the dominant statistical framework for data analysis inference problems. In the Bayesian paradigm, parameters are treated as random variables to reflect our ignorance about their "true" values. Our prior beliefs about the parameters θ are then updated in the presence of new data $\mathcal{D}$ using Bayes' theorem:

$$P(H|D) \propto P(D|H)P(H), \tag{3.12}$$

where H is a certain hypothesis about θ, $P(D|H)$ is known as the *likelihood* and encodes the relation between the data generation and the parameters, and $P(H)$ reflect our prior beliefs. Bayesian frameworks have two main advantages. Firstly, they allow us to make our assumptions explicit by setting up the prior $P(H)$. This allows us to clarify, discuss and criticize prior knowledge in a clear way. Secondly, they provide a natural approach to quantify uncertainties in the inference parameters.

Bayesian neural networks are models that incorporate stochastic elements and that

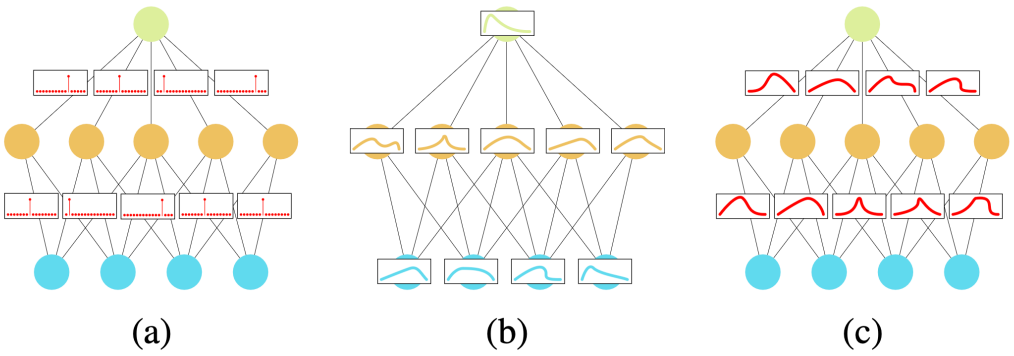


Figure 3.8: Illustration for a non-stochastic neural network a), a network with stochastic activations b), and a network with stochastic weights in c). Source: [34].

are trained using Bayesian inference techniques [34]. This is generally implemented either by considering stochastic weights in the network, or by considering stochastic activations, see figure 3.8.

A Bayesian neural network is defined by a prior distribution over the weights (when they are set to be stochastic), a functional model that forwards the inputs and generates the outputs, and a likelihood model that defines the predictive power of the network $p(y|x, \theta)$. The model weights are update in the presence of data $\mathcal{D}$ according to Bayes' formula

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}_y|\mathcal{D}_x, \theta)p(\theta) \tag{3.13}$$

When the model has seen the data, we can use the posterior distribution on the parameters to generate data or make prediction:

$$p(y|x, \mathcal{D}) = \int_\theta p(y|x, \theta)p(\theta|\mathcal{D}). \tag{3.14}$$

Note that equation (3.14) weights the predictions by each possible parametrization of the network using the posterior distribution on the weights. In that sense, Bayesian networks can be understood as training an ensemble of networks and then averaging their predictions. Note that ensembles are also a popular technique with classical neural networks [35]. In fact, since the training of a classical network has also stochastic components (the initial weights, shuffling of the data, ...), it is common to train multiple networks with different initialization seeds and weights their predictions, much tend to outperform even the best-performing network.

Let us illustrate how a committee of networks can outperform even a top-performer

network. Suppose we train 3 independent networks A, B, C for a classification task with two classes, and that they all have the same error probability p. In an exercise of pure democracy [5], consider a classifier D who picks the class with the majority of votes. Since D is wrong if and only if at most one of the classifiers is wrong, the probability of error for D is (considering $p \to 0$)

$$p^3 - 3(1-p)^2 = \mathcal{O}(p^2),$$

and hence D outperforms A, B and C.

3.2.4 Hyperparameter selection

The performance and convergence properties of a deep learning model are highly sensitive to its precise architecture, and hence, to the *hyperparameters* that control the latter: number of layers, the exact type of layers (convolutional, dense,...) and other auxiliary parameters that influence the training process. The optimal set of hyperparameters are the ones that produces the best-perfoming model on unseen data. There are three main difficulties when choosing hyperparameters. Firstly, it is not obvious a priori how modifying a certain parameter will affect the performance of a model. For example, if we add an extra layer to a network, will that improve expressivity and performance or will it lead to overfitting? This is a consequence of the low interpretability of deep learning models. Secondly, evaluating the performance of a model requires training it, which is computationally expensive. Finally, there are possibly an infinite number of possible architectures to explore.

Many Python APIs implementing routine to find optimal hyperparameters exists. They include state-of-the-art algorithms for exploring the hyperparameter space, selecting which parameters to explore, and early-stopping the evaluation of unpromising trials. In this work we use `OPTUNA` [36], which a Python optimization API, to select the appropriate set of hyperparameters for our deep learning models. `OPTUNA` implements a wide variety of searching algorithms to explore the hyperparameter space. Among those strategies, we can choose a naive grid search, where the user defines a grid over the hyperparameter space, and all possible combination are tested and ranked in a sequential order. `OPTUNA` can also implement a random search, which randomly selects the next trial over a grid. However, `OPTUNA` also implements more complex searching algorithms. The default search strategy is the *Tree-structured Parzen Estimator* (TPE) approach.

[5] "I love democracy. I love the Republic", Senator Palpatine.

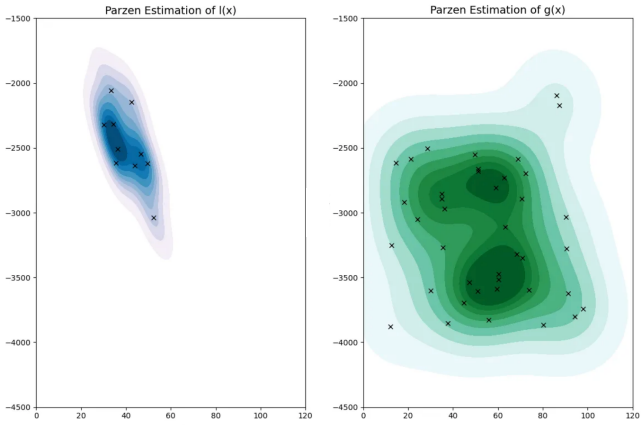


Figure 3.9: Example DE estimation for the distribution $l(x)$ and $g(x)$ used in the TPE algorithm.

TPE is designed to maximaze the expected improvement when selecting a new sample. If we have already explored a set of points x, the expected improvement (EI) for a new trial y^* is defined as

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)dy = \int_{-\infty}^{y^*} (y^* - y)p(x|y)\frac{p(y)}{p(x)}dy \quad , \tag{3.15}$$

where we are assuming a one-dimensional problem for clarity, and $p(y|x)$ represents the probability of chossing a trial y having observed x, as obtained from the searching algorithm. TPE optimizes equation (3.15) by using the following routine [37]:

1. We initiate a set $\mathcal{D}$ of explored trials with N_i parameters and compute their performance.

2. $\mathcal{D}$ is split as $\mathcal{D} = \mathcal{D}^l \cup \mathcal{D}^g$ in the top γ performers (a common value is $\gamma = 0.2$) $\mathcal{D}^l$ and the lower performers $\mathcal{D}^g$.

3. We fit a two Gaussian mixtured distributions to $\mathcal{D}^l$, l(x) and to $\mathcal{D}^g$, g(x). This is a Kernel Density Estimaton (KDE) for the distribution of both sets. See figure 3.9 for an illustration of this step.

4. The new trial parameters are selected and added to $\mathcal{D}$ to maximize $l(x)/g(x)$. In practice, this can be done by generating N_s samples from $l(x)$ and then maximizing $l(x)/g(x)$ over those samples.

We can easily show that the TPE routine maximizes the expected improvement equation (3.15). Firstly, note that $\gamma = p(y < y^*)$ and that $p(x) = \int p(x|y)p(y)dy = \gamma l(x) + (1-\gamma)g(x)$. Furthermore, if $y < y^*$, we have that $p(y|x) = l(x)$, and hence,

$$\int_{-\infty}^{y^*} (y^* - y)p(x|y)p(y)dy = \ell(x) \int_{-\infty}^{y^*} (y^* - y)p(y)dy = \gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y)dy. \quad (3.16)$$

We conclude that

$$\mathrm{EI}_{y^*}(x) \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1}, \quad (3.17)$$

and so we need to maximize $l(x)/g(x)$, as previously discussed.

3.2.5 Loss function and training

In this section, we further detail to training process of a deep learning model, and describe the alhorithms that will be used to train our model. As we have already discussed in section 3.2.1, the goal of our model is the minimize the generalization error:

$$\varepsilon = \mathbb{E}_F[\mathcal{L}(Y, \hat{f}(X|\mathcal{D}_T))]. \quad (3.18)$$

The loss function $\mathcal{L}$ should be choosen to encourage the model to produce realisitc outputs. For regression, where the goal is to estimate an output tensor Y from an input X, typical choices include the usual Euclidean norm or the absolute difference:

$$\mathcal{L}(Y, \hat{Y}) = \sum_i |\hat{Y}_i - Y_i|^p, p = 1, 2, \quad (3.19)$$

where $\hat{Y}$ is the model prediction. This choise of loss function is minimzed when the model correctly predicts the target tensor. An important information to note is that equation (3.5) is generally unkown, since we don't have acces to all the possible realizations of the data, but only to a limited training dataset. As a consequence, we typically minimize the loss function computed on small samples of the training data, called *batches*, by averaging the loss over all samples in the batch. Observe that then, computing the average loss over a large batch will yield a closer approximation to the true loss function. The training process then uses these estimations of the loss to update the network parameters, θ, using techniques that involve the computation of the loss gradient. This process is normallh repeated multiple times over the training dataset.

Each time the model sees the whole dataset is knwown as an *epoch*. A training loop with N_e epochs and batches of size B_s, with N_b batches, can then be written as:

Algorithm 1 Classical deep learning training loop

while epoch $< N_e$ **do**
  while batch $< N_b$ **do**
    Loss $\leftarrow \frac{1}{B_s}\sum_i \mathcal{L}(\hat{Y}_i, Y)$
    Compute ∇_θ(Loss)
    Update $\theta \leftarrow \theta(\nabla_\theta(\text{Loss}))$
  end while
end while

In algorithm 1 there are two main steps we have not specified. Firstly, the comoutation of ∇_θ(Loss). This is trivial from the mathematical viewpoint, but it is not trivial to implement[6]. Most APIs implement a *backpropagation* algorithm for this step that we will not discuss here[7]. The second key aspect of a training loop is using the gradient ∇_θ(Loss) to update θ. This step in algorithm 1 was repreented by $\theta \leftarrow \theta(\nabla_\theta(\text{Loss}))$. The precise way of using the calcualted loss gradients to update the weights define the *optimizer*[8]. A naive way to optize the loss at each iteration is to use the most straightforward implementation of *gradient descent*, where the parameters are updated using

$$\theta \leftarrow \theta - \alpha\nabla_\theta(\mathcal{L}), \tag{3.20}$$

where α is a constant known as the *learning rate*. In general, optimizing the loss function $\mathcal{L}$ corresponds to optimizing a complex (perhaps non-convex) function on a highly multidimensional space, with millions or billions of parameters. As a consequence, naive gradient descent might not be the best-performing optimizer. Additionally, recall the loss $\mathcal{L}$ calculated for every batch is just an approximation of the global loss 3.5. The batch size determines how accurate the computed gradient is. Calculating the loss over all possilbe data points would yield the exact gradient. On the opposite extreme, calculating the gradient on a single data point would generate a biased gradient, and lead to a noisy gradient descent. The loss function can also have a complex strucuture. In the literature, a multitude of optimizers that improve and generlize the naive gradient descent exist. In this work, we use *Adam* [16]. Adam is a first-order momemtum-based optimizer. Adam uses recursive update of the first and second moment of the gradient

[6]Note that this is purely a computer science problem.
[7]`https://www.tensorflow.org/guide/autodiff`
[8]Note that this is purely a mathematical problem in optimization.

to update the learnable parameters. The Adam algorithm reads as follows:

Algorithm 2 Adam optimizer

Require: α : Learning rate
Require: β_1, β_2: Moment decay rates
Require: $f(\theta)$: Target function
Require: θ_0: Initial parameters
Require: ε: Numerical tolerance constant
  $m_0 \leftarrow 0$: Initialize first moment
  $v_0 \leftarrow 0$: Initialize second moment
  $t \leftarrow 0$: Initialize time
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f(\theta_{t_1})$: Get gradient
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$: Update first moment
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$: Update second moment
    $m_t \leftarrow m_t / (1 - \beta_1^t)$: Correct first moment bias
    $v_t \leftarrow v_t / (1 - \beta_2^t)$: Correct second moment bias
    $\theta_t \leftarrow \theta_{t-1} - \alpha m_t / (\sqrt{v_t} + \varepsilon)$: Update parameters
  end while

Adam differs in two key features with respect to the naive gradient descent. Firstly, it includes two decay constants β_1 and β_2 used to include information about the past state of the optimization into the next current step. Since typical values are $\beta \sim 0.9$, recent gradient values contribute more to the current step update than older ones, but the inclusion of a momentum term allows the optimization to potentially escape local minima. Secondly, Adam includes a term v_t to account for the second moment in the gradient estimation. The running averages in Adam can potentially help to navigate noise functions by smoothing the local gradient.

Most current deep learning implementations are deployed or trained using HPC infrastructure. This becomes a necessity when training on large datasets. A quickly deployable way of paralellize training, and which requires minimal code modification is known as *Synchronous Distributed Training*. Synchronous Distributed Learning aims to harvest to computational power of multiple machines (GPUs,...) in the training stage by having different workers perform parallel calculations in a single batch. This can be done, for instance, by splitting the batch and sending each part to a different worker. In this work, we use the TensorFlow implementation `tf.distribute.MirroredStrategy`. This training strategy mirros the model and its parameters in every worker (for instance, in every GPUs), slices the training batch and distributes them accross all workers. Each

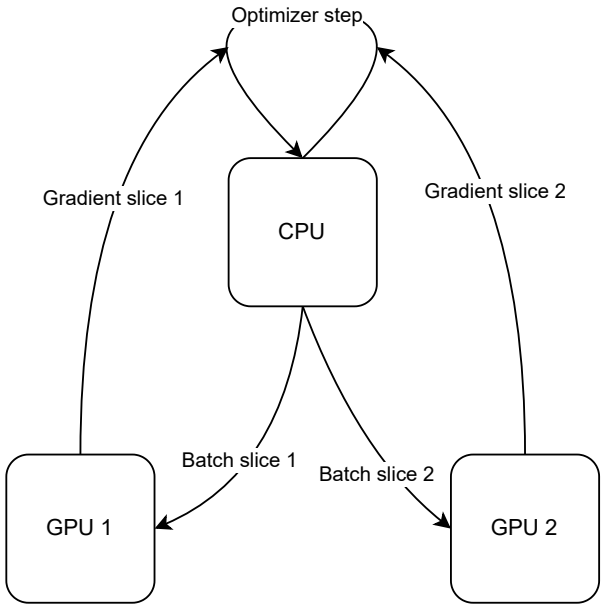


Figure 3.10: Distributed mirrored learning strategy with two workers (GPUs) and a CPU agregating the gradient and updating the model parameters.

worker then calculates the loss and gradient in the corresponding batch slices. The gradient is then aggregating before updating the parameters and moving onto the next batch.

3.3 Workflow implementation: Recovering IGM conditions from the Lyman-alpha forest

This section describes the deep learning implementation used in this work and built in the ideas that have already been introduced regarding machine learning. The code used is largely based on [38] with very minor modifications and is avaible at `https://github.com/nicenustian/bh2igm` and based on TensorFlow and TensorFlow Probability.

The ultimate aim is to use a Bayesian neural network to recover the IGM gas density from an input Lyman-α skewer. Note that this is essentially equivalent to invert Equation **??**, which describes the Lyman-α opcaity as a function of the IGM gas conditions. As we have already discussed int 2.4, we will work int the optical depth-weighted space. This avoids two main potential difficulties in the anaylisis. Firstly, by working in this

new space, the network will not have to learn the relation between peculiar velocities and the Lyman-α opcaity, which simplifies both the architecture design and learning phase. Secondly, it breaks the degeneracy introduced by peculiar velocities, since a shift in the physical space can produce the same opacity as a shift in the velocity space.

We specify the architecure design by the hyper-parameter list found in table 3.1. The global architecure that describe the number and size of the layers is specified by two hyper-parameters. Firstly, the parameter "Layers per block" is an integer list whose size is the number of blocks in the network and whose elements are the number of layers in each block. Secondly, the parameter "Filters per block" is also an integer list that specifies the number of convolutional filters of the layers for each block. If the architecure is a simple MLP, this parameter does not have any affect. The layers are choosen among a simple densly connected layer (MLP), and convolutional layer (ConvNet) or a residual layer (ResNet). Every feedforward pass through a convolutional or densely connected layer is followed by a batch normalization layer and by an activation function, see section 3.2.2. We use the PReLU activation function, wich is a generalization of the ReLU activation with a learnable weight α such that:

$$\mathrm{PReLU}(x;\alpha) = \begin{cases} \alpha x & , x < 0 \\ x & , x \geq 0 \end{cases} \tag{3.21}$$

We append the same final block to all three potential architecutres, which consists of a flattening layer transforming the tensor being manipulated to a one-dimensional vector, a dense layer a finally a Gaussian probabilistic layer. This final block includes the stochastic components of the neural network (hence the name Bayesian). For each target output density pixel the Gaussian layer outsputs a full Gaussian probability distribution parametrized by the expected density and the standar deviation. This standar deviation will later be used as the estimation for the espistemic uncertainty in the network prediction. We illustrate the fiducial architecture in figure 3.11.

We optimize the netowork hyper-parameters using `OPTUNA` as discussed in section 3.2.4. Table 3.1 the optimized hyper-parameters and the range considered during the grid search. Note that these values can potentially deppend on the nature of the training dataset, and hence may vary with redshift. In table 3.1 we present the fiducial architecture at $z = 4.4$. This optimizating process is automatically carried out whenever the redshift is varied.

For each redshift, we train a different network using the Sherwood simulation suite presented in section 2. Recall that we consider two training datasets, `SHERWOOD` and

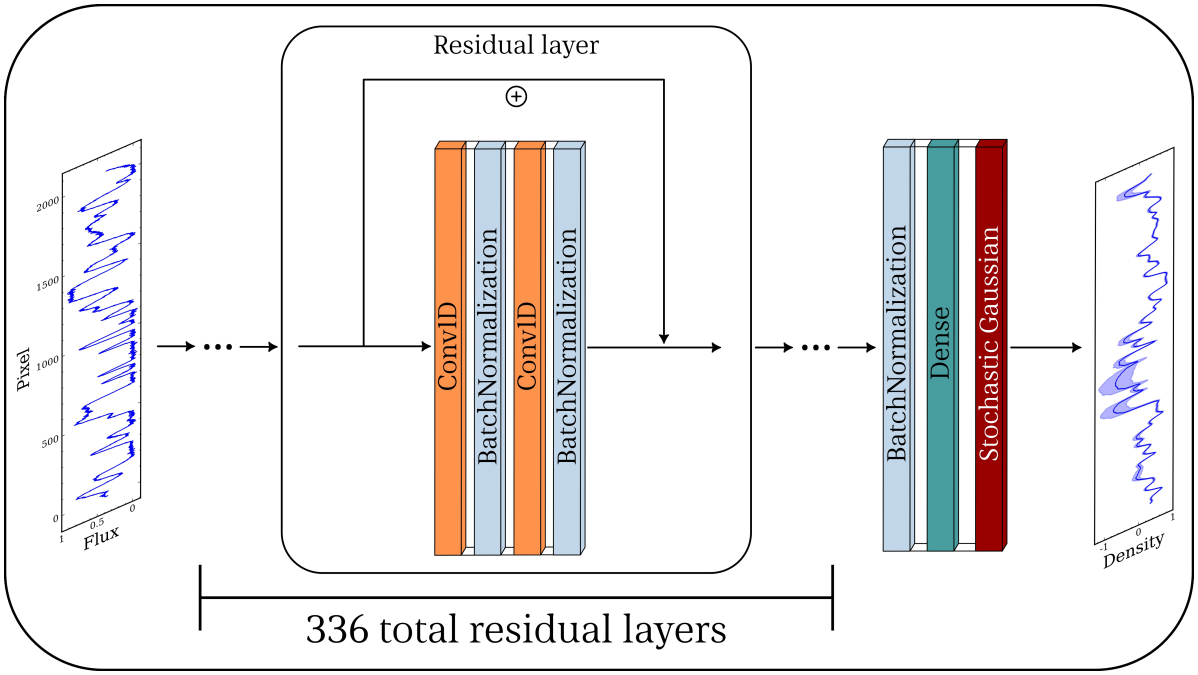


Figure 3.11: Fiducial architecture for our Bayesian neural network trained at $z = 4.4$ on the Sherwodd simulation suite. The fiducial parameters can be found in table 3.1.

`SHERWOOD THERMAL`, where the latter includes variations in the thermal parameters and reionization history. See [39] for more details. The inclusion of thermal parameters will lead to a more robust network when used on unseen real data, but its performance will naturally be lower than the network trained on a single thermal hsitory. We use `SHERWOOD` for an initial model testing, and `SHERWOOD THERMAL` for our final anaylisis on real data. 80 % of the data is used for training, and the rest is used for validation purposes. For reference, the training dataset `SHERWOOD` with,7 different WDM models has a total size of $\sim$ 3.5 GB. We use the Raven[9] HPC cluster at the Max Planck Computing and Data Facility to train our networks. With 4 Nvidia A100 GPUs, a typical training time is $\sim$ 1 hour, depending on the model's exact architecture and number of parameters.

The network's input consist of a Lyman-α flux skewer, and the network's output consists of the mean density and standard deviation at each pixel. The labelled training data consists of individual Lyman-α flux skewers with their associated density optical depth-weighted density field, $(F, \log(\Delta_\tau))$. For each labelled training pair, the loss function is taken to be the negative log-likelihood (NLL) for the normal distribution

[9] `https://www.mpcdf.mpg.de/services/supercomputing/raven`

Hyper-parameter	Min.	Max.	Best value
Layers per block (int)	1	4	[1, 2, 4 ,4]
$\log_2$(Filters per block) (int)	1	5	[4, 5, 5, 5]
Number of blocks (int)	1	4	4
$\log_2$(Batch size) (int)	3	8	3
Learning rate ($\log_s$, float)	10^{-4}	0.1	0.004937
Network (MLP, ConvNet, ResNet)	...	...	ResNet

Table 3.1: Hyper-parameter grid search for the fiducial model at $z = 4.4$. "$\log_s$" indicates the parameter is sampled in the log domain. "Int" and "float" mean they are sampled as integers or floats, respectively.

that the network parametrizes:

$$-\log \mathcal{L} = \frac{1}{N} \sum_{\mathrm{i}} \left((Y_{\mathrm{i}} - Y_{\mathrm{i,pred.}})^2 / \sigma^2_{\mathrm{i,pred.}} + \log \left(\frac{1}{\sigma^2_{\mathrm{i,pred.}}} \right) \right), \tag{3.22}$$

where the sum runs over all skewer pixels,Y_i is the real density at pixel i, $Y_{\mathrm{i,pred.}}$ is the predicted expected density and $\sigma^2_{\mathrm{i,pred.}}$ the predicted standard deviation.

The input skewers are processed as follows. Firstly, the input flux is rebinned into a target number of pixels to match the real data that will be used. Downsampling is done by taking the avarage flux over nearby pixels, while upsampling is done by appending to every pixel a copy of itself. The flux is then convolved using a gaussian kernel to simulate a given instrumental resolution. In most of our analysis the resilution is taken to be 6 km/s, in accordance with state-of-the-art spectrographs, such as UVES [40]. During training, the training data is stacked and randomly shuffled to ensure a correct mixing and representativity of the models. Lastly, we process the reescale input optical depth to match the observed mean flux at the considered redshift [41]. The training data is augmented on-the-fly in two ways. We first roll the input spectra by application translations, this helps the network learning dynamical features, independently of their positions. We also add random uncorrelated gaussian noise to simulate a finate signal-to-noise ratio (SNR). The default SNR per pixel used for testing purpises is 50. When applying our methods to real-data, the network is retrained using a noise model for each target object.

We use the Adam optimizer with fixed moment decay rates as implementation in the TensorFlow API and a learning rate set by the Optuna grid search. The training is evaluated using two main metrics: the loss function NLL, and the mean absolute error, MAE, defined as the Euclidean norm 3.19 in with $p = 1$. To prevent over-fitting, the

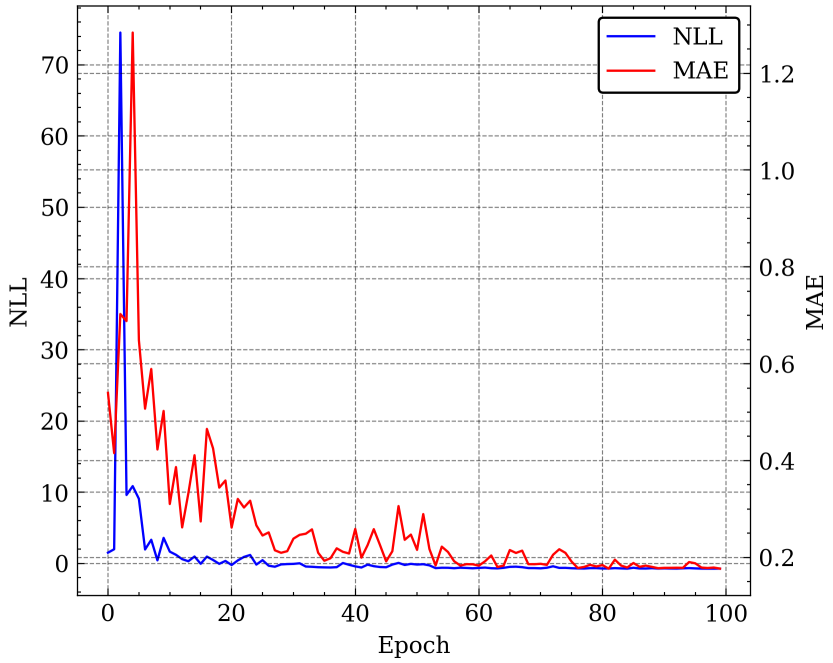


Figure 3.12: Learning curve for our fiducial architecture at $z = 4.4$ on the SHERWOOD dataset. The figure shows the NLL and the MAE on the validation split as a function of the epoch.

network parameters are only updated if the NLL loss improves on the validation split of the data. A policy to halve to learning rate if there is no loss improvement in the test set for 10 epochs is also included. This is the most straightforward adaptation of an adaptive learning rate. Figure 3.12 shows the evolution of the NLL and the MAE on the validation split during training for our fiducial architecture at $z = 4.4$ and the SHERWOOD suite. It is interesting to note that while the MAE is always positive by definition, negative valueso of the NLL are consistent with its definition. Figure 3.12 demostrates that, for our problem, the network training coverges in $\sim 100 - 150$ epochs. Recall that we always work with $\log(\Delta_\tau)$, which also serves as a regulatization step with respecto to the density.

In figure 3.13 we show an example $20\mathrm{h}^{-1}$cMpc Lyman-α validation skewer from the SHERWOOD dataset. The top panel shows the input flux to the network. The bottom panel shows the true Δ_τ density fields, the (mean) recovered densities and the 1σ envelope predicted by the Bayesian network. Note that, in the spectral regions with large features and variations in the flux, the predicted mean density closely follows the true field. In those regions, there is enough physical information for the network to accurately recover Δ_τ. In contrast, in the saturated regions with low flux, the noise dominates,

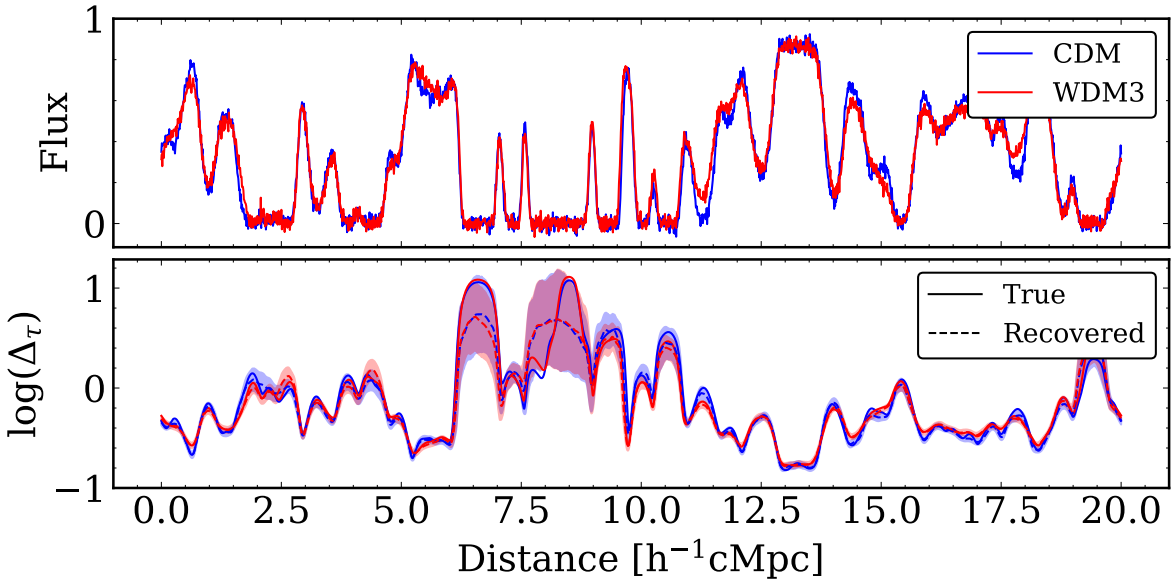


Figure 3.13: An example $20h^{-1}$cMpc Lyman-α validation skewer for the CDM and WDM3 Sherwood models. The top panel shows the input flux to the network. The bottom panel shows the true Δ_τ density fields, the (mean) recovered densities and the 1σ envelope predicted by the Bayesian network.

and the network predicts larger uncertainties (observe, for instance, the saturated region in Figure 3.13 around $8h^{-1}$cMpc). This should be regarded as a strength of Bayesian netoworks, since they are able to accurately detect and make explicit situations where the predictions should not be trusted. To minimize the mean error in regions where the network cannot make accurate preditions, note how there is a bias towards quasi-constant mean prediction. This is visible around $8h^{-1}$cMpc, where the true density has a steep increasing profile, and the predition has an almost constant u-shaped profile.

On the `SHERWOOD` validation split, the network reaches a 1σ accuracy of 79%, meaning that 79% of the pixels are correctly predicted within 1σ. Note that this is a larger accuracy that expected from purely normally-distributed densities.

,.... violin...compare thermal models and non thermal

3.4 Recovered field statistics and uncertainties

ADD bootstrapping and PDF OF PDF ADD MATRIX CORR and resampling and recoveries using finite skerwers

3.5 Model interpretability and limitations

Deep learning models tend to have, by definition, numerous parameters. As a consequence, giving an interpretation for individual model parameters is far from being a trivial task. On top of the large number of parameters, the nonlinearities and the potentially biased and uncomplete dataset can lead to complex training behaviors. In that regard, deep learning models have classically been regarded as "black-box" models. They are often more accurate than simpler statistical models, but lack explainability. Great efforts have been recently made in undertanding the learning dynamics of neural networks [42], [43]. Due to the complexity of this interpretation task, here we choose to only give a qualitative analysis of some aspects that can help gain intuition on how the newtork operate internally.

Although many open-source libraries, such as *Captum*[10], implement popular methods for deep learning model visualization and interpretation, in this work we use TensorFlow's built-in'differentiation capabilities. In particular, we use Automatic Differentiation to compute the *saliency* score of the model, defined as the gradient of the model output with respecto to the model input. As a consequence, for every target density pixel, the saliency score measure which flux pixel variation contributes the most to a change in density. Saliency is a simple socre giving us insights into how the model uses flux to recunstruct densities. Additionally, note that calculating such gradients do not requeire any numerical finite diffrence approximation, since TensorFlow's GradientTape class can build an exact computational graph with all the operations performed on the input flux. The saliency at density pixel i and input flux pixel j is simply

$$\text{Saliency}(i) = \frac{\partial \mu_i}{\partial f_j}. \tag{3.23}$$

As we have already mentioned when discussing Figure 3.13, saturated regions lead to larger uncertainties in the network's predictions, reflecting the fact that noise dominates over the physical signal. Observe again the saturated region around 7h^{-1}cMp and 8h^{-1}cMp in Figure 3.13. Both of these regions have completly different density profiles, but the network predicts the same u-shape profile with a similar peak density. This peak density is slighlt above the mean density in the simulation box and is similar for the CDM and WDM3 models. We can interpret this as the network leaning a unique mean "high-density" value for saturated regions and the whole dataset.

talk about the no train tests weird inputs filters number, values, activation. saliency

[10]https://github.com/pytorch/captum

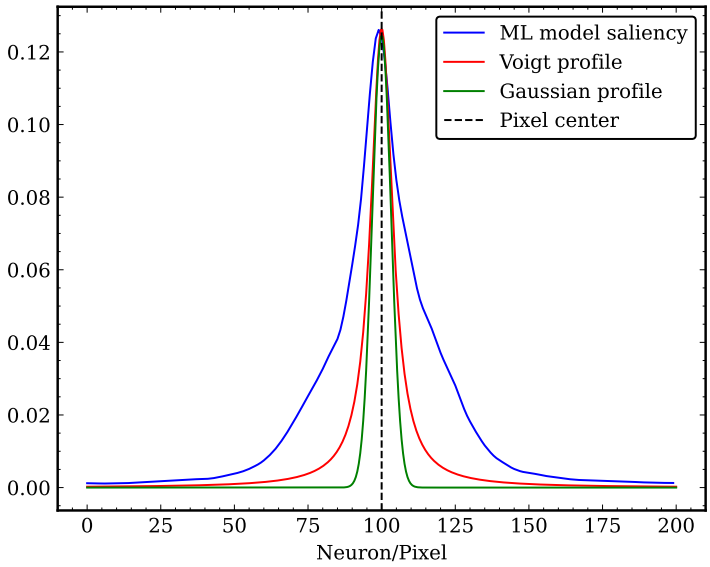


Figure 3.14: Saliency of the CDM and WDM3 Sherwood network at $8h^{-1}$cMpc Lyman-α flux.

4 Constraining Warm Dark Matter at the density level

4.1 Inference pipeline: from Lyman-alpha skewers to WDM constraints

In section 3 we have given a detrailed analysis of our Bayesian deep-learning algorithm to recover IGM densties from a Lyman-α skewer. The baryonic density of the IGM is sensitive to the WDM mass through a clear physical mechanism related to gravitational clustering. In this section, we use the recovered IGM density fields to constrain WDM candidates. Note that the natural observable quantity related to the Lyman-α forest is the flux. As a consequence, an almost omnipresent choice in the literature has been to work directly with summary quantities on the flux, which is a proxy of the underlying density. Such summaries include the power spectrum (PS) [6], the curvature [44], the probability distribution function (PDF), etc. The deep-learning approach introduced in section 3 allows us to directly recover the baryonic density field along a line of sight, thus having full acces to the field level IGM properties. In this section, we use the recovered Δ_τ fields by our neureal network to constraint WDM directly at the density field level. Recall Figure 1.2 and Figure 2.1 showing how different WDM models affect the density field. We strive to capture that difference in the WDM models to constrain which free-streaming length are compatible with QSO observations. Note that for a given line of sight, the precise value of the density field not only depends on the WDM masses (and other possible physical paramters), but most crucially it depends on the random density fluctuation that have seeded the gravitational collapse process. Equivalently, in a simulation setting, the obtained densities would depend on the seed used to initiliaze each simulation. This makes infeasible to campare a given Lyman-α skewer to a simulated one, and means that we must use aggregated summaries over multiple skewer that capture the global properties of the field. In this work, we perform the inference use

the density PDF as the summary statistic of choice. This is a well-tested and robuat statistic [45]. In section 4.4.4 we give an additional argument, based on Information Maximising Neural Networks, to support this choice of summary statistic.

The basic working principle of inference pipeline is to fit the observed Δ_τ PDF, with its associated uncertainties, to the corresponding Δ_τ PDF produced by each WDM model. To compare similar quantities, we always work with the recovered field by our neural network from section 3. Note that in the `SHERWOOD` only a finite number of DM models are avaible, due to the computational cost of running this simulation. In more detail, we only have acces to the models CDM, and WDM1,2,3,4,8,12. We smoothly and linaerly interpolate the PDF by interpolating each PDF bin to generate a Δ_τ PDF in the range of WDM masses from 0 to 1 KeV. Note that, since we expect the real observations to fall close to the CDM model, and have multiple simulations close to CDM, we expect this interpolation not to limit the inference pipeline. See Figure 2.1 again and observe how similar are the PDF for CDM and WDM3. For more massive models in `SHERWOOD` the PDF converge to the CDM PDF.

For each DM model of inverse mass m we denote by PDF(m) the Δ_τ PDF computed over the recovered densities by our NN network over all avaible sightlines in the `SHERWOOD` or `SHERWOOD THERMAL` datasets. We refer to this as the model PDF. Let us denote by $\widehat{\text{PDF}}$ the recovered PDF for a target set of (observed) skewers. Then, we fit $\hat{\text{PDF}}$ to the model PDFs using a simple χ^2 fit:

$$\chi^2(m) = \sum_i \frac{(\text{PDF}(m)_i - \widehat{\text{PDF}}_i)^2}{(\sigma_i)^2}, \tag{4.1}$$

where the index i refers to each PDF bin and σ_i are the uncertainties on the observed data. If the data is normally independently and normally distributed, the quantity in Equation 4.1 follows a χ^2 distribution [46]. The model that minimises the quantity is the best-fit model, on which we can compute uncertainties and obtain a confidence region of compatible models with the observed data. Since we are only fitting a single parameter model, the 1 and 2-sigma confiderence regions on the WDM mass are given, respectively, by the boundaries of

$$\chi^2(m) - \chi^2_{\text{min}} = 1, 4, \tag{4.2}$$

where χ^2_{min} is the best-fit χ^2 value. In the following, we will be interested in the 2σ confidence regions. This region can be interpreted as the set of WDM models that guarantee

to contain the "true" model with a 95% probability. In the current literature, WDM constraints are often reported as the 2σ upper limit, where the lower limit typically corresponds to CDM. The current more stringent 2σ WDM limit constraints are ~ 3 KeV, see [6] and [39]. Note that this fitting procedure is non-Bayesian, in the sense that we don't include any prior knowledge or use Bayes' theorem. Again, this procedure is compared in section 4.4.4 to a IMNN Bayesian fit, leading to similar results.

4.2 Inference testing on Sherwood spectra under realistic observational conditions

In this first section we run our inference pipeline from section 4 using a set of toy observed skewers. More precisely, we use our neural netowork trained on different subsets of the `SHERWOOD` dataset and use validation `SHERWOOD` skewers as the "observed" skewers.

4.2.1 Untrained DM models

We begin by testing the robustness and inter(extra)polation capabilities of the neural networks by considering the `NOTRAIN` models that are trained on data that iteratively excludes each one of the WDM models. For each one of those trained neural networks, for instance `NOTRAINWDM4` (which was not trained on WDM4), we predict on the WDM4 sightlines, compute the recovered Δ_τ with its uncertainties according to 3.4 and run the inference pipeline. We also perform additional variations by running the pipeline only on the PDF bins whose value is greater than a fixed constant. This has the effecto of only fitting the peak of the PDF, and neglecting the low-information tails. Figure **??** summarises this inference tests. Each plot corresponds to a different fit combining predictions from each `NOTRAIN` model with each of the masks applied to the PDF when fitting. The light and dark blue regions correspond to the 1 and 2 sigma confidence regions. The red line corresponds to the true DM model mass, and the black line to the best-fit model that minimises the χ^2. The blue curve is the χ^2 metric. Note that we are using all 5000 sightlines on the inference step. This is not a realistic sample size, but rather a test to the inter(extra)polation of the pipeline.

Recall that on WDM2,3,4 the models are interpolating. Observe that as a consequence, the recovered mass is consistently recovered within the 1σ region. In contrast, with the models CDM and WDM1, the neural networks have to extrapolate on unseen DM models. As expected, the recovered model mass might not even be included in

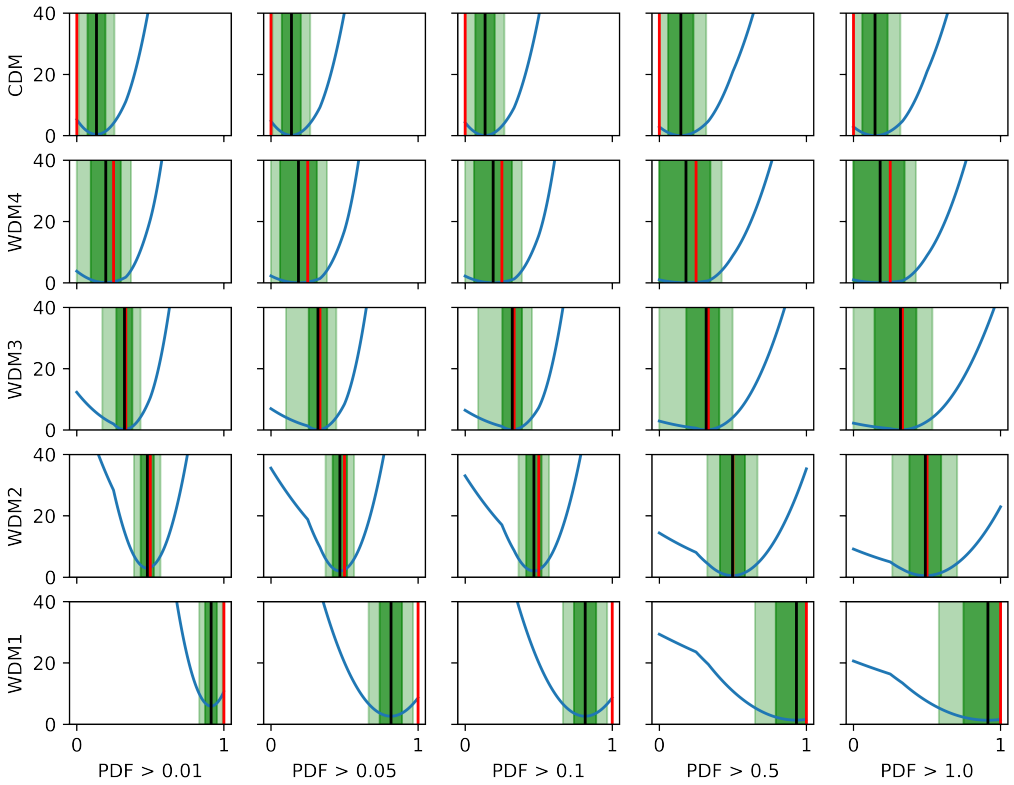


Figure 4.1: Inference results on the `NOTRAIN` neural netoworks. Each plot corresponds to a different fit combining predictions from each `NOTRAIN` model with each of the masks applied to the PDF when fitting. The light and dark blue regions correspond to the 1 and 2 sigma confidence regions. The red line corresponds to the true DM model mass, and the black line to the best-fit model that minimises the χ^2. The blue curve is the χ^2 metric. Note that we are using all 5000 sightlines on the inference step.

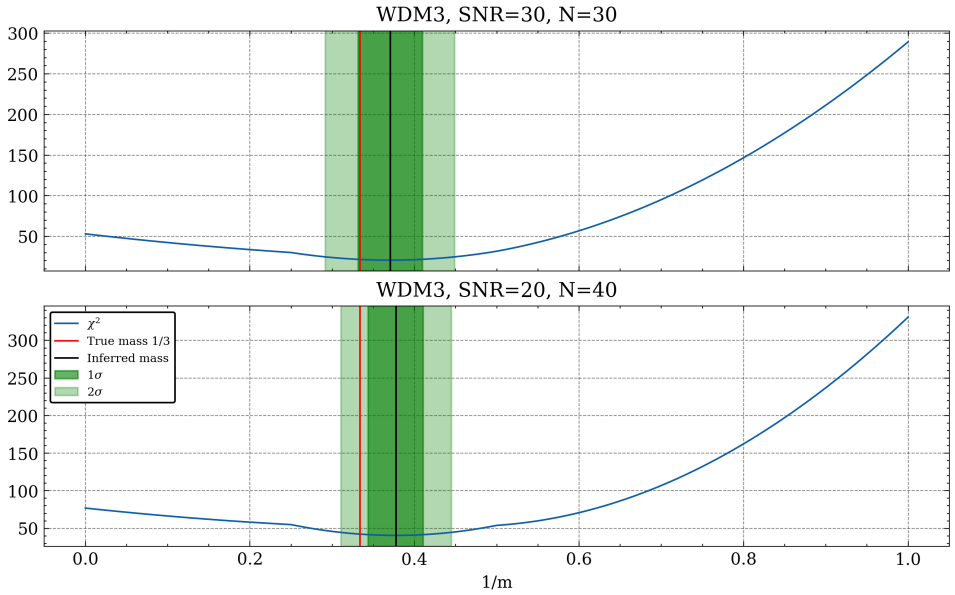


Figure 4.2: Inference predictions on WDM3 for different combinations of SNR and number of targets.

the 2σ regions, meaning that the pipeline fails to correctly recover the true mass if we interpolate models. This does not affect our prediction with real data, since, as we have already mentioned, current WDM constraints favour a lower mass limir of ~ 3 KeV. Lastly, observe in Figure 4.1 that the mask applied on the horizontal axis does not significantly affect the recovered masses.

4.2.2 Realistic UVES observational conditions

In this section, we explore the effect of realistic observational conditions, such as the number of observed quasars, the signal-to-noise ratio (SNR, or the instrumental resolution, in the infered DM constraints. For that purpose, we use typical parameters for the Ultraviolet and Visual Echelle Spectrograph (UVES) on the European Southern Observatory's Very Large Telescope [47]. We consider a spectral resolution of 6 km/s per pixel, variable SNR in the range $20-30$ and a variable number of targets in the range $30-40$. Note that the skewers in the `SHERWOOD` dataset are $20\mathrm{h}^{-1}$cMpc in length, while the spectral range in the UVES instrument expands multiple times that range. In particular, since measurements can extand up to redshifts differences $\Delta z \sim 1$, we assume that each observed spectrum can be descomposed into ~ 15 of our `SHERWOOD` skewers.

A common compromise in an observational program with a fixed observational time

is between number of targets and exposure time per target, which determines the SNR. In Figure 4.2 we show how prioritising SNR or the number of targets affects to infered WDM masses on the WDM3 model. In general, we find that increasing the target number leads to slighly tighter confidence regions, while increasing the SNR leads to more accurate constraints.

4.3 Inference on alternative hydrodynamical codes

4.4 Comparison of the inference pipeline against Information Maximising Neural Networks

The inference pipeline presented so far in this section is based on a simple χ^2 fit of the Δ_τ recovered PDFs from our fiducial NN model. This pipeline rellies on a series of contingent choices, most notably the use of the density PDFs as the summary statistics of the fields. In this section, we explore, in an agnostic way, the possiblity of using other summaries different from the Δ_τ PDFs to perform the inference (note that other summaries such as the density power spectrum, curvature,... could potentially be used). More concretely, we will introduce Information Maximising Neural Networks (IMNNs) and use them to perform the inference within a Bayesian framework. We then compare the results of this procedure with the inerence pipeline discussed in section 4.

4.4.1 Information Maximising Neural Networks

Information Maximising Neural Networks aim at obtaining optimal summaries of data [48]. Neural networks are used to parametrise these summaries in an agnostic way by maximising the information of the summaries with respect to the model parameters of interest.

Consider a data-generating procedure depending on some model parameters θ, generating data realization $d_i(\theta)$ where i labels a realization or initial seed of the simulation. We want to obtain a function $f\colon d \mapsto x$ that maps each simulation to a summary vector of the same size as θ. This is, essentially, a compression algorithm. IMNNs work by transforming the original likelihood of the data, which is a priori not known, into the gaussian form

$$-2\ln\mathcal{L}\left(x|\theta\right) = \left(x-\mu\left(\theta\right)\right)^T C^{-1}\left(x-\mu\left(\theta\right)\right), \tag{4.3}$$

where C is the covariance matrix of the calculated summaries with a set of n_s simulations, and μ the summary mean depending on the model parameters. The information of the observed summaries with respect to θ is then the Fisher information matrix [49]:

$$F_{\alpha\beta} = -\,\mathrm{E}\left[\frac{\partial^2}{\partial\theta_\alpha\partial\theta_\beta}\log\mathcal{L}(x;\theta)\middle|\theta\right] = \frac{\partial\mu}{\partial\theta_\alpha}C^{-1}\frac{\partial\mu}{\partial\theta_\beta}, \tag{4.4}$$

whose determinant we note as $|F|$. The goal is to obtain summaries that maximise the Fisher information while mainting a minimum covariance condition to generate independent summaries. The summaries produced by the network can then be used to perform inference on them. Since the Fisher information is a quantity that depends on the model paramters θ, the quantity in Equation 4.4 needs to be evaluated at some fiducial model parameters in order to obtain a numerical result.

IMNNs have been succesfully leveraged in the IGM community. Recent papers have explored the possibility of using them to perform IGM thermal paramter inference from Lyman-α skewers, see [50] for instance, where authors find IMNNs to yield tighter and more robust constraints than classical Markov Chain Monte Carlo approaches. Despite these promising results, many challenges arise when using IMNNs on real data, primarly related to the correct identification and interpretation of model parameters.

4.4.2 IMNN training and non-linear summaries

In this section we consider a simple MLP architecture with linear layers followed by PReLU(α) activation functions and a dropout layer that randomly (with probability p) sets to 0 the any layer weight during each epoch to prevent over-fitting. The network takes as input a simulated data vector d and prodcued a summary vector x of the same size and the parameter vector θ. We use the Adam optimiser to maximise |F| by minimising the following loss function

$$\mathcal{L}_{IMNN} = -\log(|F|) + \lambda\frac{\mathcal{N}}{\mathcal{N}+\exp(-\mathcal{N})} * \mathcal{N}, \tag{4.5}$$

where $\mathcal{N} = ||C - I|| + |C^{-1} - I|||$ measure the deviation from independt summaries and λ is a coupling constant. In Equation 4.5, the second term sets a scale for the Fisher information by producing summaries whose covariance approaches the identity matrix. Once this is achieved, the term containing the exponential factor vanishes and the network will maximise $|F|$. Note that there is not a unique set of potential optimal sumamries. In fact, any bijective function of a sufficient statistic for a certian likelihood

is also a sufficient statistic.

For each paramter update in the training procedure, we generate a batch of data at the fidicual parameters θ_f. The derivatives in Equation 4.4 are numerically approximated with finite differences by running simulations at parameters $\theta_f \pm \Delta\theta_\alpha$, where $\Delta\theta_\alpha$ are a small parameter variation, and then calcualting

$$\frac{\partial x}{\partial \theta_\alpha} = \frac{x(d(\theta_f + \Delta\theta_\alpha)) - x(d(\theta_f - \Delta\theta_\alpha))}{2\Delta\theta_\alpha}. \tag{4.6}$$

We then calculate C, the covariance of the summaries at fiducal paramters, and use it to compute the Fisher information in Equation 4.4 and the loss function in Equation 4.5. Note that, since the covariance matrix and the derivative in the Fisher information matrix are computed at the data summaries, they implicitely depend on the NN parameters.

4.4.3 Summarising a Gaussian signal

We implement IMNNs using Pytorch[1], a deep-learning Python framework. We test the implementation first by exploring its behavior on a toy model, where we generate ramdon samples from a Gaussian distribtion $\mathcal{N}(\mu, \sigma)$. The sufficient statistic for the model parameters $\theta = (\mu, \sigma)$ are, in this case, the sample mean and standard deviation:

$$\hat{\mu} = \frac{1}{n_d}\sum d_i \qquad \hat{\sigma}^2 = \frac{1}{n_d - 1}\sum (d_i - \hat{\mu})^2. \tag{4.7}$$

Note that the statistic for σ is non-linear. For this example, we select fiducial parameters $\theta_f = (\mu = 0, \sigma = 1)$ and $\Delta\theta = (0.1, 0.1)$ and generate random fields with 100 pixels. In total, 5000 fields are generated for each parameter set, including a validation dataset. Note that testing the network performance in the validation dataset is crucial in performing early-stopping during the training process. Indeed, since the training dataset is limited, it will contain spurious correlations that the network will use to infer a higher information than expected. By stopping the training when the network information on the validation set saturates, we can avoid this problem. We use a simple architecture, with layers $[128, 128, 128, 2]$, learning rate of 0.001, dropout rate of $p = 0.5$, and batch size of 500. Observe the training evolution in Figure 4.3, where we show $|F|$ and $||C - I|| + ||C + I||$ as a function of the epoch for the training and validation sets. As can be seen, the validation information quickly saturates in ~ 100 epochs, and then slowly

[1] `https://pytorch.org`

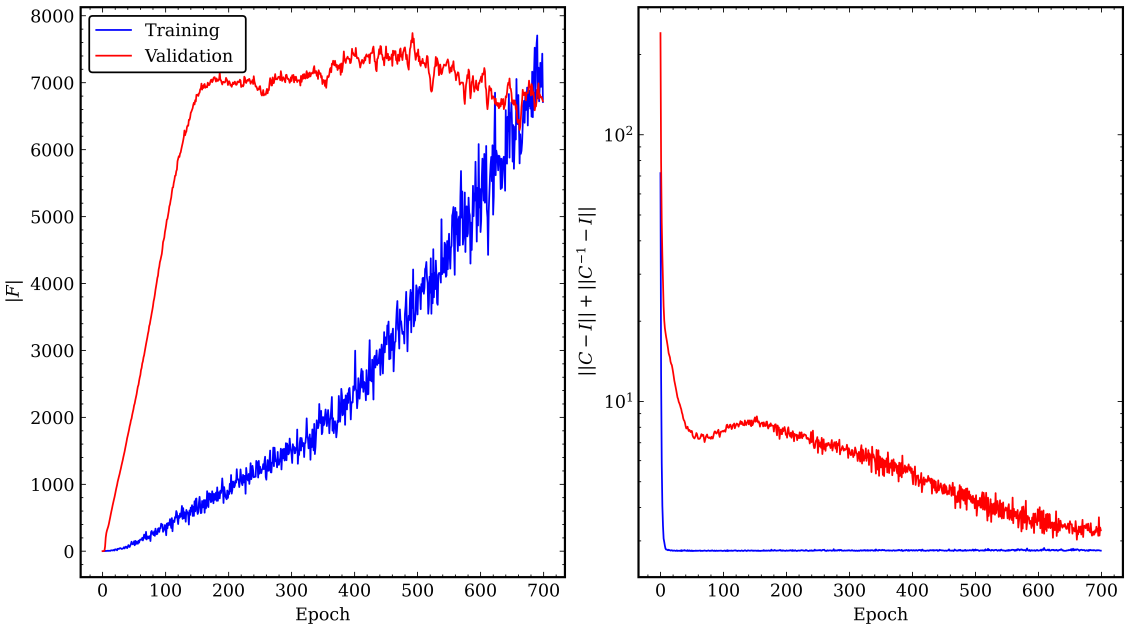


Figure 4.3: $|F|$ and $||C - I|| + ||C + I||$ as a function of the epoch for the training and validation sets during the training of the IMNN on a normal field. The goal is to find summaries to optimally extract information about the mean and variance of the field.

decreases as the network over-fits.

To better interpret the network output and to understand its behavior, we generate samples of the same size with the zero mean but a standar deviation randomly sampled from $(0, 12)$. We then compute the exact statistic (the sample standar deviation) and plot it against the second IMNN sumamry output. The result is Figure 4.4. Observe that the exact statistic in the x-axis is highly correlated to the network summary in the y-axis. Since the relation between the two quantities is clearly bijective, the model has succesfully learnt to extract all the possible information for the field covariance. The natural scatter is due to having a simple NN model.

Note that the network has not seen any normal field with such variances $\sigma \in (0, 12)$ during training, yet it is able to extract the correct summary. This is of crucial importance, since it means that we could use the IMNN summaries to do inference on a field with parameter values slighly different from the fiducial ones used during training.

To conclude the exploration of this toy model, we use the network to perform Bayesian inference. We implement a simple Approximate Bayesian Computation (ABC) [51] algorithm that obtains approximate posterior samples from a given set of prior samples and observed data. As the observed data, we take 50 Gaussian fields simulated at

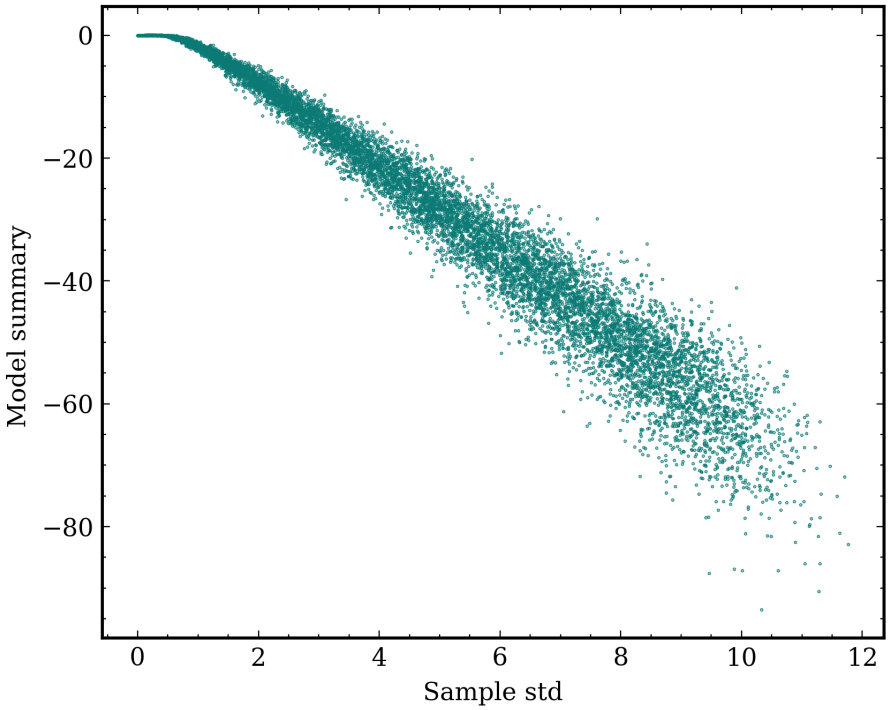


Figure 4.4: The IMNN summary plotted agains the exact sufficient statistic for the standard deviation using multiple samples with $\sigma \in (0, 12)$.

fiducial parameter ($\mu = 0, \sigma = 1$) values. As priors, we take 5000 samples from a non-informative uniform distribution in $(-5, 5)$ for μ and $(0, 10)$ for σ. The ABC rejection algorithm is described in Algorithm 3.

In Figure 4.5 we show the posterior samples and Gaussian Kernel Density Estimation (KDE) for the distributions of μ and σ. The dashed vertical lines show the true parameter values. As expected, and even with a non-informative prior, the IMNN summary contain sufficient information to produce tight posterior around the true model parameters. Note that the posterior scatter on the non-linear summary σ is larger than on the linear summary μ.

4.4.4 IMNN inference results on WDM masses

We can now deploy a simple IMNN as an alternative way of constraining WDM models. We follow section 4.4.3 and consider a similar architecture but now with 4 dense layers of size $[512, 512, 256, 2]$. As input to the NN, we consider Lyman-α flux skewers. Since in flux space the skewers has many simulation-specific and prominnent features that can be picked-up by a NN, we work in Fourier space. More precisely, the input to the

Algorithm 3 Approximate Bayesian Computation Rejection Algorithm

Input: Observed data $\mathbf{y}$, threshold ϵ, number of simulations N, prior distribution $\pi(\theta)$
Output: Accepted parameter values $\{\theta_i\}_{i=1}^{M}$
Initialize $M \leftarrow 0$
for $i = 1$ to N **do**
    Sample θ^* from the prior distribution $\pi(\theta)$
    Simulate data $\mathbf{y}^*$ from the model using θ^*
    if $d(\mathbf{y}, \mathbf{y}^*) \leq \epsilon$ **then**
        Accept θ^*: $\theta_{M+1} \leftarrow \theta^*$
        Increment $M \leftarrow M + 1$
    end if
end for
return $\{\theta_i\}_{i=1}^{M}$

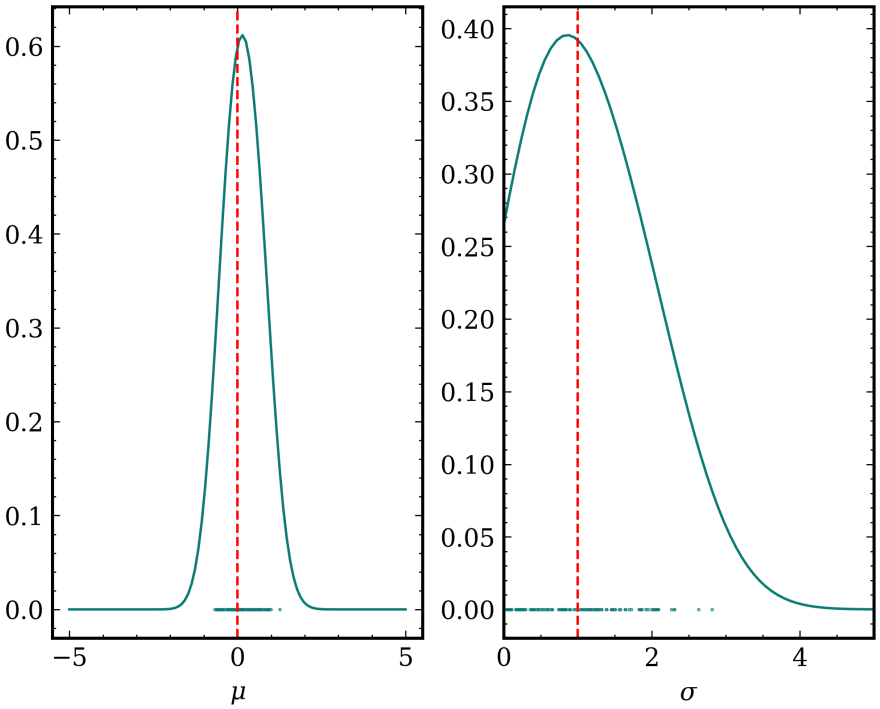


Figure 4.5: Posterior samples and KDE for the ABC rejection algorithm applied to the normal toy model, where we infer the mean and variance of a Gaussian field with flat priors and the summaries output of a IMNN. The dashed vertical lines show the true parameter values.

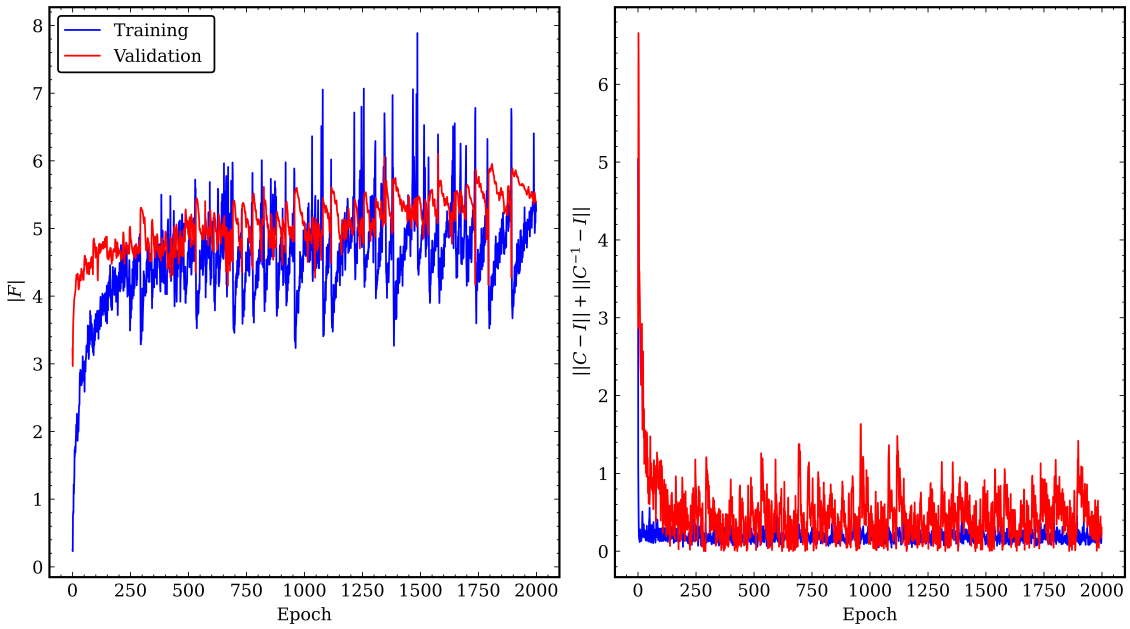


Figure 4.6: $|F|$ and $||C - I|| + ||C + I||$ as a function of the epoch for the training and validation sets during the training of the IMNN on the SHERWOOD dataset Lyman-α skewers.

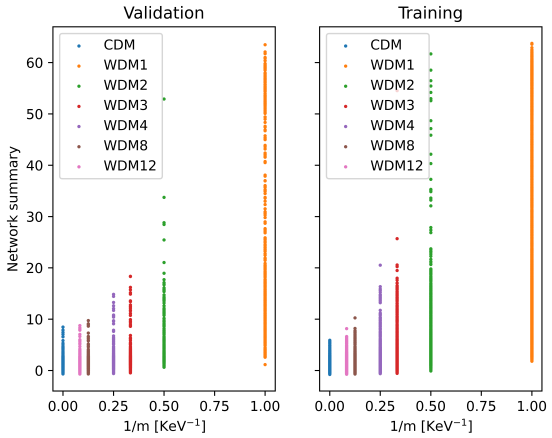
network are

$$\sqrt{k}|\delta_F(k)|, \tag{4.8}$$

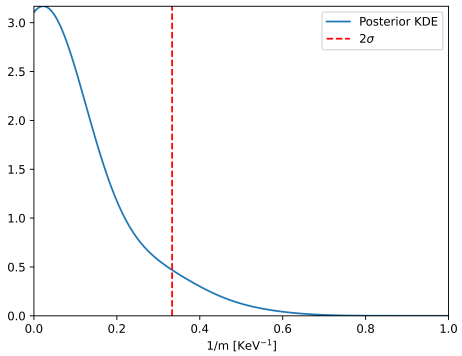
where δ_F is the flux contrast of the skewer. We use the SHERWOOD simulation suite with varied WDM mass to train the IMNN. Note that this means that we are assuming that WDM is the only model parameter affecting the Lyman-α forest property. We ignore thermal parameters variations for this demosntration. We train our model on the fiducial CDM mass corresponding to 0 Kev^{-1}, and use the WDM3 model corresponding to 0 Kev^{-1} to calculate the summary derivatives. The choice of WDM3 is due to the flux skewers showing sufficient variation with respect to CDM.

In Figure 4.6 we show the training progress of the IMNN as a function of the epoch. The information extracted on the validation split quickly saturates at ~ 250 epochs. It is clear that the network is able to learn a map from Lyman-α skewers in Fourier space into a one-dimensional parameter space. Since the SHERWOOD suite has a fix number of simulations, interpreting the network output summaries is a challenging task.

In Figure 4.7a) we show the summaries of the trained IMNN for all the aviable SHERWOOD Lyman-α skewers is Fourier space. Observe how the summaries show a large scatter for every model, corresponding to the large simulation variability within each



(a) IMNN outputs for the SHERWOOD Lyman-α skewers.



(b) ABC WDM mass posterior for a test of 50 observed CDM skewers. The 2σ WDM constraint is ~ 3 Kev.

Figure 4.7

skewer. However, the mean summaries show a clear bijective trend, manifesting that the IMNN has learnt an informative summary. Note that this can be interpreted as the necessity of a large number of observed samples when constraining WDM models. We use the IMNN summaries to perform an inference test and obtain a Bayesian posterior as follows. First, we select 50 CDM skewers from the validation dataset and obtain their corresponding summaries by passing them through the IMNN. We now use all validation skewers from the SHERWOOD suite and obtain their summaries. We use the ABC rejection algorithm 3 to generate to posterior distribution in Figure 4.7b). The 2σ limit for the WDM mass is ~ 3 KeV. Recall that this is a comparable constraint to the one obtained in section 4.2. We interpret this not only as a robustness sign of our original pipeline involving a χ^2 fit the Δ_τ PDFs, but most notably as a sign that it optimally extracts the majority of the information of the Lyman-α skewers with respect to the WDM mass parameter.

PDF:

$$p(\lambda|f(x),N) \propto \frac{\Gamma(N+1)}{\Gamma(N\lambda+1)\Gamma(N-N\lambda+1)} f(x)^{N\lambda}(1-f(x))^{N(1-\lambda)} \tag{4.9}$$

4.5 WDM constraints from SQUAD DR1 observational data

5 Conclusions

Bibliography

[1] Sean M. Carroll. *Spacetime and Geometry: An Introduction to General Relativity*. Cambridge University Press, 2019.

[2] Planck Collaboration, P. A. R. Ade, N. Aghanim, C. Armitage-Caplan, M. Arnaud, et al. "Planck 2013 results. XVI. Cosmological parameters". In: *APP* 571, A16 (Nov. 2014), A16. DOI: 10.1051/0004-6361/201321591. arXiv: 1303.5076 [astro-ph.CO].

[3] James E. Gunn and Bruce A. Peterson. "On the Density of Neutral Hydrogen in Intergalactic Space." In: *APJ* 142 (Nov. 1965), pp. 1633–1636. DOI: 10.1086/148444.

[4] Robert H. Becker, Xiaohui Fan, Richard L. White, Michael A. Strauss, Vijay K. Narayanan, et al. "Evidence for Reionization at z$\sim$ 6: Detection of a Gunn-Peterson Trough in a $z = 6.28$ Quasar". In: *The Astronomical Journal* 122.6 (Dec. 2001), pp. 2850–2857. ISSN: 0004-6256. DOI: 10.1086/324231. URL: http://dx.doi.org/10.1086/324231.

[5] Ian D. McGreer, Andrei Mesinger, and Valentina D'Odorico. "Model-independent evidence in favour of an end to reionization by z $\approx$ 6". In: *MNRAS* 447.1 (Feb. 2015), pp. 499–505. DOI: 10.1093/mnras/stu2449. arXiv: 1411.5375 [astro-ph.CO].

[6] Bruno Villasenor, Brant Robertson, Piero Madau, and Evan Schneider. "New constraints on warm dark matter from the Lyman-alpha forest power spectrum". In: *Physical Review D* 108.2 (July 2023). ISSN: 2470-0029. DOI: 10.1103/physrevd.108.023502. URL: http://dx.doi.org/10.1103/PhysRevD.108.023502.

[7] Elisa Boera, George D. Becker, James S. Bolton, and Fahad Nasir. "Revealing Reionization with the Thermal History of the Intergalactic Medium: New Constraints from the Ly-alpha Flux Power Spectrum". In: *The Astrophysical Journal* 872.1 (Feb. 2019), p. 101. ISSN: 1538-4357. DOI: 10.3847/1538-4357/aafee4. URL: http://dx.doi.org/10.3847/1538-4357/aafee4.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: http://dx.doi.org/10.1038/nature14539.

[9] Xia Zhao, Limin Wang, Yufei Zhang, Xuming Han, Muhammet Deveci, and Milan Parmar. "A review of convolutional neural networks in computer vision". In: *Artificial Intelligence Review* 57.4 (Mar. 2024). ISSN: 1573-7462. DOI: 10.1007/s10462-024-10721-6. URL: http://dx.doi.org/10.1007/s10462-024-10721-6.

[10] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. "A review on the long short-term memory model". In: *Artificial Intelligence Review* 53.8 (May 2020), pp. 5929–5955. ISSN: 1573-7462. DOI: 10.1007/s10462-020-09838-1. URL: http://dx.doi.org/10.1007/s10462-020-09838-1.

[11] Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. "Minimum Width for Universal Approximation". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=O-XJwyoIF-k.

[12] R.R. Schaller. "Moore's law: past, present and future". In: *IEEE Spectrum* 34.6 (1997), pp. 52–59. DOI: 10.1109/6.591665.

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[14] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. 2019. arXiv: 1907.10701 [cs.LG].

[15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: http://dx.doi.org/10.1038/323533a0.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[17] A. Vallenari, A. G. A. Brown, T. Prusti, J. H. J. de Bruijne, F. Arenou, et al. "GaiaData Release 3: Summary of the content and survey properties". In: *Astronomy and Astrophysics* 674 (June 2023), A1. ISSN: 1432-0746. DOI: 10.1051/0004-6361/202243940. URL: http://dx.doi.org/10.1051/0004-6361/202243940.

[18] Dean Richardson, Robert L. Jenkins III, John Wright, and Larry Maddox. "ABSOLUTE-MAGNITUDE DISTRIBUTIONS OF SUPERNOVAE". In: *The Astronomical Journal* 147.5 (Apr. 2014), p. 118. ISSN: 1538-3881. DOI: 10.1088/0004-6256/147/5/118. URL: http://dx.doi.org/10.1088/0004-6256/147/5/118.

[19] Romina Ahumada, Carlos Allende Prieto, Andrés Almeida, Friedrich Anders, Scott F. Anderson, et al. "The 16th Data Release of the Sloan Digital Sky Surveys: First Release from the APOGEE-2 Southern Survey and Full Release of eBOSS Spectra". In: *APJS* 249.1, 3 (July 2020), p. 3. DOI: 10.3847/1538-4365/ab929e. arXiv: 1912.02905 [astro-ph.GA].

[20] Jordi Miralda-Escude. "Reionization of the Intergalactic Medium and the Damping Wing of the Gunn-Peterson Trough". In: *The Astrophysical Journal* 501.1 (July 1998), pp. 15–22. ISSN: 1538-4357. DOI: 10.1086/305799. URL: http://dx.doi.org/10.1086/305799.

[21] Bradley Greig, Sarah E. I. Bosman, Frederick B. Davies, Dominika Ďurovčíková, Hassan Fathivavsari, et al. *Blind QSO reconstruction challenge: Exploring methods to reconstruct the Lyα emission line of QSOs*. 2024. arXiv: 2404.01556 [astro-ph.CO].

[22] Sarah E I Bosman, Dominika Ďurovčíková, Frederick B Davies, and Anna-Christina Eilers. "A comparison of quasar emission reconstruction techniques for z > 5.0 Lyman-alpha and Lyman-beta transmission". In: *Monthly Notices of the Royal Astronomical Society* 503.2 (Feb. 2021), pp. 2077–2096. ISSN: 1365-2966. DOI: 10.1093/mnras/stab572. URL: http://dx.doi.org/10.1093/mnras/stab572.

[23] Bin Liu and Rongmon Bordoloi. "A deep learning approach to quasar continuum prediction". In: *Monthly Notices of the Royal Astronomical Society* 502.3 (Jan. 2021), pp. 3510–3532. ISSN: 1365-2966. DOI: 10.1093/mnras/stab177. URL: http://dx.doi.org/10.1093/mnras/stab177.

[24] Jonah C. Rose, Paul Torrey, Francisco Villaescusa-Navarro, Mark Vogelsberger, Stephanie O'Neil, Mikhail V. Medvedev, Ryan Low, Rakshak Adhikari, and Daniel Angles-Alcazar. *Inferring Warm Dark Matter Masses with Deep Learning*. 2023. arXiv: 2304.14432 [astro-ph.CO].

[25] Parth Nayak, Michael Walther, Daniel Gruen, and Sreyas Adiraju. *LyαNNA: A Deep Learning Field-level Inference Machine for the Lyman-α Forest*. 2023. arXiv: 2311.02167 [astro-ph.CO].

[26] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer US, 2021. ISBN: 9781071614181. DOI: 10.1007/978-1-0716-1418-1. URL: http://dx.doi.org/10.1007/978-1-0716-1418-1.

[27] Shaeke Salman and Xiuwen Liu. *Overfitting Mechanism and Avoidance in Deep Neural Networks*. 2019. arXiv: 1901.06566 [cs.LG].

[28] V. Roshan Joseph and Akhil Vakayil. "SPlit: An Optimal Method for Data Splitting". In: *Technometrics* 64.2 (June 2021), pp. 166–176. ISSN: 1537-2723. DOI: 10.1080/00401706.2021.1921037. URL: http://dx.doi.org/10.1080/00401706.2021.1921037.

[29] Michael W Browne. "Cross-Validation Methods". In: *Journal of Mathematical Psychology* 44.1 (Mar. 2000), pp. 108–132. ISSN: 0022-2496. DOI: 10.1006/jmps.1999.1279. URL: http://dx.doi.org/10.1006/jmps.1999.1279.

[30] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 2019). ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: http://dx.doi.org/10.1186/s40537-019-0197-0.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[32] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[33] Udo von Toussaint. "Bayesian inference in physics". In: *Rev. Mod. Phys.* 83 (3 2011), pp. 943–999. DOI: 10.1103/RevModPhys.83.943. URL: https://link.aps.org/doi/10.1103/RevModPhys.83.943.

[34] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. "Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users". In: *IEEE Computational Intelligence Magazine* 17.2 (May 2022), pp. 29–48. ISSN: 1556-6048. DOI: 10.1109/mci.2022.3155327. URL: http://dx.doi.org/10.1109/MCI.2022.3155327.

[35] Thomas G. Dietterich. "Ensemble Methods in Machine Learning". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000, pp. 1–15. ISBN: 9783540450146. DOI: 10.1007/3-540-45014-9_1. URL: http://dx.doi.org/10.1007/3-540-45014-9_1.

[36] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[37] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, pp. 2546–2554. ISBN: 9781618395993.

[38] Fahad Nasir, Prakash Gaikwad, Frederick B. Davies, James S. Bolton, Ewald Puchwein, and Sarah E. I. Bosman. *Deep Learning the Intergalactic Medium using Lyman-alpha Forest at* $4 \leq z \leq 5$. 2024. arXiv: 2404.05794 [astro-ph.CO].

[39] Vid Iršič, Matteo Viel, Martin G. Haehnelt, James S. Bolton, Margherita Molaro, et al. "Unveiling dark matter free streaming at the smallest scales with the high redshift Lyman-alpha forest". In: *PRD* 109.4, 043511 (Feb. 2024), p. 043511. DOI: 10.1103/PhysRevD.109.043511. arXiv: 2309.04533 [astro-ph.CO].

[40] Sandro D'Odorico, Stefano Cristiani, Hans Dekker, Vanessa Hill, Andreas Kaufer, Taesun Kim, and Francesca Primas. "Performance of UVES, the echelle spectrograph for the ESO VLT and highlights of the first observations of stars and quasars". In: *Discoveries and Research Prospects from 8- to 10-Meter-Class Telescopes*. Ed. by Jacqueline Bergeron. Vol. 4005. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. June 2000, pp. 121–130. DOI: 10.1117/12.390133.

[41] George D. Becker, Paul C. Hewett, Gábor Worseck, and J. Xavier Prochaska. "A refined measurement of the mean transmitted flux in the Ly-alpha forest over $2 < z < 5$ using composite quasar spectra". In: *Monthly Notices of the Royal Astronomical Society* 430.3 (Feb. 2013), pp. 2067–2081. ISSN: 1365-2966. DOI: 10.1093/mnras/stt031. URL: http://dx.doi.org/10.1093/mnras/stt031.

[42] Ravid Shwartz-Ziv and Naftali Tishby. *Opening the Black Box of Deep Neural Networks via Information*. 2017. arXiv: 1703.00810 [cs.LG].

[43] Vanessa Buhrmester, David Münch, and Michael Arens. *Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey*. 2019. arXiv: 1911.12116 [cs.AI].

[44] George D. Becker, James S. Bolton, Martin G. Haehnelt, and Wallace L. W. Sargent. "Detection of extended HeII reionization in the temperature evolution of the intergalactic medium: IGM temperatures over $2 < z < 5$". In: *Monthly Notices of the Royal Astronomical Society* 410.2 (Nov. 2010), pp. 1096–1112. ISSN: 0035-8711. DOI: 10.1111/j.1365-2966.2010.17507.x. URL: http://dx.doi.org/10.1111/j.1365-2966.2010.17507.x.

[45] Prakash Gaikwad, Raghunathan Srianand, Martin G Haehnelt, and Tirthankar Roy Choudhury. "A consistent and robust measurement of the thermal state of the IGM at $2 < z < 4$ from a large sample of Ly-alpha forest spectra: evidence for late and rapid HeIIreionization". In: *Monthly Notices of the Royal Astronomical Society* 506.3 (July 2021), pp. 4389–4412. ISSN: 1365-2966. DOI: 10.1093/mnras/stab2017. URL: http://dx.doi.org/10.1093/mnras/stab2017.

[46] William H. Press, Brian P Flannery, Saul A Teukolsky, and William T Vetterling. *Numerical Recipes in C book set: Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. Cambridge, England: Cambridge University Press, Oct. 1992.

[47] Michael T Murphy, Glenn G Kacprzak, Giulia A D Savorgnan, and Robert F Carswell. "The UVES Spectral Quasar Absorption Database (SQUAD) data release 1: the first 10 million seconds". In: *Monthly Notices of the Royal Astronomical Society* 482.3 (Oct. 2018), pp. 3458–3479. ISSN: 1365-2966. DOI: 10.1093/mnras/sty2834. URL: http://dx.doi.org/10.1093/mnras/sty2834.

[48] Tom Charnock, Guilhem Lavaux, and Benjamin D. Wandelt. "Automatic physical inference with information maximizing neural networks". In: *Physical Review D* 97.8 (Apr. 2018). ISSN: 2470-0029. DOI: 10.1103/physrevd.97.083004. URL: http://dx.doi.org/10.1103/PhysRevD.97.083004.

[49] Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul Grasman, and Eric-Jan Wagenmakers. *A Tutorial on Fisher Information*. 2017. arXiv: 1705.01064 [math.ST]. URL: https://arxiv.org/abs/1705.01064.

[50] Soumak Maitra, Stefano Cristiani, Matteo Viel, Roberto Trotta, and Guido Cupani. *Parameter estimation from Lyα forest in Fourier space using Information Maximising Neural Network*. 2024. arXiv: 2404.04327 [astro-ph.CO]. URL: https://arxiv.org/abs/2404.04327.

[51] Clara Grazian and Yanan Fan. *A review of Approximate Bayesian Computation methods via density estimation: inference for simulator-models*. 2019. arXiv: 1909.02736 [stat.CO]. URL: https://arxiv.org/abs/1909.02736.