universe

MDPI

Review

Observational Cosmology with Artificial Neural Networks

Juan de Dios Rojas Olvera [1], Isidro Gómez-Vargas [2] and Jose Alberto Vázquez [2,*]

[1] Facultad de Ciencias, Universidad Nacional Autónoma de México, Ciudad de Mexico 04510, Mexico; juan_97dd_@ciencias.unam.mx
[2] Instituto de Ciencias Físicas, Universidad Nacional Autónoma de México, Cuernavaca 62210, Mexico; igomez@icf.unam.mx
* Correspondence: javazquez@icf.unam.mx

Abstract: In cosmology, the analysis of observational evidence is very important when testing theoretical models of the Universe. Artificial neural networks are powerful and versatile computational tools for data modelling and have recently been considered in the analysis of cosmological data. The main goal of this paper is to provide an introduction to artificial neural networks and to describe some of their applications to cosmology. We present an overview on the fundamentals of neural networks and their technical details. Through three examples, we show their capabilities in the modelling of cosmological data, numerical tasks (saving computational time), and the classification of stellar objects. Artificial neural networks offer interesting qualities that make them viable alternatives for data analysis in cosmological research.

Keywords: cosmological parameters; cosmology; machine learning

Citation: de Dios Rojas Olvera, J.; Gómez-Vargas, I.; Vázquez, J.A. Observational Cosmology with Artificial Neural Networks. *Universe* **2022**, *8*, 120. https://doi.org/10.3390/universe8020120

Academic Editors: Antonino Del Popolo, Yi-Fu Cai and Jean-Michel Alimi

Received: 23 December 2021
Accepted: 4 February 2022
Published: 12 February 2022

1. Introduction

Observational cosmology has three relevant scientific pillars: cosmological theory, astronomical observations and statistical methods for data analysis. These areas allow us to compare theoretical cosmology with the observational evidence and thus validate or discard cosmological models. As the amount of observational data increases, the choice of data analysis methods becomes more crucial. Therefore, computational methods, including machine learning, have been incorporated into the cosmological field in recent years [1–3].

Machine learning is a branch of artificial intelligence focused on the mathematical modelling of the data. In recent times, the most growing field of machine learning is deep learning, which focuses on computational models called artificial neural networks (ANNs). In several respects, neural networks have been proven to have more flexibility and computational power than other machine learning methods, so they have been successfully applied in a large number of fields, ranging from industry and medicine to education and science, to name a few.

There are several types of artificial neural network: some of them fall under the category of supervised learning and others under unsupervised learning. Hence, the problems that deep learning is able to solve include: regression, classification, pattern recognition, generative processes, and time series, among many others. The ability of the ANNs to deal with complex and large datasets has allowed them to be good alternatives in several observational cosmological works [4,5], where theoretical models can be highly nonlinear, numerical methods be computationally expensive, and there may be very complex datasets. Artificial neural networks have been used in several cosmological applications, such as N-body simulations [6,7], image analysis [8,9], and statistical methods [10–12], and they have also been used to perform non-parametric reconstructions of cosmological functions [13–16]. In addition, ANNs have decreased the computational time taken for cosmological calculations [17–21]. Furthermore, neural networks have made it possible to analyse CMB signals [22,23], and to classify observational measurements from extensive

surveys—for example, quasars in the Sloan Digital Sky Survey (SDSS) [24]. Finally, the use of deep learning in cosmology has increased considerably in recent years, and several works have already incorporated this type of research, i.e., [25–28].

The main goal of this paper is to present an introduction to deep learning followed by some examples of neural network applications to cosmology. In Section 2 we present an overview of neural networks with the basic concepts. Section 3 contains the cosmological background necessary for the subsequent examples. In Sections 4–6 we show three applications of the ANNs in cosmology. First, in Section 4 we present a reconstruction of the Hubble parameter. Second, an ANN is trained from the solutions of the dynamical system of the Universe and its content (Section 5). Third, an example of stellar object classification is shown in Section 6. Finally, in Section 7 we provide some final comments and conclusions.

2. A Deep Learning Overview

The initial steps of artificial neural networks started in 1943 when the first computational logic model for learning neurons was introduced [29]. However, this model was only capable of solving a few linearly separable logic gates and required plenty of information about the problem. Some years later, in 1957, Frank Rosenblatt [30] proposed the closest precursor to modern neural networks, the computational model known as the *perceptron*.

The perceptron proposed changes in the learning process, and unlike its predecessor, the neuron was now able to learn by itself from the input signals. However, in 1969 Marvin Minsky and Seymour Papert, famous artificial intelligence researchers, highlighted the inability of the perceptron to solve some non-linear problems, such as the well-known classification problem called *exclusive OR* (XOR) [31]. Due to this, the scientific and technological community lost interest in neural networks, and it took many years to change this situation. This stagnation stage is known as the *artificial intelligence winter*. The curiosity in neural networks was regained in the 1980s, driven by Geoffrey Hinton and colleagues, who presented the *backpropagation* algorithm that solved some of the computational limitations of neural networks [32]. Nevertheless, it was not until the 21st century, due to computing advancements, that neural networks regained great relevance, and a new scientific discipline was born: *deep learning*, focused exclusively on the study of artificial neural networks.

Nowadays there are several types of neural networks—for example, the recurrent networks that are widely used for time series, the convolutional neural nets that are very successful in image processing, the autoencoders used in image denoising, and the modern generative adversarial networks. In this work, we focus on the most basic deep neural network, which is known as the multilayer perceptron (MLP).

2.1. The Perceptron

The *perceptron* is considered the fundamental unit of neural networks, which consists of a mathematical model for a biological neuron. Neurons in the brain receive information from neighbouring cells, process it, and then, through synapses, transmit an electrical or chemical signal to other neurons. The perceptron performs a similar task. It takes as input n signals, i.e., $x_1, x_2, \cdots, x_n$, each of them associated with a coefficient w_j, called a *weight*. Then, the perceptron computes a linear combination of weights and inputs, known as the *weighted sum* z:

$$z = \sum_{j=1}^{n} (x_j w_j) + b, \tag{1}$$

where b is called *bias* and indicates the output value when all of the x_i are null. The weights determine the strength of influence of each input on the neural network model.

As in biological neurons, a modulation of the signal z is needed. In an ANN, that modulation is carried out through the *activation functions*, whose characteristics are described in Appendix A. The output of the perceptron is given by the activation function (denoted by σ) applied to the weighted sum:

$$a = \sigma(z). \tag{2}$$

For a schematic description, see Figure 1. This computational structure is commonly called a *node* or *neuron*, which acts as a brick for deep neural networks (multiple arrays of nodes).

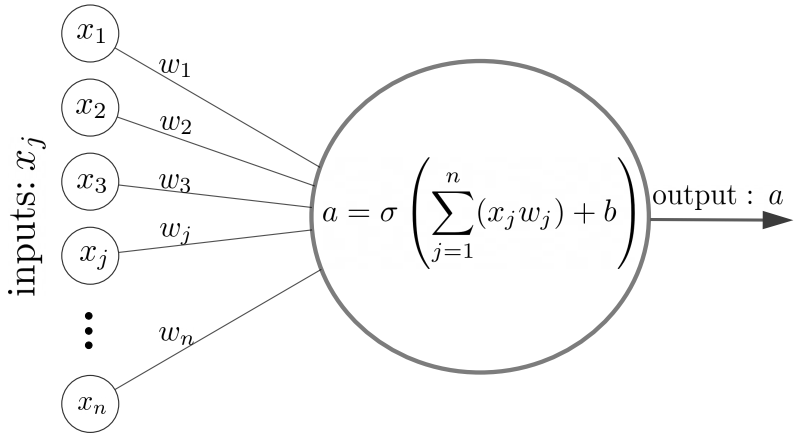


Figure 1. The perceptron processes the inputs and regularises them with an activation function, generating a numerical output.

Algorithm 1 shows the learning rule for updating the weights that the perceptron follows, given a dataset made of $x_i \in X$ independent variables and $y_i \in Y$ dependent variables. For the learning process, the output of the perceptron should be compared with the target value included in the original dataset. This can be done with an error function called a *cost function*, which in this case is the mean squared error between the target y_i and the prediction a_i. The *learning rate* η indicates how much the weights change (see Appendix B). The iteration steps in the updating process are known as *epochs*.

Algorithm 1: The perceptron rule.

Data: $X = \{x_1, ..., x_n\}$, $Y = \{y_1, ..., y_n\}$.
Generate random weights w.
for *i in range*(epochs) **do**
  for $x_i \in X$ and $y_i \in Y$ **do**
    - Compute an output a_i.
    - Update the weights:
      $w_i \longleftarrow w_i - \Delta w_i$,
      where $\Delta w_i = \eta(a_i - y_i)\sigma'(z)x_i$,
      - η: learning rate.
      - y_i: true value of $f(x_i)$.
      - a_i: perceptron prediction.
      - σ': activation function derivative.
  end
end

2.2. Deep Neural Networks

The learning mechanism for a single neuron can be generalised to many of them with diverse arrangements. The simplest deep neural network is known as the *multilayer perceptron* (MLP) or *feedforward neural network*, which consists of several perceptrons or nodes interconnected with each other through different arrangements of neurons called *layers*. Every multilayer perceptron has an input layer; at least one intermediate layer, often called *hidden layers*; and one output layer (see Figure 2). The input layer of an ANN corresponds to the data to be processed through the subsequent layers, and the number

of nodes in this input layer must match the number of independent variables (features or attributes) of the dataset. Similarly to the perceptron, the output layer has to be compared with the dependent variables (labels or classes) of the dataset, which are the predicted ones.

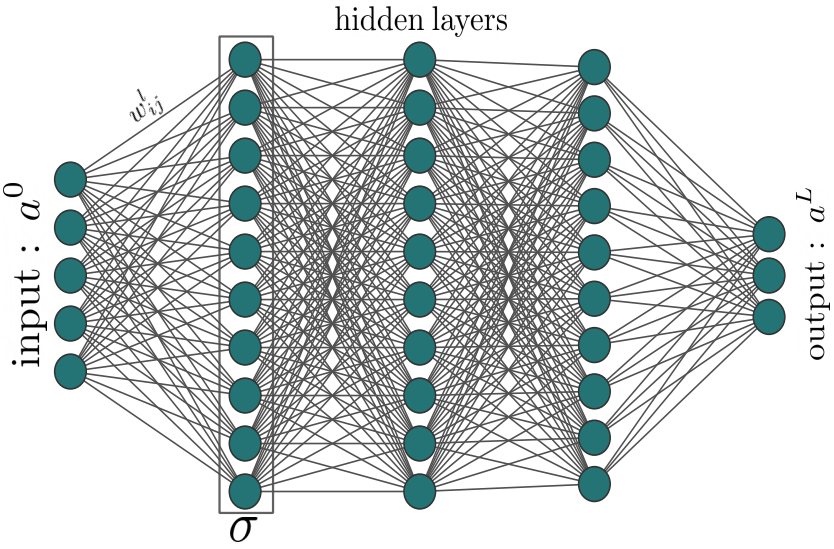


Figure 2. Structure of a deep neural network (or multilayer perceptron) with three hidden layers. Each neuron in the input layer receives a feature (or attribute) from the original dataset. The output layer generates a prediction that is compared, using a loss function, to the dependent variable within the original dataset.

The connections between one node to another have an associated weight w_{ij}^l and a bias b_j^l, where l represents the layer number. The weight connects the node i in the layer $l-1$, with the node j in the layer l. In consequence, the node j in the layer l will have an output a_j^l. The weighted sum of each node contains information on the weights, biases, and inputs of each layer, and is calculated as follows:

$$z_j^l = \sum_i a_i^{l-1} w_{ij}^l + b_j^l. \tag{3}$$

Then, the activation functions σ are applied to the weighted sum, Equation (3), and are denoted by:

$$a_j^l = \sigma(z_j^l). \tag{4}$$

Using the matrix notation allows us to establish the structure of the MLP in a simpler way. Let L be last layer. Then the MLP output is the vector that takes each node a_j^L as an entry:

$$\begin{pmatrix} a_1^L \\ a_2^L \\ \vdots \\ a_m^L \end{pmatrix} = a^L. \tag{5}$$

Thus, we can call intermediate layers recursively as vectors a^l. For each layer, we can also build a weight matrix W^l and denote the output z^l of every layer as follows:

$$z^l = a^{l-1} W^l + b^l. \tag{6}$$

Notice that the inputs of the associated weight matrix, with layer number l, are defined as:

$$[W^l]_{ij} = w_{ij}^l,$$

with b^l being the vector that contains the biases of each node in the l layer. Afterwards, it is necessary to apply the activation function σ, as indicated in Equation (4):

$$a^l = \sigma(z^l). \tag{7}$$

The process by which the information is transmitted, from the input layer of the neural network to the generation of an output or prediction in the last layer, is known as *forward propagation*.

A multilayer perceptron can be considered as a function $f : a^0 \in \mathbb{R}^n \longrightarrow a^L \in \mathbb{R}^k$. In this function, the hidden layers must also be considered even without being expressly mentioned. The forward propagation of a MLP is represented in the following diagram:

$$\begin{aligned} a^0 \longrightarrow W^1 a^0 + b^1 \xrightarrow{\sigma} a^1 \longrightarrow \cdots \\ \longrightarrow a^{l-1} \longrightarrow W^l a^{l-1} + b^l \xrightarrow{\sigma} a^L. \end{aligned} \tag{8}$$

For example, let us assume the structure of a neural network consists of a single hidden layer with h nodes and a non-linear activation function σ. Using Equations (6) and (7), we can calculate the hidden layer weighted sum z^1 as follows:

$$z^1 = a^0 W^1 + b^1.$$

The weight matrix W^1 is constructed by taking as a column j the vector of weights that enter the node j; thus, in this case $W^1 \in \mathrm{M}_{n\times h}$ and $b^1 \in \mathbb{R}^h$. Now, we choose the activation function and apply it to each input of z^1:

$$\sigma(z^1) = a^1 \in \mathbb{R}^h.$$

Finally we apply this process to the hidden layers, if any. In this case, for the output layer:

$$z^2 = a^1 W^2 + b^2,$$

with $W^2 \in \mathrm{M}_{h\times k}$, $b^2 \in \mathbb{R}^k$.

Forward propagation is the first step in training a neural network. The output of forward propagation must be evaluated with an error function known as the cost function. Next, other mechanisms are necessary to update the neural network parameters and get the predictions closer to the target values. This will be discussed in the next section.

2.3. Learning Process

Once the neural network has generated an output, it continues with the *learning process*. The learning mechanism for a single neuron can be generalised to many of them with diverse arrangements. Here, the network parameters are updated so the output is as close as possible to the label values. The measure for this comparison will be indicated by the cost function. A typical cost function for regression tasks is the mean squared error (Equation (A3)). Thus, the lower the value of the cost function, the better the neural network model. In deep learning, the most popular method for optimising cost functions is known as *gradient descent* (see Appendix B for details). When applying gradient descent to update the network parameters in both the output layer and the hidden layers, an algorithm known as *backpropagation* should be taken into account (Appendix C). This algorithm indicates how to modify the parameters of the neural network such that the value of the cost function becomes smaller in an efficient way. Algorithm 2 outlines the learning process of an MLP, which can be considered as a generalisation of Algorithm 1. The weights and bias updates are performed by applying the backpropagation Equations (A6)–(A9). A common way to initialise the random values for the weights and biases is generating random numbers normally distributed around zero.

Algorithm 2: Learning process.

Data: X, Y.
Step 1: generate random weights W^1 and biases b^1 for the network.
for *i in range*(epochs) **do**
  Step 2: compute an output of the ANN using forward propagation over X, then evaluate it with the cost function C and the expected label Y.
  Step 3: Update the network parameters with the backpropagation equations:

$$W^l \longrightarrow W^l - \eta \nabla_{W^l} C,$$
$$b^l \longrightarrow b^l - \eta \nabla_{b^l} C.$$

end

2.4. Overfitting and Underfitting

During the training process the neural networks tune the values of the weights to minimise the loss function. However, there may be several combinations of weights that similarly minimise the loss function. Therefore, a neural network must generate a good model for the data, capable of generalising and predicting values for the data not included in the training set. These machine learning issues have been rigorously studied in the branch of mathematics known as *statistical learning theory*. In this work, we do not delve into these details, but for references, see [33–35] p. 142.

In ANN training, the dataset is usually split up into two parts: a *training* set and a *validation* set. The training set is used to train the neural network and therefore to fit the weights and biases. On the other hand, the validation set verifies whether that trained neural network can generalise beyond the dataset with which it was trained. The split in the dataset allows one to evaluate whether a neural network generates an acceptable model for the data. Then, we analyse the behaviour of the loss function and its change throughout the epochs by plotting the behaviour of the cost function on the training and validation sets. When a dataset is large enough, it is convenient to split it into three parts: a training set, a validation set, and a test set; the latter is used to perform a final test of the neural network's performance.

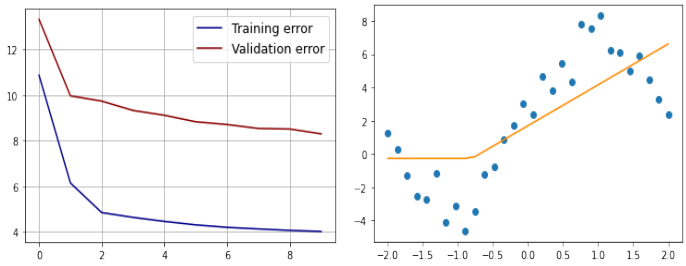
If throughout the epochs the model does not reduce the training error, this is a sign that the model is too simple, so it is not learning the pattern that underlies the training set. This phenomenon is called *underfitting* (Figure 3a) and it could be solved with a more complex model. On the other hand, if the training error is remarkably lower than the validation error—that is, the validation error does not decrease through the epochs while the training error does—this may be a sign that the model is unable to generalise what it has learned. This is called *overfitting* (Figure 3b). Overfitting can be solved with processes called *regularisation techniques*; some examples are dropout [36] and regularisation norms [37,38].

2.5. Coding Tips

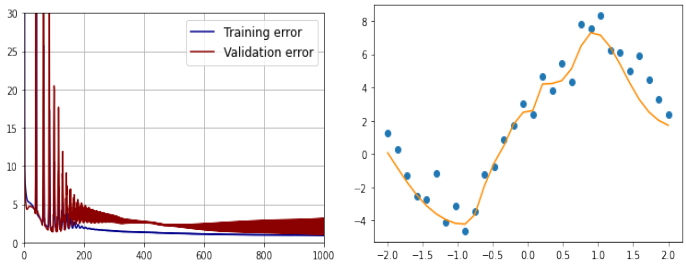
In this section we share some highlights about neural network coding. For technical details, we implemented an MLP from scratch, which is available in [39].

The neural network's architecture must be defined: the number of layers and the number of nodes. In addition, other intrinsic parameters (hyperparameters) of the neural network must also be established, such as the activation function, the number of epochs, and the learning rate, among others. The hyperparameters must be calibrated so the model generated by the neural network does not present problems such as underfitting or overfitting. It is important that the cost function be as small as required by the problem of interest. However, it is also necessary to take into account that the difference between the training and the validation errors should be small. By taking care of these aspects, the model generated by the neural network will be able to generalise to data not included in the original training dataset, and therefore make good predictions.

(**a**) Underfitting



(**b**) Overfitting



(**c**) Good fitting

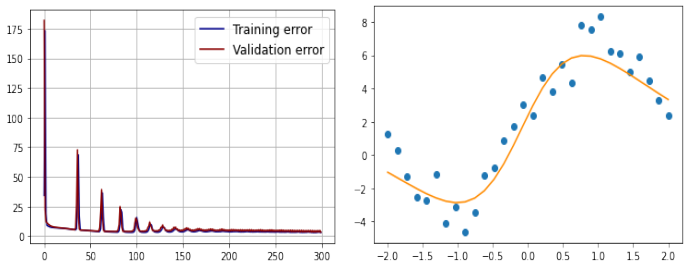


Figure 3. Comparison between different training processes and how we can infer information about model performance by plotting epochs (x-axis) with training and validation error. (**a**): the model was trained with too many epochs, and while the training error converges to zero, the validation error oscillates and the model was too well adapted to the training set that it is unable to generalise. (**b**): the difference between the training curve and the validation is minimal, both converge to zero and the number of epochs is appropriate enough. A good model does not give up on training data, but emulates it well and can be generalised to more elements of the distribution. (**c**): the difference between the training curve and the validation is minimal, both converge to zero and the number of epochs is appropriate enough; a good model does not give up on training data, but emulates it well and can be generalised to more elements of the distribution.

An interesting way to take advantage of a trained neural network model is to store it in a binary text file. In this way, the values of the weight matrices W^l and the bias vectors b^l obtained after proper training can be loaded to make more predictions; i.e., the neural network can be reusable.

Building a neural network from scratch is a very good way to understand the fundamental concepts of deep learning. However, in practice it is necessary to work with different network architectures, several cost functions and activation functions, and a large variety of hyperparameters; therefore, it is best to use specialised deep learning libraries. For instance, in `Python` one can use `Pytorch`, `TensorFlow`, or `Keras`, where the code is very efficient and contains a wide range of options. Other programming languages such as `R` and `Julia` also have deep learning libraries.

3. Cosmological Framework

One of the most important equations in cosmology is the *Friedmann equation*, which describes the evolution of the Universe and its expansion rate:

$$H(t)^2 = \frac{\kappa_0}{3}\rho - \frac{kc^2}{a(t)^2}, \tag{9}$$

where H is the Hubble parameter defined by $H(t) \equiv \dot{a}/a$ with $a(t)$ being the scale factor of the Universe; c is the speed of light; k is a constant that specifies the space geometry (curvature); and κ_0 is given in terms of the gravitational constant G, $\kappa_0 = 8\pi G$. ρ represents the total energy density of the content of the cosmos, which is $\rho = \sum_i \rho_i$, where the index i shows the possible components: radiation (r), baryonic and dark matter (m), and dark energy (Λ).

Another important equation in cosmology is the *continuity equation* or *fluid equation*, which describes the behaviour and evolution of the content of the Universe:

$$\dot{\rho}_i + 3\frac{\dot{a}}{a}\left(\rho_i + \frac{p_i}{c^2}\right) = 0, \tag{10}$$

where p_i represents the pressure associated with every i component. A relationship between density and pressure can be established through an equation of state. The simplest way is to assume that the components behave as perfect fluids and they are described by a barotropic equation of state:

$$p_i = (\gamma_i - 1)\rho_i c^2, \tag{11}$$

where γ_i describes each fluid: radiation ($\gamma_r = 4/3$), baryonic and dark matter ($\gamma_m = 1$), and dark energy in the form of cosmological constant ($\gamma_\Lambda = 0$). By substituting the value of p_i in Equation (10), for each component, a system of couple differential equations is obtained.

$$\dot{\rho}_i + 3\gamma_i H\rho_i = 0. \tag{12}$$

Once we introduce the dimensionless *density parameters*, defined as

$$\Omega_i = \frac{\kappa_0}{3H^2}\rho_i, \tag{13}$$

Equations (12) can be written as a dynamical system with the following form:

$$\Omega_i' = 3(\Pi - \gamma_i)\Omega_i, \tag{14}$$

with $\Pi = \sum_i \gamma_i \Omega_i$, and prime notation means derivative with respect to the e-fold parameter $N = \ln(a)$. The Friedmann equation becomes a constraint for the density parameters at all time:

$$\sum_i \Omega_i = 1. \tag{15}$$

This system can be solved traditionally, by setting some initial conditions for the parameters of radiation density, matter, dark energy, and the Hubble constant: $\Omega_{r,0}$, $\Omega_{m,0}$, $\Omega_{\Lambda,0}$, H_0.

4. MLP Applied to the Hubble Parameter

In this example, we demonstrate the use of a neural network to generate a model from 20 simulated data points [14] corresponding to cosmic chronometers that consist of $H(z)$ measurements at redshift z and their respective statistical error bars. This is a simple application that, in a more rigorous way, has been used in other research works [13–16].

The neural network input corresponds to the redshift z and the output to both H and its error. Therefore, the neural network can be considered as a function from $z \in \mathbb{R} \to \mathbb{R}^2$. Top panel of Figure 4 shows the architecture of the neural network.

In this example, the trained neural network has one hidden layer with 100 nodes, and it was calibrated with a total of 300 trainable parameters (weights and bias). The dataset contains 20 observational data points; however, the computational model generated by the neural network had a good reconstruction of the Hubble parameter without assuming any theoretical model beforehand; its cost function was 1.7. This value is a bit large due to the small size of the dataset; however, it could be improved using more advanced strategies to choose the hyperparameters and to train the neural network [14].

The cost function curves of the training and validation sets show that their errors decreased at each epoch, which suggests that the model was able to learn from the data points provided and generalise to any other missing z. Once the training process was completed, the network was able to predict $H(z)$ values and their errors to values of redshift (z) not included in the original dataset. See the lower panel of Figure 4 for details. In this figure, it can be noted that the neural network generated a model for the $H(z)$ measurements, and thus produced a reconstruction of the Hubble parameter based only on the dataset without making any cosmological assumptions beforehand. This result could be compared with the analytical $H(z)$ function coming from various cosmological models to assess how similar or different the curves are. That is, the MLP can be used to reconstruct various cosmological functions to obtain some physical conclusions [40,41].

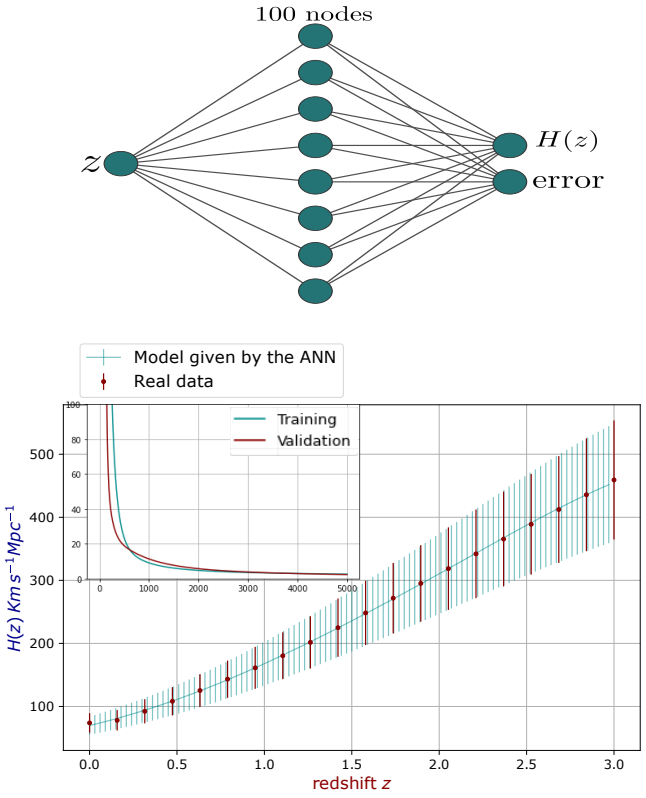


Figure 4. (**Top**): Neural network architecture. The input layer receives the redshift and the output layer generates a value for the Hubble parameter and its respective error. (**Bottom**): Predictions of an ANN model for the Friedmann equation from a few data points. Red error bars are the original data used to train the architecture. Turquoise bars represent the predictions of the trained ANN. In the top left corner is the behaviour of the loss function on both the training and validation sets.

5. Cosmological Differential Equations

Solving a system of differential equations can be computationally demanding, especially when this process has to be repeated multiple times. The use of neural networks in solving systems of differential equations has already been addressed by various authors [42–44]. In cosmology, it is common that some cosmological functions have to be evaluated multiple times, for example, in the case of simulations or in the Bayesian infer-

ence process [17,19,21]. In this example, with various combinations of initial conditions, we solve the dynamical system to generate our training set. Then, we build the optimal architecture of a MLP to generate a model for this dataset. Once the neural network is well trained, we can do the following with the predictions:

- Obtain the solutions of the system by evaluating the initial conditions at missing points in the training set.
- Reduce the computational time to obtain multiple solutions.

Throughout this example, we consider a flat Universe whose evolution goes back to the early times when radiation dominates. We also assume that its temperature is measured with great reliability, and this is reflected in a fix present radiation parameter with value $\Omega_{r,0} = 10^{-4}$. Once this value is established, and since the sum of all densities must be equal to one (Equation (15)), it is enough to vary the parameters $\Omega_{m,0}$ and H_0 to have the initial conditions for Equation (14). With these parameters, the solution of the differential equations can be treated as a function $\Omega_i(N, \Omega_{m,0}, H_0)$.

To generate the set of solutions, the following was performed: First, the intervals where the initial conditions were selected, correspond to $N = \ln(a) \in [-12, 0]$, $\Omega_{m,0} \in [0.1, 0.4]$, and $H_0 \in [65, 80]$. For the choice of these intervals, see [45]. Then, the dataset X was generated by computing the Cartesian product between all the intervals: $X = [-12, 0] \times [0.1, 0.4] \times [65, 80]$, to get a set of about 30,000 elements. Last, we computed the solutions corresponding to these elements X to form the training dataset Y.

The network architecture used to process these data can be seen in Figure 5. Before starting the training process, a max-min transformation was performed on the Y dataset, because the size of the Hubble factor is large for the early stages of the Universe. The MLP architecture had four hidden layers with a sigmoid activation function, and considering both weights and biases, they had a total of 69,154 trainable parameters. It was trained for 500 epochs with a dataset of 30,000 elements and the final cost (error) function has a value of 2.9×10^{-5}.

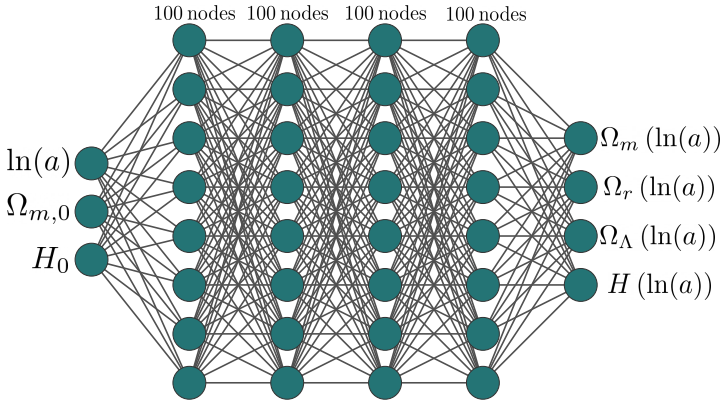


Figure 5. In this architecture the input corresponds to the initial conditions $\Omega_{m,0}$ and H_0, and the domain variable $N = \ln(a)$. The MLP outputs are the solutions of each parameter Ω_i and H, in terms of $\ln(a)$.

Considering the flat Universe components mentioned above, the dynamical system (14) has analytical solutions; therefore, the Friedmann equation, in terms of the redshift, $z = 1/a - 1$, takes the following form:

$$\frac{H^2}{H_0^2} = \Omega_{r,0}(1+z)^4 + \Omega_{m,0}(1+z)^3 + \Omega_{\Lambda,0}. \tag{16}$$

Hence, in the top panel of Figure 6, the solutions and the predictions of the neural network for the density parameters are compared, whereas in the bottom are the results for the Hubble parameter H, and the error curves, which show a good performance. Notice

the dynamical system in (14) can be extended too a more complex scenario that requires sophisticated numerical methods, i.e., [46,47].

Once the ANN was properly trained, we saved the generated model and used it to predict solutions of the system. However, this time, it was enough to evaluate the model under a combination of initial conditions to obtain the solutions (Figure 7). Finally, the model was evaluated in 10,000 different combinations of initial conditions, and we found that the ANN reduces the computing time by 53% compared to the numerical solutions of the differential equations.

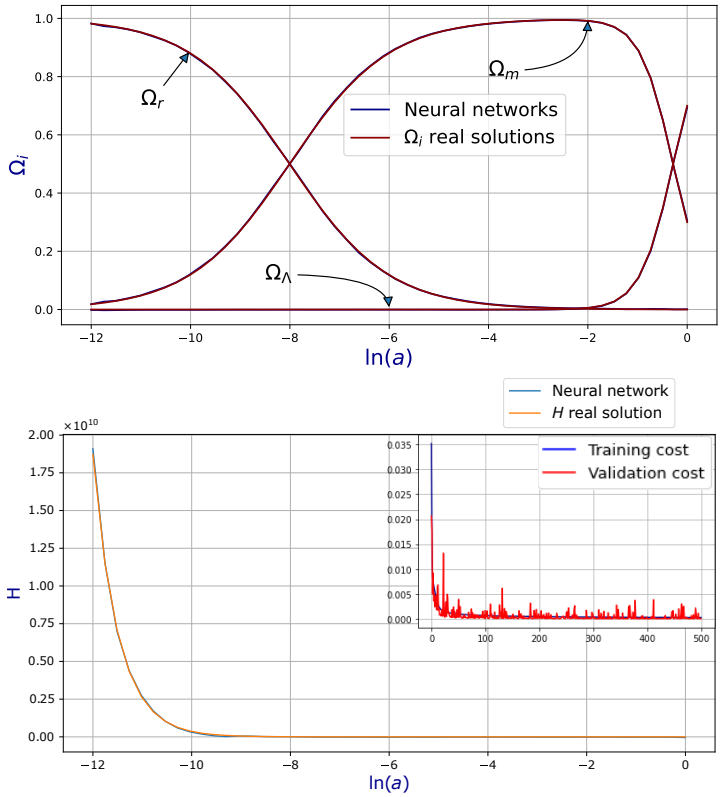


Figure 6. Comparison between the trained model and the real solutions, evaluated over the whole domain $\ln(a) \in [-12, 0]$, with $\Omega_{m,0} = 0.3$, $H_0 = 70$, for the density parameters (**top**) and the Hubble parameter (**bottom**). In the upper right corner we can see the evolution of the error during the training process.

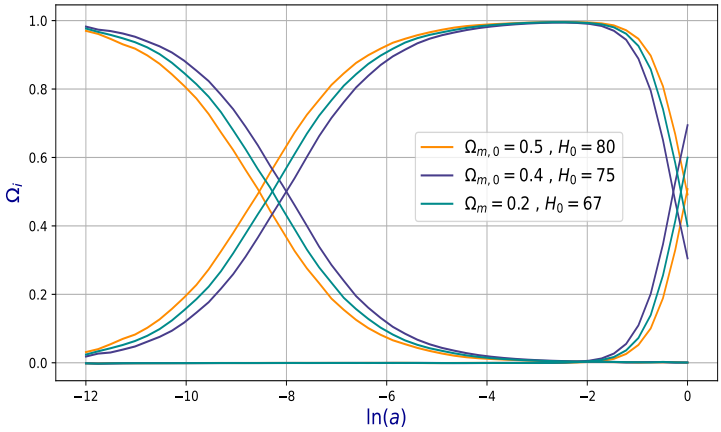


Figure 7. Predictions for different initial conditions, evaluating the ANN model instead of fully solving the differential equations.

6. Classification of Astronomical Objects

The classification of celestial objects is one of the main tasks that astronomers have carried out throughout history. The increase in the flow of data we receive from the cosmos is both an opportunity and a challenge. It allows us to classify objects based on many of their characteristics, but the amount of information is so large that it can become a daunting task. This example shows that deep learning is a fantastic option for performing classification problems, in this case the classification of stellar objects.

In classification problems, the independent variables X usually represent features or attributes, and the dependent variables Y indicate the classes (or labels) of each item in the dataset [3]. Unlike the previous examples, the categorical (or classification) labels must be transformed to numeric elements called *one-hot vectors*. It is convenient to apply the *Softmax* activation function in the last layer, since this activation function makes it possible to associate the ANN output with the probability that the element being processed belongs to each of the existing classes in the dataset. The softmax function, applied to a vector with k inputs x_i, is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}. \tag{17}$$

For this case we have used the *cross-entropy* as a cost function, which is recommended for classification problems, and it is defined as follows:

$$C(Y, a^L) = -\sum_{j=1}^{k} Y_j ln(a_j^L); \tag{18}$$

therefore, this is the function to be minimised during neural network training.

The dataset used for this example comes from the *Sloan Digital Sky Survey DR14* (https://www.sdss.org/dr14/ (accessed on 23 December 2021)), which consists of observations of different stellar objects: stars, quasars (QSO), and galaxies. These classes are represented in the Y label set. On the other hand, there are 17 features which include the redshift and the celestial coordinates, along with characteristics of the spectrograph used. These attributes make up the input features X.

In Figure 8, one can observe the ANN architecture used, with three hidden layers with sigmoid activation and the output layer with the softmax activation function. Taking into account the weights and bias, this neural network has 70,803 trainable parameters.

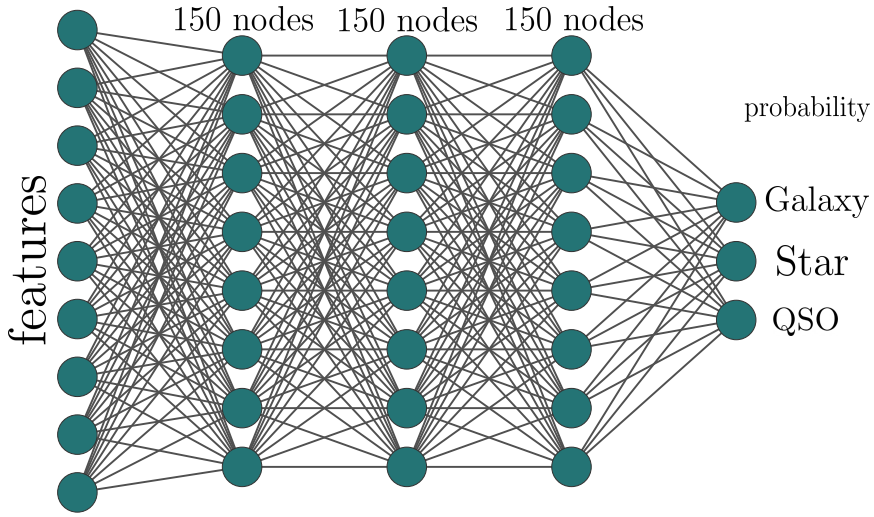


Figure 8. The ANN architecture used for the classification of astronomical objects. The input layer reads the attributes of the SDSS dataset and the output layer corresponds to the probability that each record is a given astronomical object, whether star, galaxy, or quasar.

The ANN was trained for 80 epochs. To measure the performance in this classification task, we used the accuracy metric, defined as follows:

$$\text{accuracy} = \frac{\#\text{Correct predictions}}{\#\text{ Total predictions}}. \tag{19}$$

The ANN model, trained with the astronomical objects, reached 98.32% accuracy on its training set and 98.25% on the validation set, as shown in Figure 9.

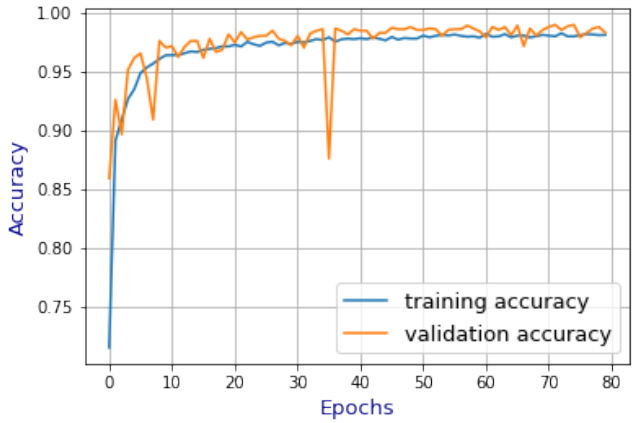


Figure 9. Behaviour of the accuracy metric over the epochs in the training and validation sets. It can be noticed that in both curves the accuracy is very similar and high; therefore, the ANN was well trained.

Finally, for testing the model's efficacy, the ANN was asked to classify a set of 2000 elements outside of the training process, thereby achieving classification with 97.65% accuracy. Results of this classification can be organised in a confusion matrix (see Figure 10), which is an arrangement where the numbers of errors and successes of the ANN can be seen graphically. In a confusion matrix, the horizontal axis represents the real values and the vertical axis those predicted by the model, such that the accuracy ratios are shown diagonally across the matrix, and the mistakes everywhere else. It can be noticed that, in this example, the errors are minimal compared to the correct data labelling done by the ANN.

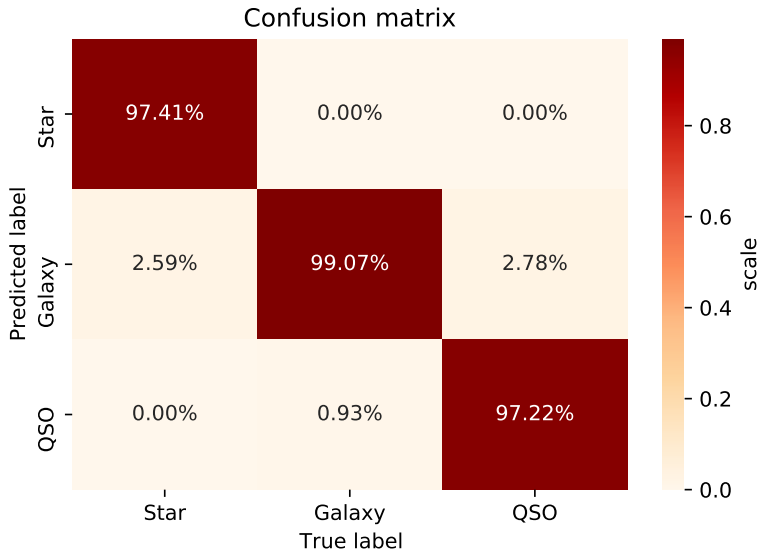


Figure 10. Confusion matrix of the classification model: percentage of accuracy for each of the classes in the test set.

7. Conclusions

This paper presented an introduction to the fundamental concepts of deep learning, with the intention of providing new tools for cosmological analysis. The applications were presented through three practical examples, and through them we showed that:

- Neural networks have the ability to emulate any function or pattern that pervades a given dataset. This can be very useful in various scientific areas, because such networks can generate a computational model for the data and are a good alternative when a satisfactory analytical model is not available.
- A properly trained neural network can be used to replace traditional computational calculations in a wide variety of problems and thus decrease the computational time necessary. In addition, in Section 5 we showed that the neural network also provides a model that can be evaluated and mathematically manipulated, something that traditional numerical methods may not always offer.
- With the last example, we showed the great efficiency of neural networks in tasks that can be complicated to perform—for example, object classification. In our case, we performed a classification with numerical features; however, the literature indicates that neural networks are also a great tool in image and video classification.

The examples that we presented in this article are just a small sample of the potential that deep learning has for cosmology. It is a branch of artificial intelligence that is booming and that, little by little, is being incorporated into various scientific disciplines.

Even though ANNs are considered as a great tool for several problems, in many cases they may present some disadvantages. For example: the training time can be considerably long for a very large database. It is commonly said that neural networks are "black boxes", because the huge number of parameters are just a set of real numbers with no information about the phenomena they are modelling. The ANN must be used with caution to prevent overfitting or underfitting to ensure that the model can be generalised.

Continuing with this line of work, more deep learning tools could also be presented to facilitate data management in science, mainly in astronomy and observational cosmology. A next step could be numerical simulations of the Universe and its contents or using convolutional neural networks, which are widely used in image processing. However, that is a topic for another time.

Author Contributions: All authors contributed equally. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: J.A.V. acknowledges the support provided by FOSEC SEP-CONACYT Investigación Básica A1-S-21925, PRONACES-CONACYT/304001/2020, and UNAM-DGAPA-PAPIIT IA104221. I.G.-V. appreciates the CONACYT postdoctoral grant and ICF-UNAM. J.d.D.R.O. thanks ICF-UNAM, and UNAM-DGAPA-PAPIIT IA102219 and IA104221, for their support.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Activation Functions

Artificial neural networks are known as universal approximators. The approximation theorem [48] proves that, given a continuous function on a compact set of an n-dimensional space $f : \mathbb{R}^n \longrightarrow \mathbb{R}^k$, there exists a neural network with (at least) one hidden layer and a non-linear activation function, which approximates it with any desired degree of precision.

The activation function role is essential in deep learning; without them only linear transformations could be represented, no matter how many hidden layers and nodes are used. The choice of the activation functions depends on each case or the problem being addressed, and the type of behaviour required in each layer of the network.

There are several activation functions that meet the features of being non-linear and continuous; however, a few have been studied and applied more than others. Here are some common examples:

- **ReLU**—*Rectified Linear Unit* (Figure A1a) is the most popular and simple activation function. Given a number x,

$$\text{ReLU}(x) = \max\{0, x\}.$$

This provides a very simple non-linear transformation over $\mathbb{R}$. The ReLU function retains only positive elements and discards all negative by setting $x < 0$ to 0. Although its derivative is undefined when $x = 0$ (Figure A1b), it is not necessary to worry about it because the input values may never actually be all zero at the same time.

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x > 0. \end{cases}$$

Derivatives of the activation functions are a relevant part of the learning process. Therefore, it is important to consider their properties.

- **Sigmoid**—The sigmoid function (Figure A1c) maps the real line to the interval $(0, 1)$. The behaviour of this function was defined by keeping in mind the behaviour of real neurons, which receive stimuli and communicate each other through pulses. The sigmoid function squashes the very negative x to zero, and if x tends to infinity, its image will be mapped to 1; that is, it is a good way to emulate a smoothed step function with 0 or 1. It is defined as:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}},$$

and its derivative (Figure A1d) reaches its maximum when $x = 0$, and when it moves away from this value, it tends to zero

$$\frac{d}{dx}\text{sigmoid}(x) = \frac{e^{-x}}{(1 + e^{-x})^2} =$$

$$\text{sigmoid}(x)(1 - \text{sigmoid}(x)).$$

The second definition is better in computational terms.

- **Hyperbolic tangent**—This function has a similar behaviour to the sigmoid function, but it provides negative values. In the same way, it maps the set of real numbers to the interval $(-1, 1)$. For the points close to 0, the hyperbolic tangent function (tanh) has almost linear behaviour and it is symmetric with respect to the y axis (Figure A1e)

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$

Regarding its derivative (Figure A1f), the behaviour is similar to the derivative of the sigmoid

$$\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2.$$

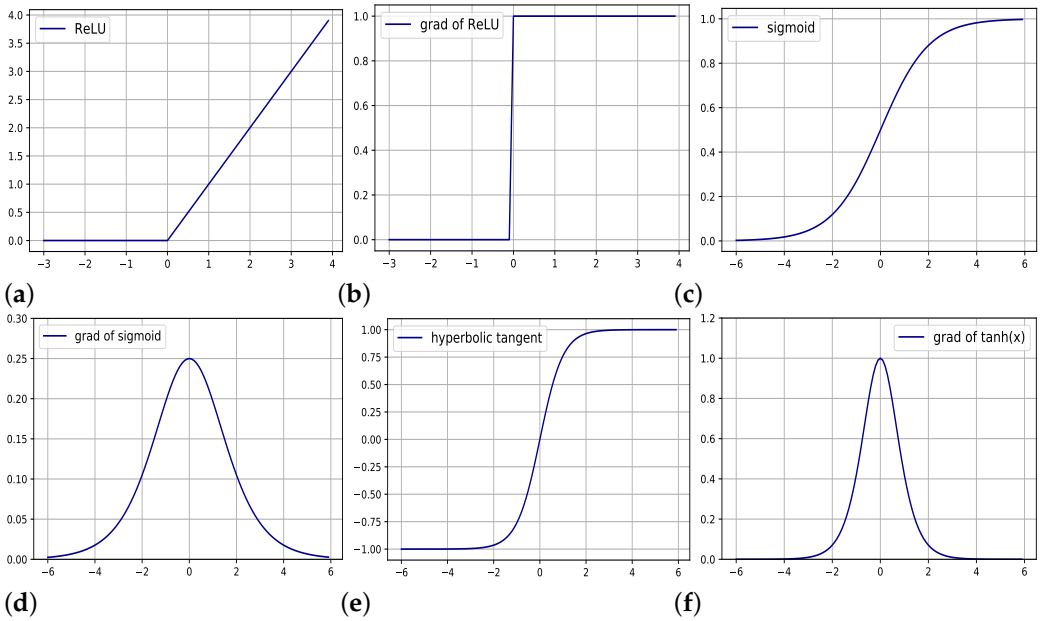


Figure A1. Some common activation functions: rectified linear unit (ReLu), sigmoid, and hyperbolic tangent (tanh). All of them are nonlinear functions. **Left**: the curves of the activation functions, and **Right**: their derivatives which are intensively used through backpropagation. (**a**) ReLu(x); (**b**) ReLu'(x); (**c**) sigmoid(x); (**d**) sigmoid'(x); (**e**) tanh(x); (**f**) tanh'(x).

Appendix B. Gradient Descent

Even though there exists a broad diversity of optimisation algorithms used to minimise the cost function, in deep learning, gradient descent is the most popular. It is versatile and may be generalised to multivariable functions. For example, given a scalar differentiable function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, the vector whose components are the partial derivatives of f, is called a gradient $\nabla f(w)$. In addition, the gradient has some interesting geometric properties:

- $\nabla f(w_0)$ is orthogonal to the level curve $f(w) = k$ at the point w_0.
- $\nabla f(w)$ points to the direction of the maximum increase of f.
- Conversely, $-\nabla f(w)$ points to the direction of the maximum decrease.

The gradient descent algorithm relies on the third property. Let f be a function such that it has a minimum at x_0, with a point x located at certain region of the domain. By taking a step in the $-\nabla f(x)$ direction, the new position is one step closer to the critical point x_0, and with sufficient number of iterations, the value for which f is minimised can be found. The following equation synthesises the mechanism of the gradient descent:

$$w' \longrightarrow w - \nabla f(w), \tag{A1}$$

where x' is the new point to be taken by the algorithm and x is the current point. It can be noticed that one of the relevant advantages of the gradient descent over other types of optimisation algorithms is that it does not use second derivatives, and this property makes it more suitable.

An important parameter within the gradient descent is the *learning rate*, commonly denoted by η. This parameter determines the step size that it takes in each iteration; therefore, this value is very important because it influences whether or not the algorithm is able to converge on the target value in an appropriate way. The learning rate is usually a positive real constant number ($\eta \in [0, 1]$) by which the gradient is multiplied. When η is incorporated into Equation (A1). This results in:

$$w' \longrightarrow w - \eta \nabla f(w). \tag{A2}$$

What should value for η be? It depends on the function to be minimised; different types of problems correspond to different values. There are some cases where it is more useful to consider a dynamic learning rate, whose size increases and shrinks as it approaches the minimum of the function. In this case, the step size in each iteration depends on two factors: the norm of the gradient vector and the η value. As we get closer to the critical point, the step size tends to get shorter, even though η is constant, since at the critical point the gradient tends to have a zero vector. In addition, there are three cases to consider for the learning rate magnitude:

- **Too small an** η: If η is very small, the step size will be too short, then it might increase the total computation time to a very large extent or could even fail to converge on the desired point (see Figure A2a).
- **Too large an** η: A very large learning rate can cause an exploding gradient and diverge from the minimum, or the algorithm may bypass the local minimum and overshoot (see Figure A2b).
- **Optimal** η: A proper learning rate ensures that the algorithm can converge on the minimum of the function on a reasonable number of attempts, which also reduces the computational time taken for the process (Figure A2c).

We can implement gradient descent by following these three steps:

1. Compute partial derivatives of each variable x_k and evaluate them at a random starting point w_0 in the domain of the function: $\frac{\partial f}{\partial w_k}(w_0)$.
2. The gradient of the function must be constructed. It is recommended to define a maximum value for the norm of the gradient, but while preserving its direction, to avoid exploding when doing the iterations.
3. Apply Equation (A2) to the initial point x_0, and the process is repeated for the new point until it is close enough to the minimum.

In order to summarise the ideas, we applied the gradient descent algorithm to the function $f(w_1, w_2) = w_1^2 + w_2^2$ to find its minimum, using a learning rate $\eta = 0.1$ and a starting point $w_0 = (w_{1,0}, w_{2,0}) = (-10, 2.8)$. In Figure A3 we plot the 2D trajectory the algorithm follows during the process. One can see the steps on the level curves of f until reaching the point where the minimum is located, in this case $(w_1, w_2) = (0, 0)$.

In Figure A2c, it can be noticed that the gradient descent algorithm found its minimum easily, although this is not always possible for every function. In general, there are difficulties with functions that have local minima or are not convex. The convergence of the algorithm can be guaranteed if f meets a couple of conditions [49]:

1. It is convex.
2. Its gradient vector is Lipschitz continuous, that is, for a real positive number L: $\|\nabla f(w) - \nabla f(v)\| < L\|w - v\|$.

The gradient descent can be applied to many cost functions satisfying these conditions, which ensures that the desired critical point can be obtained. An example of a function that satisfies the previous characteristics is the mean squared error (MSE). Therefore, we can use the MSE on the perceptron and then minimise it with the help of the gradient descent.

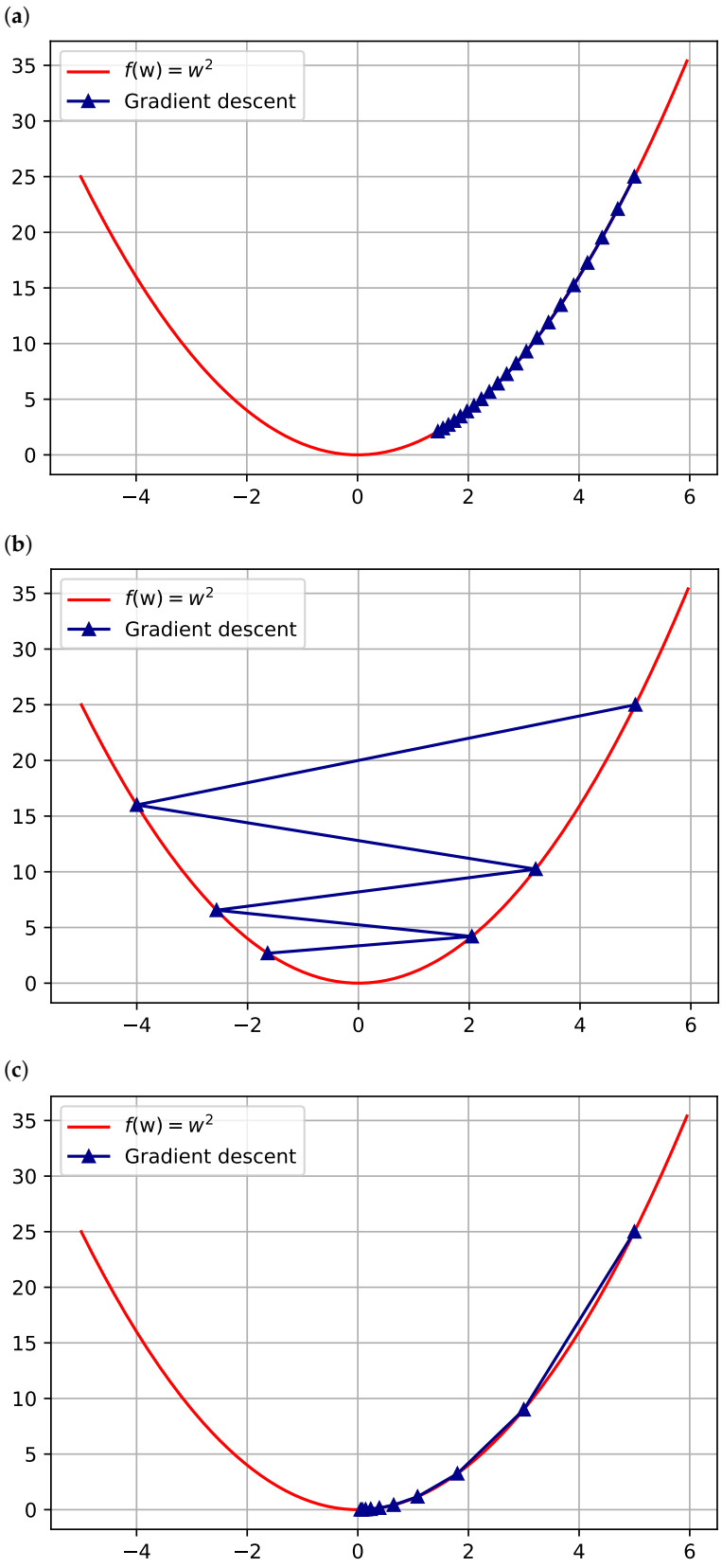


Figure A2. Three different selections of the learning rate and their impact on the gradient descent when applying to minimise a function. (**a**) Too small η; (**b**) Too large η; (**c**) Optimal η.

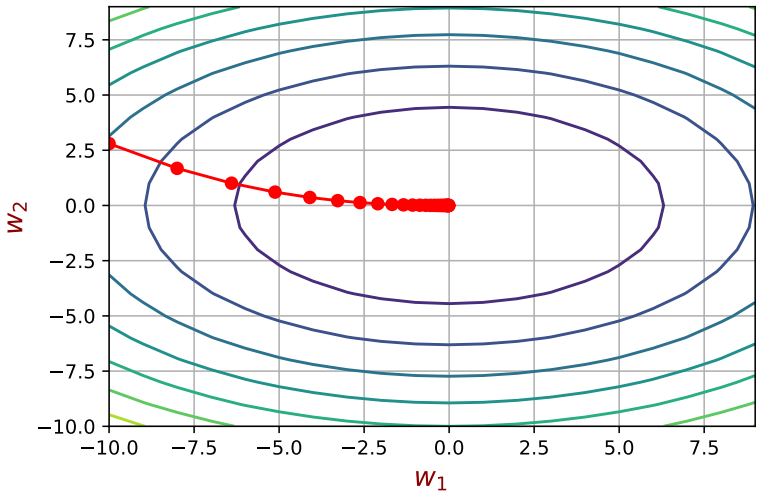


Figure A3. The red line displays the path taken by the gradient descent in the f domain to find its minimum. The ellipses correspond to the level curves of f.

In practice, there are variants of the gradient descent algorithm that are more efficient in some scenarios, for example, stochastic gradient descent and batch gradient descent.

Appendix C. Backpropagation Equations

Considering the mean squared error as the cost function, which compares a prediction of the neural network a^L with the expected value Y, we have the following:

$$C = \frac{1}{2m}\|Y - a^L\|^2 = \frac{1}{2m}\sum_j \left(Y_j - a_j^L\right)^2, \tag{A3}$$

where m is the number of samples. In order to optimise Equation (A3), the stochastic gradient descent algorithm can be applied iteratively until the minimum is reached. The weights w_{ij}^k and biases b_j^l must be updated in each iteration to improve the predictions of the MLP and to reduce the value of the cost function. However, considering a problem with multiple variables influencing the cost function, from Equation (A2) we have the new weights values:

$$W^l \longrightarrow W^l - \eta\, \partial_{W^l}\left(\frac{1}{2}\sum_j \left(Y_j - a_j^L\right)^2\right), \tag{A4}$$

and the following bias values:

$$b^l \longrightarrow b^l - \eta\, \partial_{b^l}\left(\frac{1}{2}\sum_j \left(Y_j - a_j^L\right)^2\right). \tag{A5}$$

With Equations (A4) and (A5) it is possible to know how to update the MLP hyperparameters. However, these expressions only depend on the output layer parameters, and the information of intermediate layers is unclear. Therefore, it is useful to know how much the weights and biases influence the cost function. The algorithm that allows one to propagate the error of the output layer to the elements of the previous layers is known as the *backpropagation*. Thus, it is necessary to use the chain rule to find the partial derivatives in Equations (A4) and (A5). By doing this, the *fundamental equations of backpropagation* are obtained, which is a key concept in deep learning (see more details in [50]).

The backpropagation equations are derived as follows. First of all, it is necessary to consider the change in the cost function with respect to the weighted sum of the last layer. Using the chain rule, we have:

$$\frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L},$$

$$\frac{\partial C}{\partial a_j^L} = \left(a_j^L - Y_j\right), \qquad \frac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L).$$

Thus, considering these components we can obtain $\frac{\partial C}{\partial z^L}$, which is equivalent to:

$$\frac{\partial C}{\partial z^L} = \left(a^L - Y\right) \odot \sigma'(z^L) = \delta^L, \tag{A6}$$

where $\odot$ represents the product component by component between two vectors of equal size. The expression in Equation (A6) denoted by δ^L, often called *imputed error*, can be calculated for any cost function applied to the neural network (MSE in this case). The next step is to analyse how the cost function changes with respect to the variations of the parameters W^L and b^L in the last layer:

$$\frac{\partial C}{\partial W^L} = \underbrace{\frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}}_{\delta^L}\frac{\partial z^L}{\partial W^L}, \qquad \frac{\partial C}{\partial b^L} = \underbrace{\frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}}_{\delta^L}\frac{\partial z^L}{\partial b^L}.$$

Note that in both equations, the first two factors are nothing other than the error δ^L. Furthermore:

$$\frac{\partial z^L}{\partial W^L} = {a^{L-1}}^T, \qquad \frac{\partial z^L}{\partial b^L} = 1.$$

Therefore, substituting these values, the partial derivatives are as follows:

$$\frac{\partial C}{\partial W^L} = {a^{L-1}}^T \delta^L, \qquad \frac{\partial C}{\partial b^L} = \delta^L.$$

These last two equations are the partial derivatives that the gradient descent needs to update, W^L and b^L, but this is only for the last layer; therefore, it is necessary to find the change in C due to the previous layers. This is relatively straightforward, since it is enough to calculate the change with respect to the $L-1$ layer to find the values ub the previous layers. For $L-1$:

$$\frac{\partial C}{\partial W^{L-1}} = \underbrace{\frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}}_{\delta^L}\underbrace{\frac{\partial z^L}{\partial a^{L-1}}}_{{W^L}^T}\underbrace{\frac{\partial a^{L-1}}{\partial z^{L-1}}}_{\sigma'(z^{L-1})}\underbrace{\frac{\partial z^{L-1}}{\partial W^{L-1}}}_{{a^{L-2}}^T},$$

$$\frac{\partial C}{\partial b^{L-1}} = \underbrace{\frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}}_{\delta^L}\underbrace{\frac{\partial z^L}{\partial a^{L-1}}}_{{W^L}^T}\underbrace{\frac{\partial a^{L-1}}{\partial z^{L-1}}}_{\sigma'(z^{L-1})}\underbrace{\frac{\partial z^{L-1}}{\partial b^{L-1}}}_{1}.$$

Notice that the first four factors of both equations correspond to the derivative

$$\delta^{L-1} = \frac{\partial C}{\partial z^{L-1}} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}},$$

which is equivalent to the imputed error of the $L-1$ layer. However, these partial derivatives have already been deduced and their values are known, so the imputed error is:

$$\delta^{L-1} = \delta^L {W^L}^T \odot \sigma'\left(z^{L-1}\right).$$

Even though this expression was derived for the imputed error of layer $L-1$, it can be noticed that, if we were going to find the error corresponding to the layer $L-2$, we would obtain the same relation between this one and the layer that precedes it; thus, this relation

can be generalised for any layer $l-1$. Moreover, doing a change of indices, we obtain the expression:

$$\delta^l = \delta^{l+1} {W^{l+1}}^T \odot \sigma'\left(z^l\right). \tag{A7}$$

This same generalisation and index arrangement is valid for the expressions $\frac{\partial C}{\partial W^{L-1}}$ and $\frac{\partial C}{\partial b^{L-1}}$, from which the expressions for the derivatives of the hidden layers are obtained:

$$\frac{\partial C}{\partial W^l} = {a^{l-1}}^T \delta^l, \tag{A8}$$

$$\frac{\partial C}{\partial b^l} = \delta^l. \tag{A9}$$

Equations (A6)–(A9) are known as the *fundamental backpropagation equations*, since they can update the parameters in each layer by applying the gradient descent mentioned in Equations (A4) and (A5). Furthermore, if Equations (A8) and (A9) are substituted, we obtain:

$$W^l \longrightarrow W^l - \eta \nabla_{W^l} C = \eta \sum_x {a_x^{l-1}}^T \delta_x^l, \tag{A10}$$

$$b^l \longrightarrow b^l - \eta \nabla_{b^l} C = \eta \sum_x \delta_x^l, \tag{A11}$$

where the subscript x refers to the examples contained in the training set X, with $x \in X$.

Backpropagation is a crucial process for learning in a neural network, as the parameters are updated following this algorithm. To implement the backpropagation in a neural network, the following steps are necessary:

1. Generate random values for W^l and b^l. Then compute the corresponding output a^L by forward propagation.
2. Compute the value of the cost function (A3). Then obtain the imputed error δ^L with respect to the output of the neural network (Equation (A6)).
3. Compute the imputed error for the previous layers $L-1, L-2\ldots$ using Equation (A7).
4. Use Equations (A10) and (A11) to update the MLP parameters.
5. Once the new parameters are in place, iterate this procedure until the cost function reaches a very small value.

References

1. Arjona, R.; Nesseris, S. What can machine learning tell us about the background expansion of the universe? *Phys. Rev. D* **2020**, *101*, 123525. [CrossRef]
2. Wang, G.-J.; Ma, X.-J.; Xia, J.-Q. Machine learning the cosmic curvature in a model-independent way. *Mon. Not. R. Astron. Soc.* **2021**, *501*, 5714. [CrossRef]
3. Chacón, J.; Vázquez, J.A.; Almaraz, E. Classification algorithms applied to structure formation simulations. *arXiv* **2021**, arXiv:2106.06587.
4. Lin, H.W.; Tegmark, M.; Rolnick, D. Why does deep and cheap learning work so well? *J. Stat. Phys.* **2017**, *168*, 1223. [CrossRef]
5. Peel, A.; Lalande, F.; Starck, J.; Pettorino, V.; Merten, J.; Giocoli, C.; Meneghetti, M.; Baldi, M. Distinguishing standard and modified gravity cosmologies with machine learning. *Phys. Rev. D* **2019**, *100*, 023508. [CrossRef]
6. Rodríguez, A.C.; Kacprzak, T.; Lucchi, A.; Amara, A.; Sgier, R.; Fluri, J.; Hofmann, T.; Réfrégier, A. Fast cosmic web simulations with generative adversarial networks. *Comp. Astrophys. Cosmol.* **2018**, *5*, 4. [CrossRef]
7. He, S.; Li, Y.; Feng, Y.; Ho, S.; Ravanbakhsh, S.; Chen, W.; Póczos, B. Learning to predict the cosmological structure formation. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 13825. [CrossRef]
8. Dieleman, S.; Willett, K.W.; Dambre, J. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Mon. Not. R. Astron. Soc.* **2015**, *450*, 1441. [CrossRef]
9. Ntampaka, M.; ZuHone, J.; Eisenstein, D.; Nagai, D.; Vikhlinin, A.; Hernquist, L.; Marinacci, F.; Nelson, D.; Pakmor, R.; Pillepich, A.; et al. A deep learning approach to galaxy cluster X-ray masses. *Astrophys. J.* **2019**, *876*, 82. [CrossRef]
10. Auld, T.; Bridges, M.; Hobson, M.; Gull, S. Fast cosmological parameter estimation using neural networks. *Mon. R. Astron. Soc. Lett.* **2007**, *376*, L11. [CrossRef]
11. Alsing, J.; Charnock, T.; Feeney, S.; Wandelt, B. Fast likelihood-free cosmology with neural density estimators and active learning. *Mon. R. Astron. Soc.* **2019**, *488*, 4440. [CrossRef]

12. Li, S.-Y.; Li, Y.-L.; Zhang, T.-J. Model comparison of dark energy models using deep network. *Res. Astron. Astrophys.* **2019**, *19*, 137. [CrossRef]
13. Dialektopoulos, K.; Said, J.L.; Mifsud, J.; Sultana, J.; Adami, K.Z. Neural network reconstruction of late-time cosmology and null tests. *arXiv* **2021**, arXiv:2111.11462.
14. Gómez-Vargas, I.; Vázquez, J.A.; Esquivel, R.M.; García-Salcedo, R. Cosmological Reconstructions with Artificial Neural Networks. *arXiv* **2021**, arXiv:2104.00595.
15. Wang, G.-J.; Ma, X.-J.; Li, S.-Y.; Xia, J.-Q. Reconstructing functions and estimating parameters with artificial neural networks: A test with a hubble parameter and sne ia. *Astrophys. Suppl. Ser.* **2020**, *46*, 13. [CrossRef]
16. Escamilla-Rivera, C.; Quintero, M.A.C.; Capozziello, S. A deep learning approach to cosmological dark energy models. *J. Cosmol. Astropart. Phys.* **2020**, *2020*, 008. [CrossRef]
17. Graff, P.; Feroz, F.; Hobson, M.P.; Lasenby, A. Bambi: Blind accelerated multimodal bayesian inference. *Mon. Not. R. Soc.* **2012**, *421*, 169. [CrossRef]
18. Moss, A. Accelerated bayesian inference using deep learning. *Mon. Not. R. Astron. Soc.* **2020**, *496*, 328. [CrossRef]
19. Hortua, H.J.; Volpi, R.; Marinelli, D.; Malago, L. Accelerating mcmc algorithms through bayesian deep networks. *arXiv* **2020**, arXiv:2011.14276.
20. Gómez-Vargas, I.; Esquivel, R.M.; García-Salcedo, R.; Vázquez, J.A. Neural network within a bayesian inference framework. *J.Phys. Conf. Ser.* **2021**, *1723*, 012022. [CrossRef]
21. Mancini, A.S.; Piras, D.; Alsing, J.; Joachimi, B.; Hobson, M.P. CosmoPower: Emulating Cosmological Power Spectra for Accelerated Bayesian Inference from Next-Generation Surveys. *Mon. Not. R. Astron. Soc.* **2021**, *511*, 1771. [CrossRef]
22. Baccigalupi, C.; Bedini, L.; Burigana, C.; De Zotti, G.; Farusi, A.; Maino, D.; Maris, M.; Perrotta, F.; Salerno, E.; Toffolatti, L.; et al. Neural networks and the separation of cosmic microwave background and astrophysical signals in sky maps. *Mon. Not. R. Astron. Soc.* **2000**, *318*, 769. [CrossRef]
23. Petroff, M.A.; Addison, G.E.; Bennett, C.L.; Weiland, J.L. Full-sky cosmic microwave background foreground cleaning using machine learning. *Astrophys. J.* **2020**, *903*, 104. [CrossRef]
24. Pasquet-Itam, J.; Pasquet, J. Deep learning approach for classifying, detecting and predicting photometric redshifts of quasars in the sloan digital sky survey stripe 82. *Astron. Astrophys.* **2018**, *611*, A97. [CrossRef]
25. Ribli, D.; Pataki, B.; Csabai, I. An improved cosmological parameter inference scheme motivated by deep learning. *Nat. Astron.* **2019**, *3*, 93–98. [CrossRef]
26. Ishida, E.E. Machine learning and the future of supernova cosmology. *Nat. Astron.* **2019**, *3*, 680. [CrossRef]
27. List, F.; Rodd, N.L.; Lewis, G.F.; Bhat, I. Galactic center excess in a new light: Disentangling the γ-ray sky with bayesian graph convolutional neural networks. *Phys. Rev. Lett.* **2020**, 125, 241102. [CrossRef]
28. Dax, M.; Green, S.R.; Gair, J.; Macke, J.H.; Buonanno, A.; Schölkopf, B. Real-time gravitational wave science with neural posterior estimation. *Phys. Rev. Lett.* **2021**, *127*, 241103. [CrossRef]
29. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity.. *Bull. Math. Biophys.* **1943**, *5*, 115. [CrossRef]
30. Rosenblatt, F.; Papert, S. *The Perceptron: A Perceiving and Recognizing Automaton*; Cornell Aeronautical Laboratory Report; Cornell Aeronautical Laboratory: Buffalo, NY, USA, 1957.
31. Minsky, M.; Papert, S. *Perceptron: An Introduction to Computational Geometry*; The MIT Press: Cambridge, UK, 1969; Volume 19, p. 2.
32. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533. [CrossRef]
33. Ying, X. An overview of overfitting and its solutions. *J. Phys. Conf. Ser.* **2019**, *1168*, 022022. [CrossRef]
34. Allamy, H. Methods Avoid Over-Fitting Under-Fitting SupervisedMachine Learn. (Comparative Study). In *Computer Science, Communication & Instrumentation Devices*; Academia.edu: San Francisco, CA, USA, 2015; Volume 70.
35. Zhang, A.; Lipton, Z.C.; Li, M.; Smola, A.J. Dive into Deep Learning. *arXiv* 2021, arXiv:2106.11342.
36. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929.
37. Louizos, C.; Welling, M.; Kingma, D.P. Learning sparse neural networks through *l*_0 regularization. *arXiv* **2017**, arXiv:1712.01312.
38. Phaisangittisagul, E. An analysis of the regularization between l2 and dropout in single hidden layer neural network. In Proceedings of the 2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), Bangkok, Thailand, 25–27 January 2016; pp. 174–179.
39. Full Code Repository. Available online: https://github.com/JuanDDiosRojas/Arts/tree/main/Deep%20Learning%20and%20its%20applications%20to%20cosmology (accessed on 23 December 2021).
40. Escamilla, L.A.; Vazquez, J.A. Model selection applied to non-parametric reconstructions of the Dark Energy. *arXiv* **2021**, arXiv:2111.10457.
41. Keeley, R.E.; Shafieloo, A.; Zhao, G.-B.; Vazquez, J.A.; Koo, H. Reconstructing the Universe: Testing the Mutual Consistency of the Pantheon and SDSS/eBOSS BAO Data Sets with Gaussian Processes. *Astron. J.* **2021**, *161*, 151. [CrossRef]
42. Lagaris, I.; Likas, A.; Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. doi: 10.1109/72.712178 [CrossRef]

43. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations. *arXiv* 2017, arXiv:1711.10561.
44. Dufera, T.T. Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. *Mach. Learn. Appl.* **2021**, *5*, 100058. [CrossRef]
45. Padilla, L.E.; Tellez, L.O.; Escamilla, L.A.; Vazquez, J.A. Cosmological Parameter Inference with Bayesian Statistics. *Universe* **2021**, *7*, 213. [CrossRef]
46. Vázquez, J.A.; Tamayo, D.; Sen, A.A.; Quiros, I. Bayesian model selection on scalar ϵ-field dark energy. *Phys. Rev. D* **2021**, *103*, 043506. [CrossRef]
47. Gonzalez, T.; Matos, T.; Quiros, I.; Vazquez-Gonzalez, A. Self-interacting Scalar Field Trapped in a Randall-Sundrum Braneworld: The Dynamical Systems Perspective. *Phys. Lett. B* **2009**, *676*, 161. [CrossRef]
48. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, 2, 359. [CrossRef]
49. Gower, R.M. *Convergence Theorems for Gradient Descent*; Lecture notes for Statistical Optimization; 2018. Available online: https://moodle.polytechnique.fr/pluginfile.php/246753/mod_resource/content/1/lectures%20notes%20on%20gradient%20descent%20.pdf (accessed on 23 December 2021).
50. Nielsen, M.A. *Neural Networks and Deep Learning*; Determination Press: San Francisco, CA, USA, 2015; Volume 25.