Universidad de La Laguna

Physics Degree

Undergraduate Thesis

Artificial Neural Networks and their use in simulation-based inference with the CAMELS dataset.

Cecilia Arrizabalaga Díaz-Caneja

July 2022

supervised by
Dr. Marc Huertas-Company

Acknowledgements

First of all I want to thank all my friends and family for supporting and encouraging me during the whole process. I would also like to thank my supervisor Dr. Marc Huertas-Company for guiding this work and helping me with all my doubts.

Front cover picture from the CAMELS Multifield dataset [36].

Resumen

En las últimas decadas ha habido una evolución en el uso de la Inteligencia Artificial. AI, y más específicamente Machine Learning, es la combinación de la informática y grandes conjuntos de datos que entrenan una máquina para resolver problemas extensos o problemas exigentes. En 1943 Walter Pitts (lógico estadounidense) y Waren McCulloch (neurólogo estadounidense) introdujeron la primera red neuronal, un algoritmo de aprendizaje automático. Estos estudios brindan a las máquinas la capacidad de recopilar datos y luego procesar lo que se ha recopilado utilizando herramientas de aprendizaje automático y técnicas de predicción para tomar decisiones.

Las redes neuronales son un tipo de algoritmo que hace uso de una base de datos para hacer una clasificación, i.e. clasificar nuevos datos en diferentes categorías a partir de lo aprendido con la base de datos, o una regresión, predecir valores continuos. Como su nombre indica las redes neuronales están inspiradas en el cerebro humano, imitando la forma en que las neuronas biológicas se envían señales entre sí.

En este informe se pretende explicar el funcionamiento de estas redes neuronales y comentar varias de sus estructuras, así como obtener algunos resultados utilizando este tipo de algoritmos. En la segunda mitad del trabajo se describe una base de datos que se ha utilizado para deducir algunos valores de las constantes cosmológicas y astrofísicas Ω_m, σ_8, A_{SN1}, A_{SN2}, A_{AGN1} y A_{AGN2} para diferentes regiones de universos simulados.

Una Red Neuronal consta de una capa de neuronas de entrada, unas capas de neuronas ocultas y una capa de neuronas de salida. Las conexiones entre las diferentes neuronas se llaman pesos, y describen con cuánta fuerza afecta el resultado de la neurona anterior. La siguiente ecuación describe el resultado o valor que tiene cada neurona [39]:

$$h_i^l = f\left(\sum_{j=1}^{J} V_{ij}^l \cdot h_j^{l-1} + T_i^l\right)$$

donde h_i^l es el resultado obtenido de la neurona i en la capa l, f es la función de activación que determina si la neurona es activada o no, J es el número de neuronas de la capa anterior $l-1$ que tienen conexión con la neurona i, V_{ij}^l son los pesos de las conexiones entre la neurona j e i, h_j^{l-1} es el valor que sale de la neurona j y T_i^l es el valor *bias* o *valor umbral* de cada neurona, es decir, el valor mínimo para activar la neurona. A través de esta expresión la información de entrada avanzará a través de la red neuronal para al final dar un valor de salida que, en el caso de ser un problema de clasificación será una categoría, y en el caso de ser un problema de regresión será un valor. Para poder tener alta eficacia la red neuronal necesita aprender utilizando una base de datos: lo que hace es calcular su valor de salida y compararlo con el valor esperado calculando la llamada *función de pérdida* y minimizándola actualizando los valores de los pesos y bias en cada iteración. Para conseguir esta minimización de la función de pérdida se necesitan sus derivadas, que dependen de la función de activación y los valores de los pesos. Por lo tanto, si los valores de estas derivadas o de los pesos son muy pequeños,

o por el contrario muy grandes, podrían conducir al "gradiente evanescente" y "gradiente explosivo" respectivamente. Según la ecuación 8:

$$\mathbf{V} = \mathbf{V} - \lambda \frac{\partial E_k}{\partial \mathbf{V}}$$

podemos comprobar que si las derivadas de la función de pérdida $^{\partial E_k}/_{\partial \mathbf{v}}$ son muy pequeñas, los pesos correspondientes no se actualizarán apenas y por lo tanto la red no aprenderá correctamente. Con el gradiente explosivo pasa lo contrario. Si los valores de las derivadas son demasiado grandes nos arriesgamos a que la red nunca encuentre los valores óptimos de los pesos ya que da "saltos" demasiado grandes al actualizarlos.

Para obtener los resultados hemos hecho uso de la base de datos CMD, Cosmology and Astrophysics with MachinE Learning Simulation (CAMELS) Multifield Dataset (CMD) [36], una colección de mapas 2D y 3D de diferentes regiones de universos simulados, generados a partir de simulaciones magneto-hidrodinámicas y gravitacionales de N-cuerpos. Cada una de estas simulaciones pertenece a uno de los siguientes subgrupos: IllustrisTNG, SIMBA y N-body. Los mapas de todos estos grupos tienen asociados dos parámetros cosmológicos: Ω_m y σ_8. Sin embargo, solo los mapas de las simulaciones de IllustrisTNG y SIMBA tienen asociados los parámetros astrofísicos A_{SN1}, A_{SN2}, A_{AGN1} y A_{AGN2}. A su vez, los mapas de estos grupos pueden representar diferentes propiedades como la temperatura del gas cósmico, su densidad, su metalicidad, etc.

Concluímos este trabajo con la estimación de los parámetros anteriormente mencionados con los mapas de temperatura del gas cósmico del conjunto de simulaciones IllustrisTNG a través de MDNs (Mixture Density Functions) que, en vez de devolver directamente valores específicos de los parámetros, devuelve la media y la varianza de una distribución gaussiana. La precisión de los resultados no ha sido la esperada por el efecto del over-fitting, efecto que ocurre cuando la red neuronal es muy efectiva prediciendo datos del set de entrenamiento pero no tan buena al estimar datos que no forman parte del entrenamiento, y se deduce que los resultados se podrían mejorar añadiendo mapas para el entrenamiento de la red.

Contents

Abstract

Neural Networks are a powerful resource to study problems that maybe other algorithms cannot. In this report we will discuss the functioning of a neural network as well as some of its most commonly used structures. With this knowledge we will study their applicability in a simulation-based inference of cosmological and astrophysical parameters with Mixture Density Convolutional Networks using only simulated images generated with data from The Cosmology and Astrophysics with Machine Learning Simulations (CAMELS) [36].

1 Introduction

Resumen

Muchos avances en astrofísica han sido impulsados por la evolución tecnológica, como la teoría del Heliocentrismo de Copérnico, que no pudo ser confirmada hasta el uso del telescopio por Galileo Galilei, en 1610. En las últimas decadas ha habido una evolución en el uso de la Inteligencia Artificial y el Machine Learning.

Major discoveries in astrophysics are generally driven by technological developments. An obvious example of these discoveries is The Copernican Principle in 1543, in which Nicolaus Copernicus stated that the earth is not in a central position, in contrast to the geocentrism. However, this theory was not confirmed until Galileo Galilei turned his own telescope towards the sky and observed the phases of Venus in 1610, just how Copernicus had foretold [13].

In recent times, there has been an evolution in the use of Artificial Intelligence. AI, and more specifically **Machine Learning**, is the combination of computer science and large datasets that train a machine in order to solve extensive or demanding problems.

In 1943 Walter Pitts (american logician) and Waren McCulloch (american neurologist) introduced the first neural network, a Machine Learning algorithm. Recent machine learning research has focused on computer vision, hearing, natural language processing, image processing and pettern recognition, etc. These studies provide machines with the ability to collect data and then processing what has been collected using Machine Learning tools and prediction techniques to make decisions [21].

2 Objectives

Resumen

Los principales objetivos de este trabajo son: explicar el funcionamiento de las redes neuronales y algunas de sus estructuras, describir la base de datos CAMELS (Cosmology and Astrophysics with MachinE Learning Simulation) [36] y comentar algunos resultados de la deducción de parámetros cosmológicos y astrofísicos con el CMD (CAMELS Multifield Dataset).

The key contributions of this report are listed as follows:

- Explaining the functioning of the Artificial Neural Networks and some of the architectures..

- Describing the CAMELS Multifield Dataset [36], used for the inference of cosmological and astrophysical parameters.
- Discussing some simple results of the estimation of cosmological and astrophysical parameters with the CMD (CAMELS Multifield Dataset).

3 Methodology

Resumen

Los algoritmos de Machine Learning se suelen clasificar en cuatro grupos: *aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado* y *aprendizaje por refuerzo*. Las redes neuronales se suelen utilizar en el aprendizaje supervisado y no supervisado; sin embargo en este trabajo explicaré el funcionamiento de las redes neuronales supervisadas, que utilizan pares de datos para su entrenamiento. En la Figura 1 se puede ver una estructura típica de una red neuronal. Consta de una capa de entrada, varias *capas escondidas* (capas entre la capa de entrada y salida) y una capa de salida. A través de la ecuación 1 la información de entrada atraviesa la red neuronal activando neuronas con la *función de activación* que tienen intrínsecamente todas las neuronas, para que al final haya uno o varios datos de salida que, en el caso de ser un problema de clasificación será una categoría, y en el caso de ser un problema de regresión será un valor continuo. Esta función de activación se elige dependiendo del problema que se quiera resolver. Una de las más utilizadas es la función ReLU (Rectified Linear Unit).

El objetivo de la red neuronal es que este valor de salida sea el correcto, y para eso necesita entrenarse. Con una base de datos la red calcula los valores de salida a partir de los valores de entrada y los compara con los esperados, calculando la *función error*. Para corregir el posible error compara estos dos valores e intenta minimizar la diferencia ajustando los llamados *pesos*, asociados a las conexiones entre las neuronas de las diferentes capas, utilizando el enfoque de retropropagación junto al descenso de gradiente, una técnica que estima numéricamente dónde la función error genera sus valores más bajos.

Para la inferencia de los parámetros se usarán las Redes Neuronales Convolucionales de Distribución Mixta. Las redes convolucionales son redes cuyas capas difieren de la red neuronal básica. Constan de una capa de entrada, una *capa de convolución*, que simplifica la imagen de entrada con unas matrices pequeñas de pesos fijos llamadas comúnmente *filtros*, una *capa 'pooling'*, que generalmente coge solo el valor máximo de la región que mapea, y una *capa totalmente conectada*, cuyas neuronas están conectadas con todas las neuronas de la capa previa y posterior.

Las redes de distribución máxima, en vez de calcular valores específicos de salida, calculan una distribución de salida y devuelven los parámetros que la definen, su media y su varianza.

Por último se describe la CMD, CAMELS multifield dataset [36], una base de datos que contiene más de 800.000 mapas 2D y 3D generados con simulaciones de alrededor de 2000 universos diferentes. Esta base de datos contiene 3 juegos de simulaciones: IllustisTNG, SIMBA, y N-body; cada uno de ellos tiene mapas que describen diferentes propiedades como la densidad

del gas cósmico, su temperatura, etc.

3.1 Algorithm types

Let's begin by discussing some common Machine Learning algorithms. ML algorithms are generally classified in four main types: *Supervised learning, Unsupervised learning, Semi-supervised learning, and Reinforcement learning* [21].

1. *Supervised Learning*: This technique is based on the inference of a function that assigns an output to an input using a database of input-output pairs [14, as mentioned in [30]]. Some of the Supervised learning tasks are Classification, that labels the data, and Regression, that fits the data [30].

2. *Unsupervised Learning:* It is considered unsupervised because the data used, the input examples, are not class labeled [14]. This type of algorithms learns generic features like patterns in the data and are able to represent the data. This type of learning can be used to automatically detect radio frequency interference from phones, satellites, aircrafts, etc. in Radio Astronomy Observations minimizing the need for human interaction [10].

3. *Semi-supervised Learning:* This type of algorithms work with both labeled and unlabeled data; therefore, they are a combination of the ones mentioned above [14]. The objective of Semi-Supervised Learning is to get better predictions than those inferred from only the labeled data [30]. An interesting application of Semi-supervised Learning in Astronomy is the photometric classification of supernovas by using light curves and a spectroscopically confirmed training set for developing a model capable of predicting the type of the new supernovae [27].

4. *Reinforcement:* This technique allows the machine to learn from the interaction with the environment to evaluate and to perform with maximum efficiency [30, 21]. At each stage of the interaction, the agent receives as input an indication of the current state of the environment; then the agent chooses an action to generate as output, changing the state of the environment. After this transition the *agent* (intelligent program) learns whether this action was a good choice or not through a scalar *reinforcement signal*. The agent should choose a behavior that increases the long-term sum of the reinforcement signal's values [24]. Reinforcement learning has been used to automate the process of calibration of radio telescopes by selecting the best settings for diverse observations, since it is a tedious task to adjust by hand the hyperparameters of the pipelines that process the produced data [43].

3.2 Artificial Neural Networks

In this work we will be focusing on Supervised Learning, particularly on Neural Networks. On supervised Learning, this type of algorithms require labeled data for classification and regression. Let's briefly explain what these are all about.

- **Classification.** In machine learning it is considered a supervised learning approach because the model needs a set of data to train itself in order to predict a *discrete* class label [14, as mentioned in [30]].

- **Regression.** Regression analysis predicts a *continuous* output value Y_m based on the training dataset $\{(\mathbf{x_k}, \mathbf{y_k})\}$ [14, as mentioned in [30]].

A neural network pretends to mimic the way the neurons work in the human brain and nervous system through a set of algorithms.

3.2.1 Structure

In 1943 McCulloch and Pitts created a model of a neuron that depending on the total weighted input calculated out of the different inputs received from other neurons, activated or remained inactive. This weight is the analogue of the strength of a synapse[1] in the nervous system [20]. A Neural Network consists of an input layer of neurons, some hidden layers of neurons, and a layer of output neurons, where the connections between the different neurons are the *weights*, described above.

The analytical expression of the outputs of each neuron is [39]:

$$h_i^l = f\left(\sum_{j=1}^{J} V_{ij}^l \cdot h_j^{l-1} + T_i^l\right) \qquad (1)$$

where h_i^l is the output of the neuron i in the layer l, $f()$ is the activation function, V_{ij}^l are the weights of the connections to the neuron i in layer l, h_j^{l-1} are the outputs of the nodes in the layer $l-1$, and T_i^l are the threshold terms of the hidden neurons in layer l.

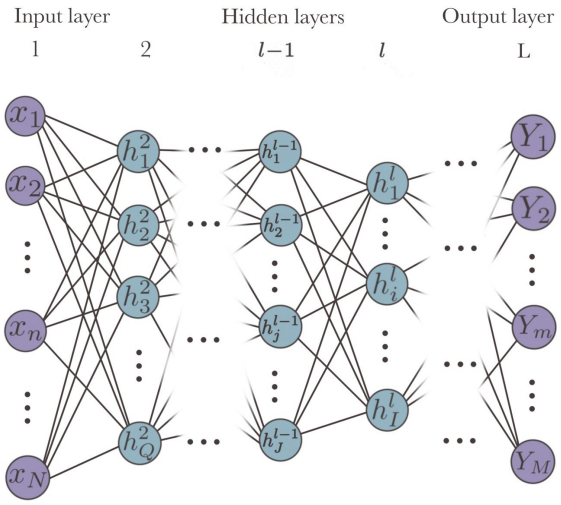

Figure 1: Representation of the used notation for a multi-layer neural network. From left to right: Input Layer, Hidden Layers and Output layer.

Let's briefly explain what each of these elements is:

- **Input layer:** receives features as input and brings the initial data into the system.
- **Output layer:** contains all the output values.
- **Hidden layers:** layers between the input and output layers. These layers compute all of the features entered through the input layer and send the results to the output layer.
- **Weights:** the effect the signal has on the neuron. Each connection has a different weight.
- **Activation function:** calculates a weighted total and then adds a bias to it in order to determine whether a neuron should be activated or not. The activation function's goal is to enter non-linearity into a neuron's output.

[1] Structure that permits a neuron (or nerve cell) to pass an electrical or chemical signal to another neuron.

Some examples will be depicted in Section 3.2.3.

- **Output of the unit h_i:** calculated by Equation 1. It is the output value of each neuron.
- **Threshold value:** also called *bias* or *offset*, is the cut off value of the activation function, the minimum required value to activate the neuron. Each neuron usually has a different Threshold value.

For the neural network in Figure 2 we have the x_1, x_2, x_3 and x_4 inputs and there is a connection between each of the neurons in the consecutive layers.

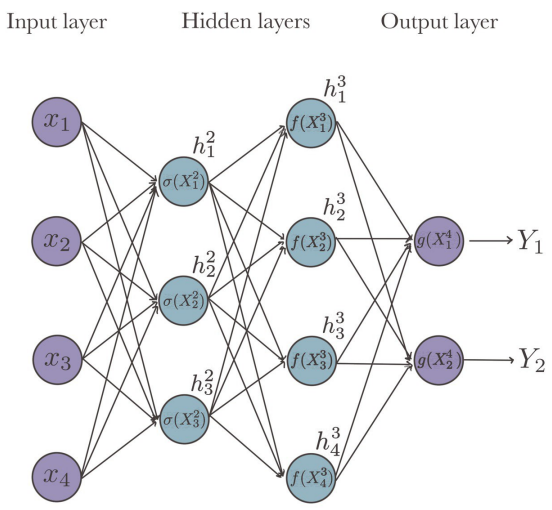

Figure 2: Example of a fully-connected four-layer neural network with activation functions $\sigma()$, $f()$ and $g()$.

As said before each connection will have a weight associated to it. For instance, the connections between the Input Layer and the first Hidden Layer will have the V_{ij}^2 weights associated to them, where i is the index of the neurons in the first hidden layer ($i = 1, 2, 3$) and n the index of the inputs ($n = 1, 2, 3, 4$). Therefore the weights associated with these first connections will be $V_{11}^2, V_{12}^2, V_{13}^2, V_{14}^2, V_{21}^2, V_{22}^2, V_{23}^2, V_{24}^2, V_{31}^2, V_{32}^2, V_{33}^2$ and V_{34}^2; being V_{11}^2 the weight associated with the connection between the input x_1 and the first hidden layer's first neuron, and so on. What would happen then? Well, let's take this first neuron from the first hidden layer.

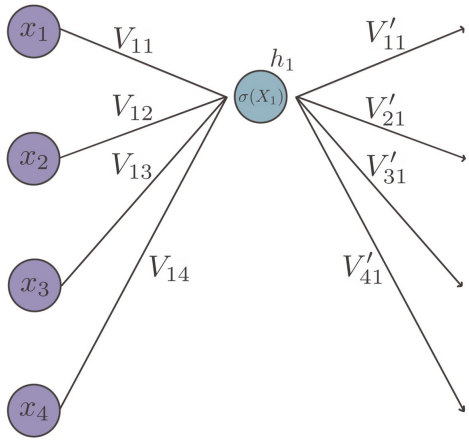

Figure 3: First neuron in the first hidden layer of the neural network represented in Figure 1.

The output of this blue neuron with index $i = 1$ would be:

$$h_1^2 = f(X_1^2) \tag{2}$$

where

$$X_1^2 = \sum_{n=1}^{4} V_{1n}^2 \cdot x_n + T_1^2 = V_{11}^2 \cdot x_1 + V_{12}^2 \cdot x_2 + V_{13}^2 \cdot x_3 + V_{14}^2 \cdot x_4 + T_1^2 \tag{3}$$

The output h_1^2 depends on the activation function $f()$.

3.2.2 Training

The basic goal of a neural model is to determine the optimal set of weights so that the network provides an estimation of the desired quantity Y [44]. As mentioned in section 3.1, in a supervised learning algorithm this is achieved by training the model with

input-output data pairs. First of all let's introduce some important terms that will be mentioned in this section [5]:

- The dataset is usually divided in groups to simplify the training process. These groups are called **batches.** The **Batch Size** is the number of samples that are in a batch. It also defines the number of samples that pass through the net before updating the weights.
- The **Epoch number** determines how many times the whole dataset passes through the net.

For example, if:

- The batch size is 64.
- Our dataset has 3000 samples.
- The epoch number is 20.

it will take around 47 batches ($^{3000}/_{64} \approx 47$) to make up one full epoch, and this process will be repeated 20 times.

Loss Function

In the training process the neural network's execution is estimated by using a metric that quantifies the difference between the inferred value and the real value. This is called the *Loss Function*. This *error* can be given by many functions, and depending on the problem it could be more convenient to choose one over the other. Here are some of the most used ones [40]:

- *Square Loss function.* It is one of the most commonly used in Regression problems, it calculates the square of the error between the real value and the inferred value. The gradient of square loss changes, which contributes to the model's quick convergence and excellent accuracy [40]. It is expressed by [44]:

$$E = \frac{1}{2}\sum_{k=1}^{K}\sum_{m=1}^{M}(Y_m(\mathbf{x_k}, \mathbf{V}) - y_{km})^2 \quad (4)$$

- *Absolute Loss function.* Usually used for Regression problems, it is preferred over the Square Loss function when there are outliers because it does not increase rapidly like the Square Loss function does. One drawback of this function is that near 0 it is not smooth[2].

$$E = \sum_{k=1}^{K}|Y_m(\mathbf{x_k}, \mathbf{V}) - y_{km}| \quad (5)$$

- *Cross-Entropy Loss function/Negative Log Likelihood Function.* It is the most used Loss Function for Classification problems. Its expression is [46, 47]:

$$E = -\sum_{k=1}^{K}\sum_{m}^{M} y_{km} \cdot log(f_{SM}(X_{km}^{L})) \quad (6)$$

where $f_{SM}(X_m^L)$ is the SoftMax function, mentioned in Section 3.2.3.

$\mathbf{V}$ are the weights of the neural network, Y_m is the output of the m^{th} output node for input $\mathbf{x_k}$ and y_{km} is the real value associated with $\mathbf{x_k}$. Let's understand this notation a little bit better: Supposing we have a dataset of simulated universes' maps with a K number of input-output pairs $\{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), ..., (\mathbf{x_k}, \mathbf{y_k}), ..., (\mathbf{x_K}, \mathbf{y_K})\}$, $\mathbf{x_k}$ would be the k^{th} map of the dataset and $\mathbf{y_k}$ would be its output vector. The $\mathbf{x_k}$ vector contains all the pixels in the k^{th} map and the $\mathbf{y_k}$ output vector could contain a number of parameters describing many different properties of cosmic gas, for

[2] A function is smooth when it is infinitely differentiable everywhere.

example.

In brief, the networks objective is to find the optimal weights V_{ij} so that this error minimizes.

Back-Propagation and Gradient Descent

In 1986 the **Back-Propagation Algorithm** was proposed by Rumelhart, Hinton and Williams [29, as mentioned in [44]], which along with the **Gradient Descent (GD)** (also called Batch Gradient Descent) optimizes the weights by minimizing the error function.

To get started, small random values of the weights are set. In theory, throughout the training process the whole dataset passes through the net: the error E_k in Equation 12 is calculated for each sample, then the total error E in Equation 11 is calculated by adding all the E_k errors and finally the weights are updated by following the *Gradient Descent* in Equation 7 [44]:

Gradient Descent

$$\mathbf{V} = \mathbf{V} - \lambda \frac{\partial E}{\partial \mathbf{V}} \qquad (7)$$

where λ is the *learning rate*[3], until it reaches the minimum value of the error E. This equation is very intuitive: given the fact that a gradient gives the direction of the fastest **increase** of the function, in order to **minimize** the error function, we must take the negative direction.

We can anticipate that this is not how the optimization goes in practice: there are ways to reduce the training time and simplify the computations. The **Stochastic Gradient Descent (SGD)** is one of the most used algorithms for the optimization task. It greatly simplifies the general gradient descent since it randomly takes one single data sample to optimize the parameters in each iteration: the **batch size of the Stochastic Gradient Descent is 1** [5]. So, instead of calculating the errors of each sample and then adding them up, only one error E_k of a random sample (x_k, y_k) is needed to update the weights [4]:

Stochastic Gradient Descent

$$\mathbf{V} = \mathbf{V} - \lambda \frac{\partial E_k}{\partial \mathbf{V}} \qquad (8)$$

There is also the **Mini-batch Gradient Descent** that instead of implementing gradient updates per sample like SGD does, or per epoch, like GD does, the parameters are updated when all the samples in a batch pass through the net [5]. So if the batch size is 64, then 64 samples will pass through the net and the errors E_k of each sample will be added up to get the error E, with which the weights will be updated:

Mini-batch Gradient Descent

$$\mathbf{V} = \mathbf{V} - \lambda \frac{\partial E_B}{\partial \mathbf{V}} \qquad (9)$$

where

$$E_B = \frac{1}{2} \sum_{k_B=1}^{K_B} \sum_{m=1}^{M} (Y_m(\mathbf{x_{k_B}}, \mathbf{V}) - y_{k_B m})^2 \qquad (10)$$

and K_B is the number of samples in the batch B. Briefly:

[3] It is a parameter that determines how much the model changes every time the weights are updated, meaning that if λ is too small the training process will be very long and if it is big then it could be too short [42].

- Gradient Descent (or Batch Gradient Descent): updates the weights per epoch, after all the samples have passed through the net. Batch size = total number of samples in the dataset.
- Stochastic Gradient Descent: updates the weights after feeding-forward a single random sample. Batch size = 1.
- Mini-batch Gradient Descent: updates the weights per batch, after all the samples in the batch have passed through the net. Batch size = up to the user.

The computation of the *Back-Propagation* for a single training example goes like this: let's suppose we have the Square Loss function

$$E = \frac{1}{2}\sum_{k=1}^{K}\sum_{m=1}^{M}(Y_m(\mathbf{x_k}, \mathbf{V}) - y_{km})^2 \quad (11)$$

Having in mind that for a single training example we have this cost or error function:

$$E_k = \frac{1}{2}\sum_{m=1}^{M}(Y_m(\mathbf{x_k}, \mathbf{V}) - y_{km})^2 \quad (12)$$

where M is the total number of nodes in the output layer; that the output values of the nodes of the l^{th} layer are given by:

$$h_i^l = f(X_i^l) = f(\frac{1}{2}\sum_{j=1}^{J} V_{ij}^l h_j^{l-1} + T_i^l) \quad (13)$$

where h_j^{l-1} and J are the output values and the total number of neurons in the previous layer respectively; and that the output values of the nodes of the output layer are given by:

$$Y_m = f(X_m^L) = f(\frac{1}{2}\sum_{j=1}^{J} V_{mj}^L h_j^{L-1} + T_m^L) \quad (14)$$

where h_j^{L-1} and J are the output values and the total number of neurons in the previous layer respectively.

Computing the back-propagation is basically calculating the derivatives of Equation 12 [44]:

$$\frac{\partial E_k}{\partial V_{ij}^l} = \frac{\partial E_k}{\partial h_i^l} \cdot \frac{\partial h_i^l}{\partial X_i^l} \cdot \frac{\partial X_i^l}{\partial V_{ij}^l} \quad (15)$$

being

$$\frac{\partial h_i^l}{\partial X_i^l} \cdot \frac{\partial X_i^l}{\partial V_{ij}^l} = \frac{\partial f(X_i^l)}{\partial X_i^l} \cdot h_j^{l-1} \quad (16)$$

The first step would be to initialize the gradient descent in the output layer, therefore Equation 15 would become:

$$\frac{\partial E_k}{\partial V_{mj}^L} = \frac{\partial E_k}{\partial Y_m} \cdot \frac{\partial Y_m}{\partial X_m^L} \cdot \frac{\partial X_m^L}{\partial V_{mj}^L} \quad (17)$$

with

$$\frac{\partial E_k}{\partial Y_m} = (Y_m(\mathbf{x_k}, \mathbf{V}) - y_{km}) \quad (18)$$

$$\frac{\partial Y_m}{\partial X_m^L} = \frac{\partial f(X_m^L)}{\partial X_m^L} \quad (19)$$

$$\frac{\partial X_m^L}{\partial V_{mj}^L} = h_j^{L-1} \quad (20)$$

(remember that $Y_m = h_i^L$).

The next step would be propagating this error from layer l to layer $l-1$ and so on [44].

$$\frac{\partial E_k}{\partial h_j^{l-1}} = \sum_{i=1}^{I_l} \frac{\partial E_k}{\partial h_i^l} \cdot \frac{\partial h_i^l}{\partial h_j^{l-1}} \quad (21)$$

Practical example

For instance let's take Figure 2 and let's calculate the "updated" value of weight V_{14}^4

for a single sample, supposing that $g(X_m^L) = sigmoid(X_m^L)$:

$$g(X_m^L) = \frac{1}{1 + e^{-X_m^L}} \tag{22}$$

After some random values for the weights are set, the neural network will give an output. Probably this output will make no sense since it is a random calculation.
In the first iteration it will calculate the error function:

$$E_k = \frac{1}{2}\sum_{m=1}^{2}(Y_m - y_{km})^2 = \\ = E_{Y_1} + E_{Y_2} = \frac{1}{2}[(Y_1 - y_{k1})^2 \\ + (Y_2 - y_{k2})^2] \tag{23}$$

Then in the second iteration it will calculate the impact of the weights in the error (the derivatives) to know how to update the weights [44]. For weight V_{14}^4:

$$\frac{\partial E_k}{\partial V_{14}^4} = \frac{\partial E_k}{\partial Y_1} \cdot \frac{\partial Y_1}{\partial X_1^4} \cdot \frac{\partial X_1^4}{\partial V_{14}^4} \tag{24}$$

where [44]:

1. $$\frac{\partial E_k}{\partial Y_1} = Y_1 - y_{k1} \tag{25}$$

2. $$\frac{\partial Y_1}{\partial X_1^4} = \frac{\partial g(X_1^4)}{\partial X_1^4} = Y_1(1 - Y_1) \tag{26}$$

3. $$\frac{\partial X_1^4}{\partial V_{14}^4} = h_4^3 \tag{27}$$

Therefore we have:

$$\frac{\partial E_k}{\partial V_{14}^4} = (Y_1 - y_{k1}) \cdot [Y_1(1 - Y_1)] \cdot h_4^3 \tag{28}$$

So now we can proceed with the calculation of the "updated" V_{14}^4 weight. Following Equation 7:

$$\boxed{\begin{aligned} V_{14}^{4\ updated} &= V_{14}^4 - \lambda\frac{\partial E_k}{\partial V_{14}^4} = \\ &= V_{14}^4 - \lambda\{V_{14}^4 - (Y_1 - y_{k1})\cdot \\ &\cdot [Y_1(1 - Y_1)] \cdot h_4^3\} \end{aligned}} \tag{29}$$

After doing the same for the weights V_{ij}^4 we would be able to propagate backwards to correct the weights V_{ij}^3 and so on. For V_{43}^3:

$$\frac{\partial E_k}{\partial V_{43}^3} = \frac{\partial E_k}{\partial h_4^3} \cdot \frac{\partial h_4^3}{\partial X_4^3} \cdot \frac{\partial X_4^3}{\partial V_{43}^3} \tag{30}$$

with:

1. $$\frac{\partial E_k}{\partial h_4^3} = \frac{\partial E_{Y_1}}{\partial h_4^3} + \frac{\partial E_{Y_2}}{\partial h_4^3} \tag{31}$$

1.1 $$\frac{\partial E_{Y_1}}{\partial h_4^3} = \frac{\partial E_{Y_1}}{\partial X_1^4} \cdot \frac{\partial X_1^4}{\partial h_4^3} \tag{32}$$

1.2 $$\frac{\partial E_{Y_2}}{\partial h_4^3} = \frac{\partial E_{Y_2}}{\partial X_2^4} \cdot \frac{\partial X_2^4}{\partial h_4^3} \tag{33}$$

and

1.1.1 $$\frac{\partial E_{Y_1}}{\partial X_1^4} = \frac{\partial E_{Y_1}}{\partial Y_1} \cdot \frac{\partial Y_1}{\partial X_1^4} = \\ (Y_1 - y_{k1}) \cdot [Y_1(1 - Y_1)] \tag{34}$$

1.1.2 $$\frac{\partial X_1^4}{\partial h_4^3} = V_{14}^4 \tag{35}$$

2. $$\frac{\partial h_4^3}{\partial X_4^3} = \frac{\partial f(X_4^3)}{\partial X_4^3} \tag{36}$$

3. $$\frac{\partial X_4^3}{\partial V_{43}^3} = h_3^2 \tag{37}$$

Putting it altogether:

$$\frac{\partial E_k}{\partial V_{43}^3} = \{(Y_1 - y_{k1}) \cdot [Y_1(1 - Y_1)] \cdot V_{14}^4 + \\ + \frac{\partial E_{Y_2}}{\partial h_4^3}\} \cdot \frac{\partial f(X_4^3)}{\partial X_4^3} \cdot h_3^2 \tag{38}$$

(To develop $\frac{\partial E_{Y_2}}{\partial h_4^3}$ we would need to calculate the derivatives implied in the correction of V_{24}^4).

$$\boxed{V_{43}^3{}_{\ updated} = V_{43}^3 - \lambda \frac{\partial E_k}{\partial V_{43}^3}} \tag{39}$$

The whole process can be can be summed up as follows:

1. Calculate the error with random weights.
2. Optimize weights.
   (a) Calculate the corrected weights V_{ij}^4.
   (b) Calculate the corrected weights V_{ij}^3.
   (c) Calculate the corrected weights V_{ij}^2.
   (d) Calculate the corrected weights V_{ij}^1.
3. Compute the output.
4. Determine the error.
5. Repeat steps 2, 3 and 4 until getting the minimum error.

For this explanation we have taken the *Square loss function*, as it is one of the most used error functions, mainly in Regression problems. However, depending on the task there are some other loss functions which may be more suitable.

Vanishing and Exploding Gradients

Training a neural network can be very burdensome. This is mainly due to the **Vanishing gradient** and the **Exploding gradient** [25].

The vanishing gradient occurs when the derivative $\partial E_k / \partial V_{ij}^l$ in Equation 8 becomes very small in the first layers. This derivative depends on the product if multiple terms. If these terms are numbers smaller than one, when multiplying each other the derivative will become even smaller. Taking this into account, when updating weights in the first layers, the derivative $\partial E_k / \partial V_{ij}^l$ will be *vanishingly* small, because due to the back-propagation process it will depend on many terms that are smaller than one. This value of the derivative is in turn multiplying the learning rate, which is as well a very small number. The fact that the derivative is small and that the learning rate is small results in an almost not updated weight, because the term that causes the weight to change is vanishingly small. Equation 15 depends a lot on the activation function just like equations 16 and 18 reflect; therefore, the vanishing gradient problem may appear if the activation function is a constant, or if it is a low-slope function, since its derivative will be a small number.

The exploding gradient problem is very similar to the vanishing gradient one, but it occurs when the terms in Equation 15 are larger than 1. This means that when updating a weight in the first layers, instead of barely changing its value as it happens with the vanishing gradient problem, the net will be changing it too much, and it probably won't ever reach its optimal value.

One solution to these problems is choosing a proper Activation Function like ReLU or LReLU, which will be mentioned in Section 3.2.3.

Batch Normalization

Another way of reducing the learning time is normalizing the input data. Not only that, but normalizing after each layer

is also recommended. This is called **Batch Normalization** [17]. What Batch Normalization does is normalize the output (or input) values of each layer, either before the activation function (what in this report has been notated as X_i^l) or after the activation function (h_i^l). For the sake of the explanation we are going to assume that the net will normalize the values after the activation function. Below are the steps the net follows when doing Batch Normalization:

1. Normalization by [17]:

$$\hat{h}_i^l = \frac{h_i^l - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \tag{40}$$

where ϵ is the *noise* in case $\sigma_l^2 = 0$, and μ_l and σ_l^2 are the mean value and variance of the values in that layer [17]:

$$\mu_l = \frac{1}{I}\sum_{i=1}^{I} h_i^l \tag{41}$$

$$\sigma_l^2 = \frac{1}{I}\sum_{i=1}^{I}(h_i^l - \mu_l)^2 \tag{42}$$

What this operation does is make the distribution of values have a mean value equal to 0 ad a variance equal to 1: $\sum_i^I \hat{h}_i^l = 0$ and $^1/_I \sum_i^I (\hat{h}_i^l)^2 = 1$.

2. Sometimes it is of interest to have other values for the mean and variance, so the net can scale and shift these normalized values with the Batch Normalization Transform [17]:

$$H_i^l = \gamma \hat{h}_i^l + \beta \tag{43}$$

where H_i^l is the final normalized output of neuron i in layer l, and γ and β are two trainable parameters [17].

When working with normalized values, the neural network uses these new values of H_i^l when doing the back-propagation, and the parameters γ and β as well as the weights and biases update per batch.

3.2.3 Activation functions

The activation function affects greatly the training of the network and hence it is crucial to choose the optimal activation function for the case at hand. The need of this function relies on the fact that without it the output signal would simply be a linear function, easy to compute, but unable to learn and recognize complex mappings from data. This functions also need to be differentiable so that we can apply the back-propagation process to optimize the weights [31].

Therefore, for an activation function to be successful it should satisfy these conditions:

- They should be non-linear.
- They should be differentiable.

The most commonly used functions are:

1. Sigmoid function
2. $Tanh$ function
3. Rectified Linear Unit (ReLU) function
4. SoftMax function

Sigmoid Activation Function

It is widely used as it is a non-linear function and it is also continuously differentiable [31]. When the X values are between 2 and +2, it can be seen that the Y values are quite steep. As a result, any change in X within this range will result in a considerable change in Y, hence the function has a propensity to bring the Y values to either end of the curve, improving the classifier by differentiating predictions in an evident way [15, as cited in [33]]. One drawback of Sigmoid Activation Functions is that their output values are

always positive. One way of solving this is re-scaling the function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{44}$$

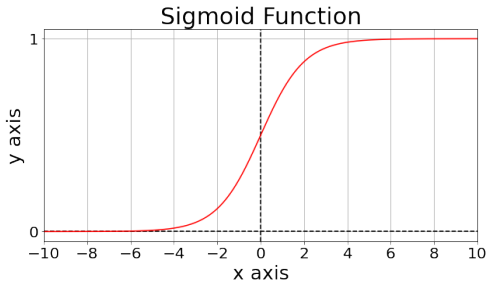

Figure 4: Sigmoid Activation Function.

Tanh Activation Function

The *tanh* function resembles the sigmoid function, however it is symmetric about zero. As a result, the outputs from previous layers will have different signs and that is one of the reasons why the *tanh* function is preferred over the sigmoid function, because they have gradients that are not constrained to varying in a certain direction [31]. The *tanh* function can be rewritten as a function of the sigmoid function:

$$f(x) = tanh = 2 \cdot sigmoid(2x) - 1 \tag{45}$$

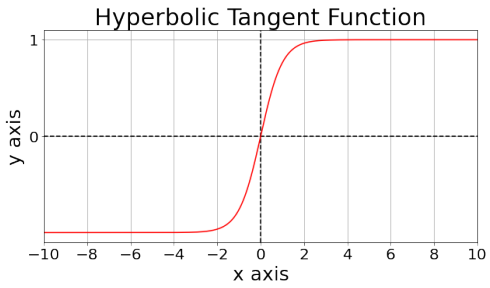

Figure 5: Hyperbolic Tangent Activation Function.

Rectified Linear Unit (ReLU) Activation Function

The Rectified Linear Unit (ReLU) is currently the most used activation function. Its appeal lies on the fact that not all neurons are activated at the same time [31] and that offers diversity in the activation of the neurons.

$$f(x) = max(0, x) = \begin{cases} 0 \ , & x < 0 \\ x \ , & x \geq 0 \end{cases} \tag{46}$$

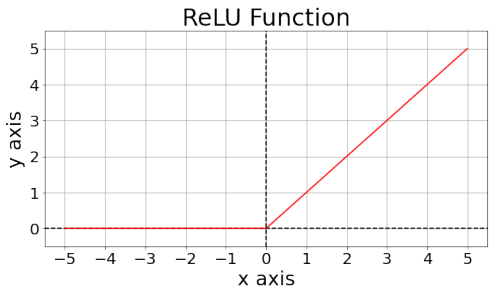

Figure 6: Rectified Linear Unit Activation function.

Its derivatives are quite used as well, like the Leaky ReLU (LReLU), in which the negative values of x have very small values of $f(x)$ [31] instead of being 0, or the Parametrized ReLU, where the slope is also very similar to the LReLU he slope of $f(x)$ for negative values of x is also a trainable variable [31].

SoftMax Activation Function

The Softmax function is a sigmoid function combination that returns probabilities of each class [31]. It is used in multi-classification problems and usually appears in the output layers of deep learning architectures [19, as mentioned in [23]]. Its expression is [31]:

$$f(X_i) = \frac{e^{X_i}}{\sum_{p=i}^{P} e^{X_p}} \tag{47}$$

where X_i is the weighted sum of the inputs plus the bias of the neuron i, X_p are the

weighted sums of the inputs plus the biases of each of the neurons in the same layer as the neuron i and P is the total number of neurons in that layer.

3.2.4 Neural Network Architectures

As in any other Machine Learning algorithm, there are many types of architectures. Here are explained some of them:

Perceptron

A Perceptron is the basic unit of a Neural Network. It consists of 3 layers, a layer of inputs, a neuron and an output. It is only a binary classifier, this is, a linear classifier as it is only able to group data into two classes [11].

Practical example

For a better understanding let's make an example [18], whether we should buy an ebook or not. A Perceptron can be described with four basic constituents: inputs, weights, a threshold or bias and an output. The analytic expression of the output Y for this example could be something like the following:

$$Y = f\left(\sum_{n=1}^{N} V_n \cdot x_n + T\right) = f\left(V_1 \cdot x_1 + V_2 \cdot x_2 + V_3 \cdot x_3 + T\right) \tag{48}$$

where the *activation function* $f(x)$ for this case is the sign function:

$$f(X) = \begin{cases} 1 & if \quad \sum_{n=1}^{N} V_n \cdot x_n + T \geq 0 \\ 0 & if \quad \sum_{n=1}^{N} V_n \cdot x_n + T < 0 \end{cases} \tag{49}$$

We should have three factors:

1. If it's more environmentally friendly.
   - Yes = 1.
   - No = 0.
2. If it has more durability.
   - Yes = 1.
   - No = 0.
3. If it saves money.
   - Yes = 1.
   - No = 0.

So for our example we will then have the following inputs:

1. It is environmentally friendlier than a conventional book $\rightarrow x_1 = 1$.
2. It is not more durable, technological items tend to get obsolete $\rightarrow x_2 = 0$.
3. It is comfortable, it weights less than a paperback $\rightarrow x_3 = 1$.

The weights would translate to the importance of each of these factors:

1. You are an environmentalist $\rightarrow V_1 = 5$.
2. You have loads of money $\rightarrow V_2 = 3$.
3. You are strong and carry big bags $\rightarrow V_3 = 1$.

If we assume a *threshold* value of 5 ($bias = -5$) and take all this information into consideration, we get:

$$Y = V_1 \cdot x_1 + V_2 \cdot x_2 + V_3 \cdot x_3 + T = \\ = 5 \cdot 1 + 3 \cdot 0 + 1 \cdot 1 - 5 = 1 \tag{50}$$

We then get that the output of the activation function $f(x)$ is 1. This means that we will buy an ebook.

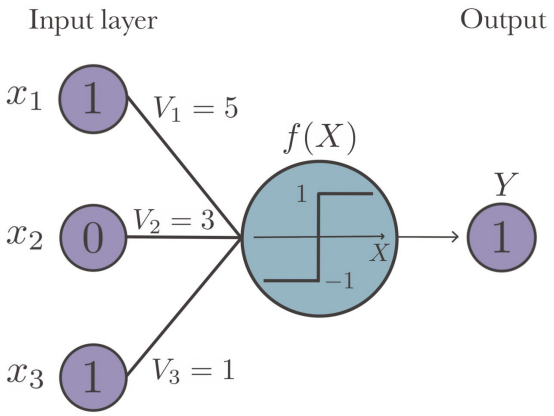

Figure 7: Example.

Multi-layer Perceptron (MLP)

A Multi-layer Feed-forward network is a neural network with multiple layers and units each unit of layer l connects directly to a node in the next layer $l+1$ and receives inputs from other node or nodes in layer $l-1$ [45]. They belong to the *Feed-forward Networks* group, when data is rigidly fed forward from input to output units, there are no connections from output units to other units in previous layers [44, 45]. The functioning and training of this kind of networks is basically what has been explained in previous sections.

Convolutional Neural Network

The convolutional network differs from a standard ANN in the layers they are constituted by, the CNNs have convolutional layers, pooling layers and fully connected layers, which will be explained later on. The input data is usually two-dimensional so it is commonly utilized in a wide range of applications, including image and video recognition, image processing and classification, medical image analysis, natural language processing, and so on [30]. Let's see what these layers do:

- *Input Layer.* As said before, convolutional networks take 2D grids as input. This means that the input of a convolutional network will be something like $[N, M, Z]$, where N and Y are the width and height (in pixels) of the image respectively and Z the depth if the image, this is, the color channels RGB [1].
- *Convolutional Layer.* The objective of this layer is to somehow simplify the input image via what are called **filters**. This filters are essentially small matrices with fixed weight values that map the input image and create much simpler feature map. Depending on what these values are the filter will detect different features like shapes, edges and more.

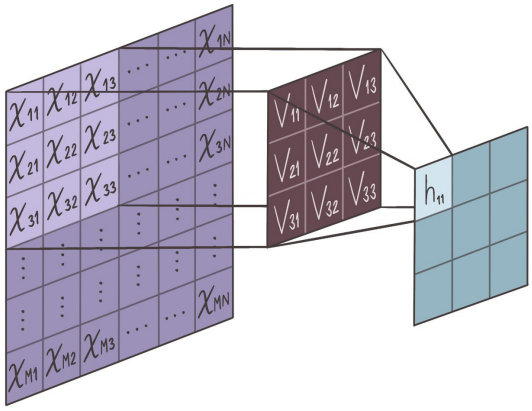

Figure 8: Convolutional Layer. The purple layer is the input image, the brown layer is the filter and the blue layer is the feature image.

It is usual to have more than one convolutional layer in order to make it more beneficial [1]. There is this concept called **stride** that determines how many steps the filter moves when mapping the image. This relation is given by:

$$O = 1 + \frac{N - F}{S} \tag{51}$$

where O is the output size, N is the width or height of the square input image,

F the filter size and S the stride [1].

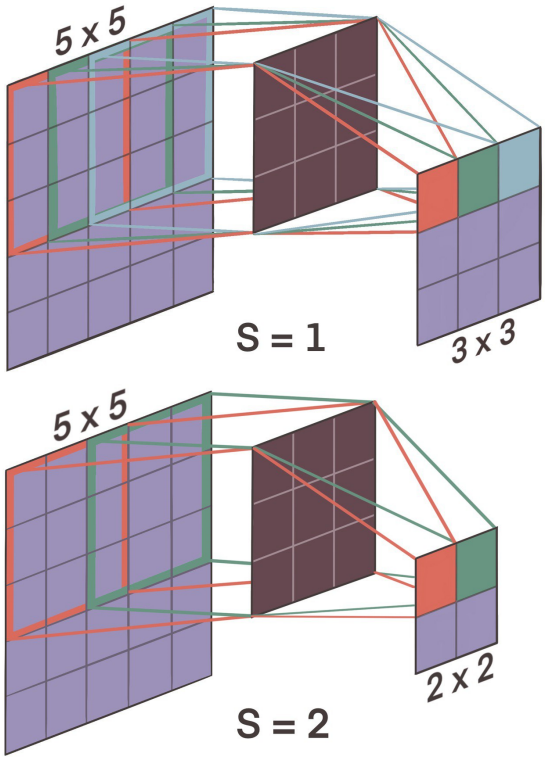

Figure 9: Examples of how a convolutional layer works with different strides. On top, the stride is 1, so the filter traverses per slide 1 column of pixels. In the bottom the stride is 2, so the filter traverses per slide 2 columns.

Due to this mapping the information in the borders of the image is lost. A solution to this problem is what is called **zero-padding**. This involves surrounding the image with 0s. An additional benefit of this is that with zero-padding the output image size can be controlled. For instance for a stride $S = 1$ we can set the zero-padding to

$$P = \frac{F-1}{2} \tag{52}$$

if we want the output size to be the same as the input size [9].

The output in the next layer will then be [1]:

$$h_{ij}^{l} = f(X_{ij}^{l}) \tag{53}$$

where

$$X_{ij}^{l} = \sum_{a=0}^{F-1}\sum_{b=0}^{F-1} V_{ab} \cdot h_{(i+a)(j+b)}^{l-1} \tag{54}$$

where f is the activation function (usually ReLU in Figure 6), F is the dimension of the filter, V_{ab} are the fixed weights of the filter and $h_{(i+a)(j+b)}^{l-1}$ the outputs if the previous layer.

For example in Figure 8 the *filter* colored in brown would map the input grid in purple and create a simpler feature map in blue, h_{11} being:

$$\begin{aligned} h_{11} = f(&V_{11} \cdot x_{11} + V_{12} \cdot x_{12} + V_{13} \cdot x_{13} + \\ &V_{21} \cdot x_{21} + V_{22} \cdot x_{22} + V_{23} \cdot x_{23} + \\ &V_{31} \cdot x_{31} + V_{32} \cdot x_{32} + V_{33} \cdot x_{33}) \end{aligned} \tag{55}$$

- *Pooling Layer.* In this step the network reduces the resolution of the image and the amount of parameters and computation. One of the most commonly used Pooling methods is **Max-Pooling**, which returns the maximum value of the region [1]. The most used pooling layers have a 2x2 size and $S = 2$.
- *Fully-connected Layer.* Lastly the neurons of this layer are fully-connected to all the neurons on the previous and the output layer. It is the part where the neural network spends most of its time training, as it is a part with many parameters.

Convolutional Neural Network have been used in Astrophysics a number of times; for instance the detection of the presence of fresh impact craters on planetary bodies [38] or analysis of strong gravitational lenses [16].

Mixture Density Neural Networks

The Mixture Density Neural Networks are a combination of a Feed-forward network and a mixture model [3]. The main difference with the neural networks mentioned above is that instead of computing output values, Mixture Density networks calculate probability distributions. The output's probability density (Fig. 10), also called *posterior*, is then given by [3]:

$$p(\mathbf{y}|\mathbf{x}) = \sum_{m}^{M} \alpha_m(\mathbf{x})\phi_m(\mathbf{y}|\mathbf{x}) \qquad (56)$$

where M is the total number of mixture components, are $\alpha_m(\mathbf{x})$ are the weights of the mixture components given by the SoftMax function, which are called the *prior* probabilities, and $\phi_m(\mathbf{y}|\mathbf{x})$ the probability distribution of $\mathbf{y}$ given a specific $\mathbf{x}$ for the m^{th} component of the mixture [3]. If we take these $\phi_m(\mathbf{y}|\mathbf{x})$ functions as Gaussians of the form [3]:

$$\phi_m(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{c/2}\sigma_m(\mathbf{x})^c} exp\{-\frac{|\mathbf{y} - \mu_\mathbf{m}(\mathbf{x})|^2}{2\sigma_m(\mathbf{x})^2}\} \qquad (57)$$

where σ_m and μ_m are the variance and mean values of the m^{th} component and c is the number of outputs the net would have if we used it in the conventional way.

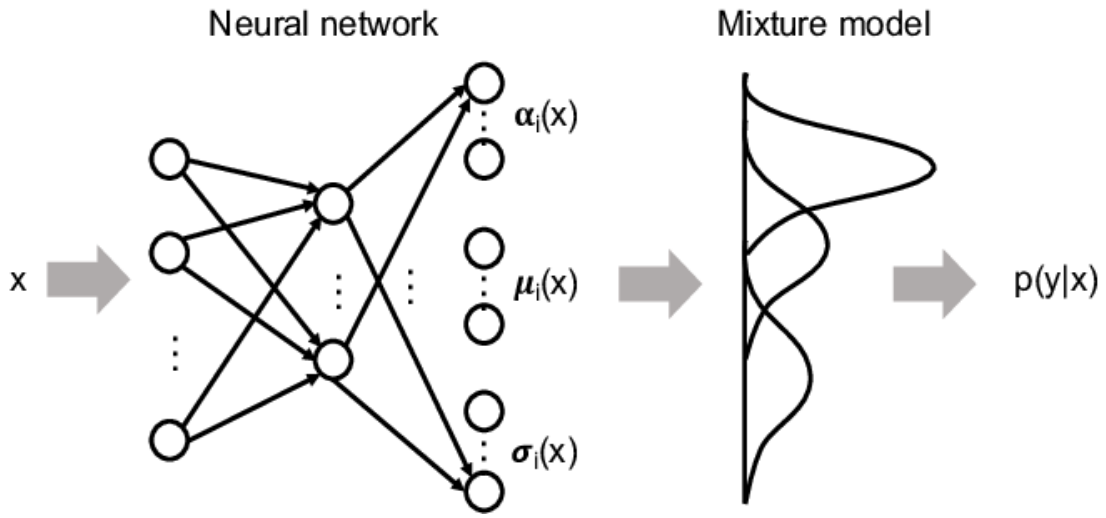

Figure 10: Representation of a Mixture Density Network [37]. The net takes as input a vector $\mathbf{x}$ and returns as an output three parameters for each component of the mixture model,α_m, σ_i and μ_m with which it constructs the mixture distribution, $p(\mathbf{y}|\mathbf{x})$.

3.3 Simulation-based Inference

To study interesting events numerous scientific fields have created elaborate simulations. These simulations become handy when dealing with complex relationships between inputs and outputs, i.e. when the objective is to estimate a value from an input (for example an image) without the analytic expression that relates them.Instead of finding this relationship, which could be a very complex task, we can *simulate* the relation: many input-output pairs can be simulated, and these simulated pairs will be the inputs of our neural network, which then will be able to estimate the value for any

input.

Mind the difference between the simulations and the inference: the **simulation** process creates many input-output pairs with the same relation. The **inference** process makes use of these simulation pairs to create a model which is able to estimate an output from any input.

3.4 The cosmological model and its parameters

Cosmology is the science that studies the universe as a whole and it is mostly an observational discipline, that is, it needs to rely on theoretical models to understand the obtained data. Electromagnetic radiation and the observable portion of the universe are the primary sources of information, with which many projects like Euclid[4], DESI[5], and Roman[6] are trying to resolve some of the biggest question marks in our present cosmology, like: *What is the nature of dark energy?*

The current accepted model that tries to describe the behaviour and evolution of our Universe is **The ΛCDM (Cold Dark Matter) Model**, also called **The standard model**. According to this model the Universe originated from the Big Bang, is composed by radiation, baryonic matter, dark matter and dark energy, is spatially flat, homogeneous and isotropic to a great degree [28, 32]. This model derived from General Relativity equations is determined by a specific collection of cosmological parameters:

- Matter density parameter Ω_m: specifies the total matter density, including the baryonic matter and the dark matter [22].
- Baryon density parameter Ω_b: determines the density of the normal baryonic matter.
- Hubble parameter $H(z)$: controls the time-dependent expansion of the universe, and depends on the red-shift z. The value of the Hubble parameter now is given by the Hubble constant H_0 [22].
- Amplitude of fluctuations on scales of $8h^{-1}Mpc^{-1}$ σ_8: it measures the amplitude of the mass density fluctuations [41].

Previous studies like Planck [26] and WMAP [32] have determined the values of some of these cosmological parameters studying the CMB (Cosmic Microwave Background), and using computer-based statistics such as weak lensing, which is a non-observable light deflection effect due to large masses [7], or clustering of galaxies [2], for example.

In this report we are going reproduce some results of Villaescusa-Navarro et al. [35] to explore whether it is possible to estimate the values of two of these cosmological parameters: Ω_m and σ_8, by using other baryonic fields and scales obtained from the CAMELS Multifield Dataset [36] and Mixture Density networks, as well as the astrophysical constants that determine some astrophysical effects such as supernova feedback and Active Galactic Nuclei (AGN): A_{SN1}, A_{SN2}, A_{AGN1} and A_{AGN2}.

[4] https://www.euclid-ec.org/
[5] https://www.desi.lbl.gov/
[6] https://roman.gsfc.nasa.gov/index.html

3.5 The CAMELS Multifield Dataset

For the second half of this report I would like to apply the theory explained above and infer some cosmological and astrophysical parameters using Mixture Density CNNs. The input vectors given to the neural network will be a collection of 2D maps and associated parameters which have been provided by the *Cosmology and Astrophysics with MachinE Learning Simulation (CAMELS) Multifield Dataset (CMD)*. To explain this dataset I am going to summarize the paper *"The CAMELS Multifield Dataset: Learning the Universe's Fundamemtal Parameters with Artificial Intelligence"* by Francisco Villaescusa-Navarro et al. [36].

This set of data contains 70 Terabytes of 2D and 3D maps from about **2000 different simulated universes** at various cosmic epochs, generated from Gravitational N-body and hydrodynamic simulations. Each 2D map covers an area of $(25h^{-1}Mpc)^2$, which is equivalent to $(7.714 \cdot 10^{20}h^{-1}Km)^2$, that is represented in a square of 256 x 256 pixels. Each map has a vector of 2 or 6 parameters associated to it: all data has two **cosmological parameters**, and only the data from hydrodynamic simulations has another 4 **astrophysical parameters**. As said before the CMD was created with data from CAMELS. Each of the CAMEL Simulation belongs to one of the following subgroups:

- *IllustrisTNG.* They contain magneto-hydordynamic simulations.
- *SIMBA.* They contain hydrodynamic simulations.
- *N-body.* They contain gravitational simulations.

We have mentioned before that the simulations of these 3 suites all have two cosmological parameters:

- Ω_m the matter density parameter [34].
- σ_8 the galaxy fluctuation parameter [34].

Simulations in SIMBA and IllustrisTNG have as well 4 astrophysical parameters associated to them which, contrary to the cosmological parameters, describe different properties in each of the simulation types.

Therefore, making reference to what was explained in Section 3.3, in this case the simulated input-output pairs would be the maps-parameter pairs, with which the neural network will be trained in order to estimate some cosmological and astrophysical parameters.

3.5.1 IllustrisTNG

This suit contains 195.000 2D maps divided in 13 groups: Mgas, Vgas, T, P, Z, HI, ne, B, MgFe, Mcdm, Vcdm, Mstar and Mtot. These will be discussed further later on. In these simulations the astrophysical parameters hold these meanings:

- A_{AGN1} specifies the energy released per unit of black hole accretion rate.
- A_{AGN2} determines the ejection speed for the kinetic mode of black hole feedback.
- A_{SN1} gives information about the energy emitted per unit of star formation rate.
- A_{SN2} controls the speed of galactic winds.

3.5.2 SIMBA

In this group there are collected 180.000 2D maps divided in 12 groups: Mgas, Vgas, T, P, Z, HI, ne, MgFe, Mcdm, Vcdm, Mstar and Mtot. The definitions of the astrophysical parameters for this suit are below:

- A_{AGN1} determines the momentum flux of kinetic outflows in quasar and jet-mode AGN feedback relative to the black hole accretion rate.
- A_{AGN2} specifies the speed of the jet-mode black hole feedback.
- A_{SN1} controls the mass loading factor of galactic winds relative to scalings derived from the FIRE simulations[7].
- A_{SN2} gives information about the speed of these galactic winds.

3.5.3 N-body

The data contained in the N-body simulations is not affected by astrophysical effects —*e.g.*, AGN and supernova feedback. This suit contains 30.000 2D maps divided in two groups: 'Mtot' for IllustrisTNG and 'Mtot' for SIMBA, and therefore it has double the amount of maps that SIMBA or IllustrisTNG has representing the total matter mass density.

3.5.4 Fields

Here are the fields of the different simulations:

- **Gas Density.** This field pictures the spatial distribution of the cosmic gas' density. In each pixel is allocated the density of the gas in that spot. Its prefix is 'Mgas'.

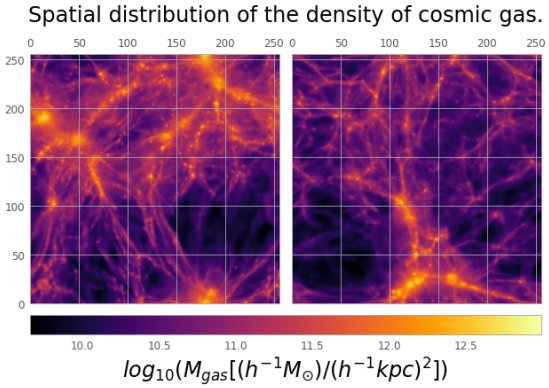

Figure 11: Gas density map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Gas Velocity.** It represents the spatial distribution of the modulus of the peculiar velocity vector of cosmic has $v_g = |\vec{v_g}|$. In each pixel is allocated the mass-weighted modulus $|\vec{v_g}|$ of all gas particles contributing in that spot. Its prefix is 'Vgas'.

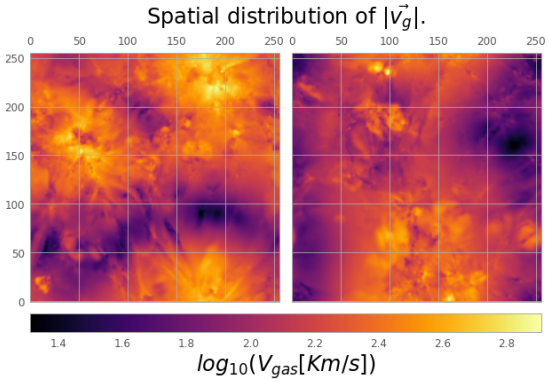

Figure 12: Gas velocity map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Gas Temperature**. This field illustrates the temperature of the cosmic gas in that area. Each pixels stores the mass-weighted temperature of cosmic gas

[7] https://fire.northwestern.edu/

of all the particles contained in that pixel. Its prefix is 'T'.

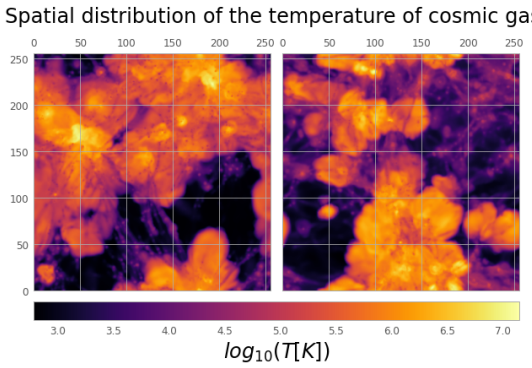

Figure 13: Gas temperature map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Gas Pressure.** The spatial distribution of the cosmic gas' pressure is represented. Each pixel contains the mass-weighted pressure of cosmic gas from all particles contributing to that spot. Its prefix is 'P'.

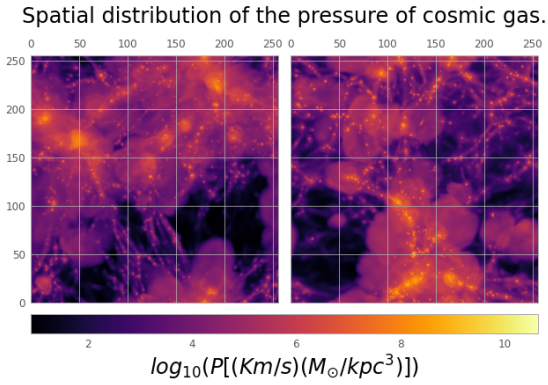

Figure 14: Gas pressure map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Gas Metallicity.** It represents the spatial distribution of the metallicity[8] of the cosmic gas. In each pixel is allocated the mean metallicity of all the gas particles in that spot. Its prefix is 'Z'.

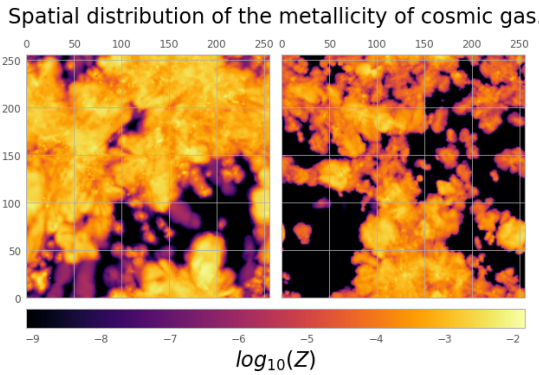

Figure 15: Metallicity map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Neutral Hydrogen Density**. The spatial distribution of the neutral hydrogen's density. Its prefix is 'HI'.

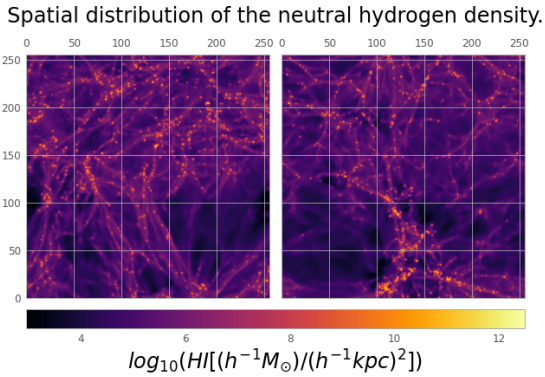

Figure 16: Neutral hydrogen density map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Electron number density.** Illustration of the spatial distribution of the density of electrons. Its prefix is 'ne'.

[8] $Z = {}^{M_{metal}}/_{M_g}$, M_{metal} being the mass in metals (in astronomy elements heavier than helium and hydrogen) and M_g being the total gas mass.

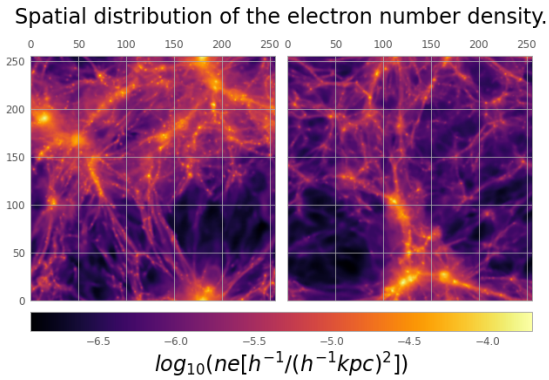

Figure 17: Electron number density map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

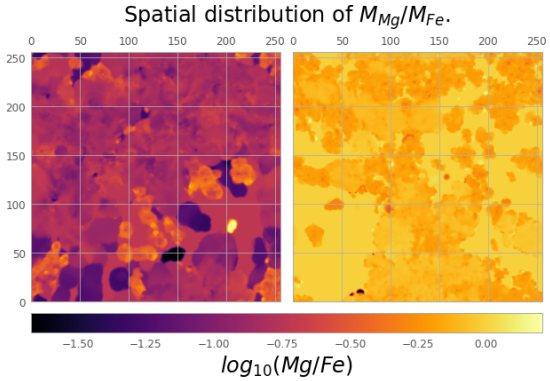

Figure 19: Magnesium over Iron ratio map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Magnetic Fields.** This field pictures the spatial distribution of de modulus of the magnetic field $\vec{B}$ of the cosmic gas, $|\vec{B}|$. Its prefix is 'B'.

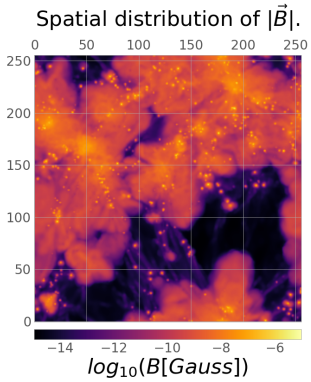

Figure 18: Magnetic fields map nº 1000 of the IllustrisTNG simulations.

- **Magnesium over Iron ratio.** Representation of the ratio between the masses of magnesium and iron of the cosmic gas. Its prefix is 'MgFe'.

- **Dark matter density.** It represents the dark matter density's distribution. Its prefix is 'Mcdm'.

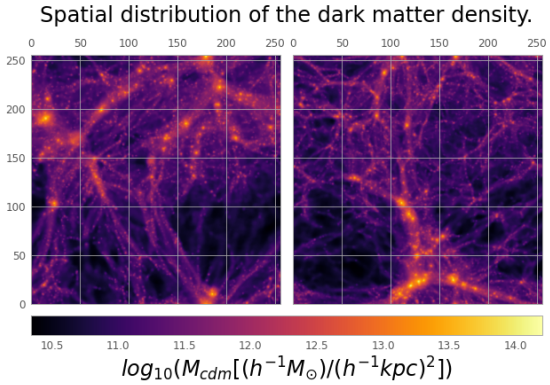

Figure 20: Dark matter density map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Dark matter velocity.** Same as the Gas Velocity but for dark matter. These maps picture the peculiar velocity modulus of of the dark matter $|\vec{v_{dm}}|$. Its prefix is 'Vcdm'.

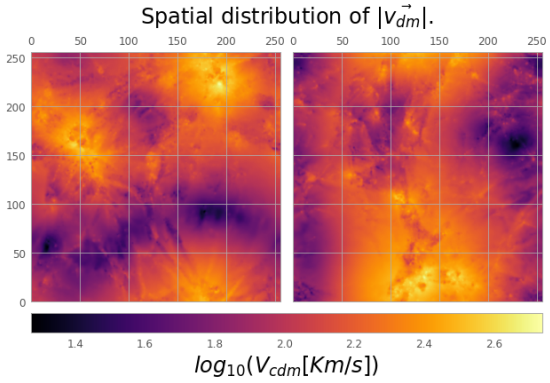

Figure 21: Dark matter velocity map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Stellar Mass Density**. It illustrates the spatial distribution of the density of the stellar mass. Its prefix is 'Mstar'.

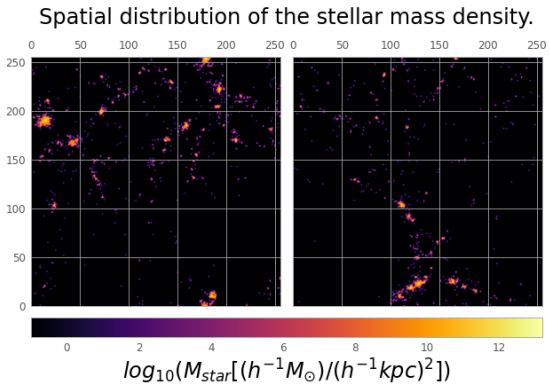

Figure 22: Stellar mass density map nº 1000 of the IllustrisTNG (left) and SIMBA (right) simulations.

- **Total matter mass**. Representation of the spatial distribution of the total matter[9] mass density. Its prefix is 'Mtot'.

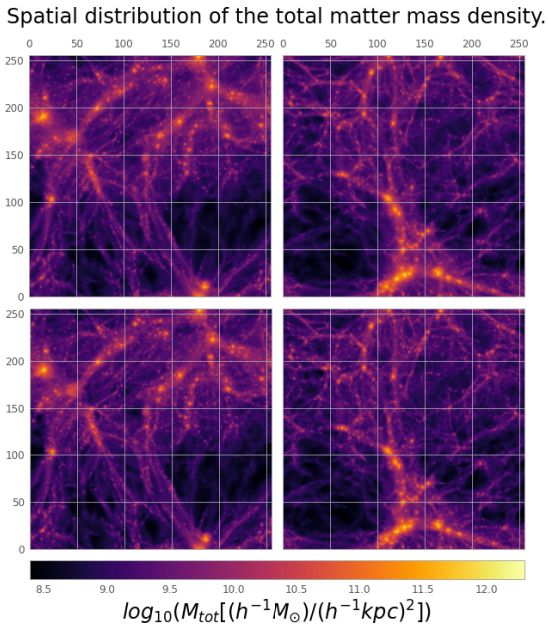

Figure 23: Total matter mass density map nº 1000 of the IllustrisTNG (left), SIMBA (right) simulations and their Nbody (bottom) match respectively.

4 Discussion of results

Resumen

En este apartado se resumen los resultados obtenidos con una red neuronal convolucional de distribución mixta y la arquitectura de ésta. Está constituida por varias capas convolucionales con varios filtros de distintos tamaños, capas de 'MaxPool' y capas normalizadoras.

Se discute que los resultados no han sido los óptimos probablemente por la utilización de pocos mapas para el entrenamiento de la red. También se comparan los resultados obtenidos con aquellos logrados por Villaescusa-Navarro et al. (2021) [35].

[9]Total matter is the sum of gas, stars, dark matter and black holes.

In this section we will be discussing the cosmological and astrophysical parameters' estimation results obtained with the IllustrisTNG Temperature 2D maps.

The used neural network is a Mixture Density Convolutional network, which implements the characteristics of both of these structures. It has the following structure:

- Input layer: 256 x 256 x 1 images.
- Convolutional layer with 16 filters 4 x 4 and a padding that makes the output have the same dimensions as the input.
  - **Input:** 256 x 256 x 1 images.
  - **Output:** 256 x 256 x 16 images.
- Batch normalization.
  - **Input:** 256 x 256 x 16 images.
  - **Output:** 256 x 256 x 16 images.
- MaxPooling Layer 2 x 2 with stride 2.
  - **Input:** 256 x 256 x 16 images.
  - **Output:** 128 x 128 x 16 images.
- Convolutional layer with 32 filters 3 x 3 and a padding that makes the output have the same dimensions as the input.
  - **Input:** 128 x 128 x 16 images.
  - **Output:** 128 x 128 x 32 images.
- MaxPooling Layer 2 x 2 with stride 2.
  - **Input:** 128 x 128 x 32 images.
  - **Output:** 64 x 64 x 32 images.
- Convolutional layer with 64 filters 2 x 2 and a padding that makes the output have the same dimensions as the input.
  - **Input:** 64 x 64 x 32 images.
  - **Output:** 64 x 64 x 64 images.
- MaxPooling Layer 2 x 2 with stride 2.
  - **Input:** 64 x 64 x 64 images.
  - **Output:** 32 x 32 x 64 images.
- Flatten layer that puts in 1D all the neurons in the previous layer.
  - **Input:** 32 x 32 x 64 images.
  - **Output:** 65536 neurons.
- Dense Layer 128. A fully connected layer.
  - **Input:** 65536 neurons.
  - **Output:** 128 neurons.
- Dense Layer 64. A fully connected layer.
  - **Input:** 128 neurons.
  - **Output:** 64 neurons.
- Dense Layer 64. A fully connected layer.
  - **Input:** 64 neurons.
  - **Output:** 64 neurons.
- Dense Layer 3 with gaussian distribution. A fully connected layer with three nodes that makes a gaussian distribution and returns its mean value and variance..
  - **Input:** 64 neurons.
  - **Output:** 3 distributions.
- Mixture distribution layer. Makes a gaussian distribution and returns its mean value and variance.

All Activation Functions are ReLUs except for the second to last dense layer, whose activation function is *tanh*, and the loss function is the Negative Log Likelihood Function.

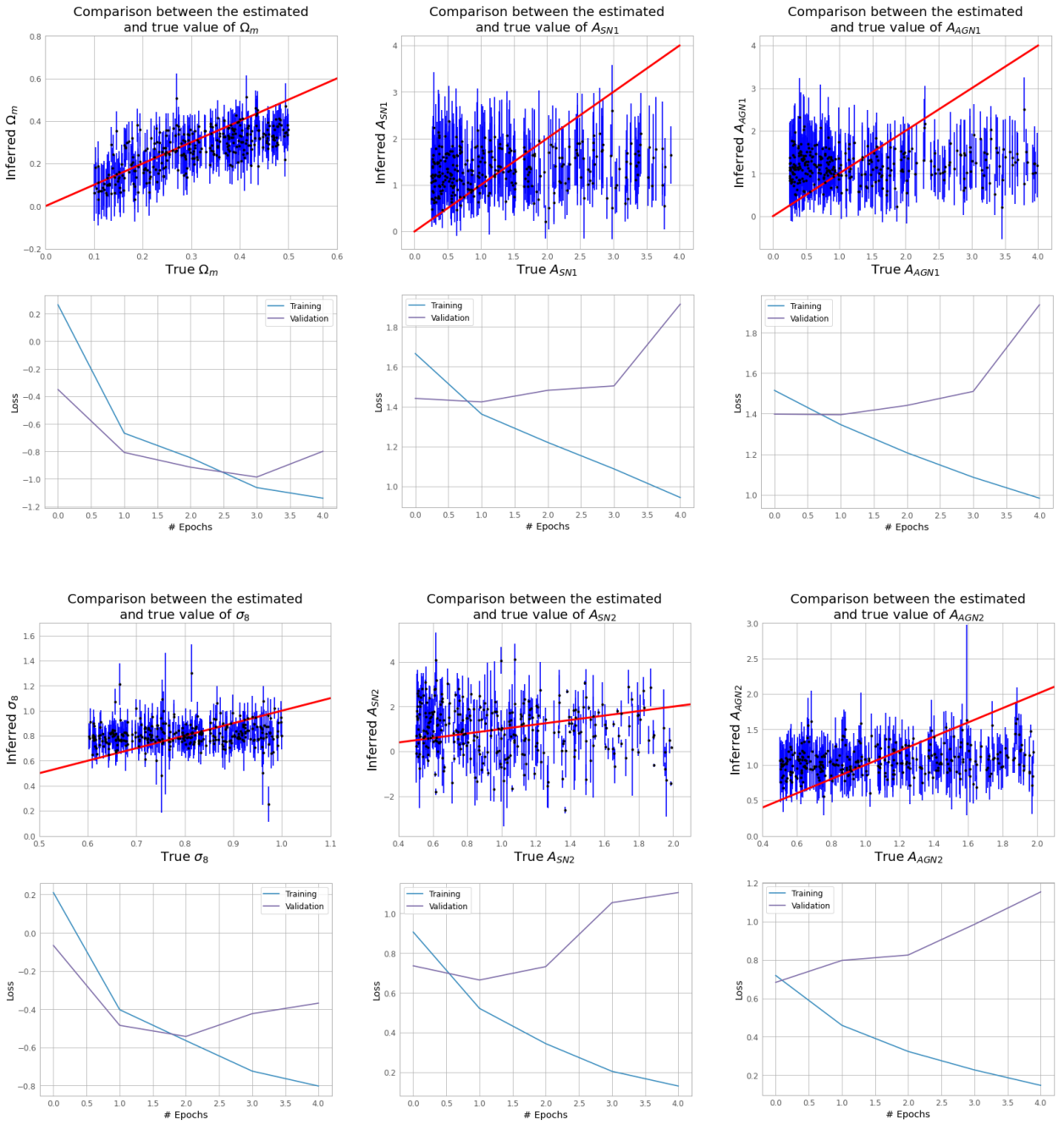

Figure 24: Obtained results for parameters Ω_m, σ_8, A_{SN1}, A_{SN2}, A_{AGN1} y A_{AGN2} with cosmic gas' temperature 2D maps from IllustrisTNG suite with their respective learning histories.

Preliminary results in Figure 24 where obtained using 3000 training maps, 300 validation maps and 300 testing maps. We can observe that the only parameter that the net infers fairly well is the cosmological parameter Ω_m, with error bars of about 0.3, but the other estimations are not accurate. This is a result of *over-fitting*, which happens when the model gets extremely effective at classifying or predicting data from the training set but not so good at identifying data from the untrained set [6]. Another approximation was tried in order to increase the amount of maps for the training process, but the convergence has not been achieved and is currently under investigation.

Comparing with the results of Villaescusa-Navarro et al. (2021) [35] in Figure 25, we can observe that the estimation of the astrophysical parameters that control AGN[10] feedback has quite bad results as well. This could happen because the net is not powerful enough to retrieve that information from these maps, or that the maps themselves do not have much information about these parameters [35].

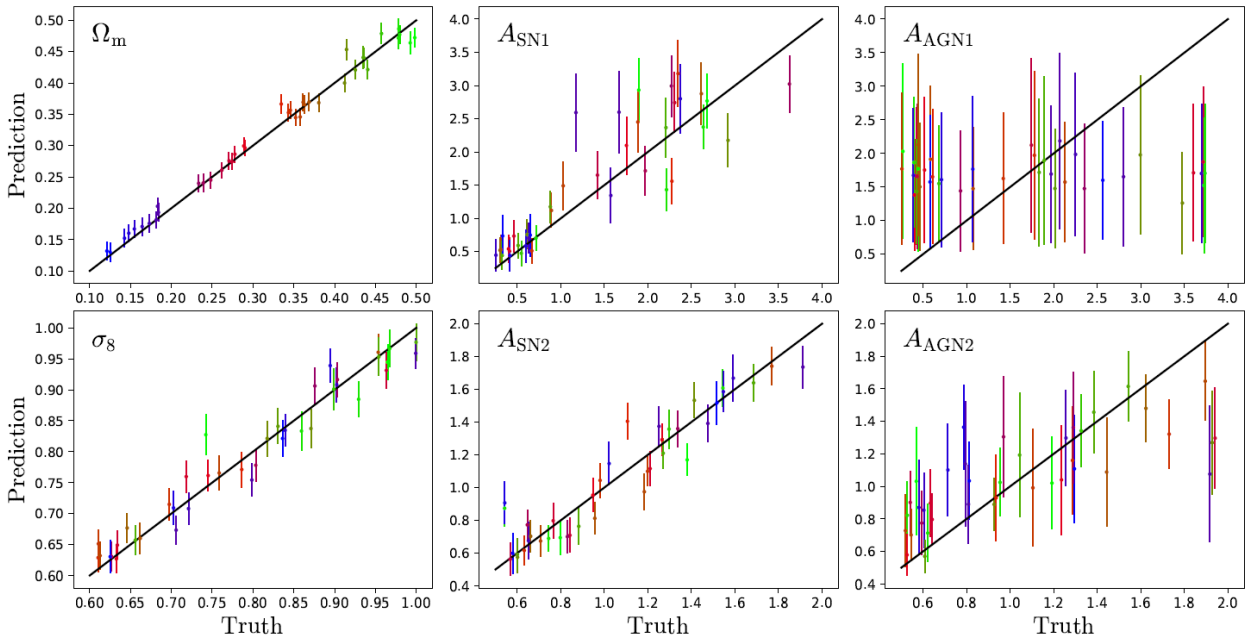

Figure 25: Obtained results by Villaescusa-Vavarro et al. (2021) [35] for parameters Ω_m, σ_8, A_{SN1}, A_{SN2}, A_{AGN1} y A_{AGN2} with cosmic gas' temperature 2D maps from IllustrisTNG .

In Figure 24 learning histories of the net for each parameters are also displayed. In the deduction of parameter Ω_m both the learning and loss function are being minimized for each epoch, what should happen. Validation data is used to select the optimal model for the inference of the output. A variety of networks can be built, and the one with the best accuracy in classifying the validation set is chosen [12]. For this reason, the loss function calculated with this data gives a hint whether the model will be accurate or not. Looking at

[10] Active Galactic Nucleus.

Figure 24 it is obvious that the validation loss function is not being minimized, this is, the trained model does not perform well with the validation data, and therefore will not give an accurate result.

Most probably using a larger training dataset would drastically improve the results obtained in Figure 24.

5 Conclusions

Resumen

Se concluye que las redes neuronales son uno de los algoritmos más apropiados para la resolución de problemas complejos por su rápido procesamiento de datos y la capacidad de aprendizaje a partir de una base de datos, entre otras razones. Aún así, este algoritmo tiene varias desventajas, como la indeterminación a la hora de elegir la estructura óptima del modelo (número de capas y neuronas), la necesidad de un amplio número de datos para su aprendizaje, y su gran dependencia en los aleatorios valores iniciales de los pesos y biases, ya que podrían conducir a los problemas del "gradiente evanescente" y "gradiente explosivo".

Se ha hecho obvia la necesidad de una base de datos grande al obtener los resultados ya que no han sido lo suficientemente precisos.

In summary, neural networks have been proved to be an state-of-the-art algorithm to solve more complex tasks. They have a number of distinguishing characteristics, including fast processing rates and the capacity to learn the answer to a problem from a set of data, making this algorithm part of the Supervised Learning algorithms. However, it has some drawbacks as well, such as the indeterminacy when choosing the optimal structure of the net (number of layers and neurons), the need of a huge amount of data that is not easily obtained, or the high dependency on the initial random values of the weights and biases that could originate the Vanishing or Exploding gradient problems.

The need for a big training set has become apparent in the results. As a result of using only 3000 maps of the CAMELS Multifield Dataset the inference of the parameters and the results have not been accurate enough.

6 References

[1] Albawi, S., Mohammed, T.A. & Al-Zawi, S. (2017). *Understanding of a convolutional neural network*. 2017 International Conference on Engineering and Technology (ICET). https://doi.org/10.1109/ICEngTechnol.2017.8308186

[2] Basilakos, S. (2002). Cosmological Parameters from the Clustering of AGN. *In: Modern Theoretical and Observational Cosmology. Astrophysics and Space Science Library. 276*. Springer. https://doi.org/10.1007/978-94-010-0622-4_14

[3] Bishop, C.M. (1994). *Mixture density networks*. Aston University. https://research.aston.ac.uk/en/publications/mixture-density-networks

[4] Bottou, L. (2012). Stochastic Gradient Descent Tricks. In: *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science. 7700*. Springer. https://doi.org/10.1007/978-3-642-35289-8_25

[5] Brownlee, J. (2018). What is the Difference Between a Batch and an Epoch in a Neural Network. *Machine Learning Mastery. 20.* https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[6] Caruana, R., Lawrence, S. & Giles L. (2000). Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In: *Advances in Neural Information Processing Systems. 13.* MIT Press. https://proceedings.neurips.cc/paper/2000/file/059fdcd96baeb75112f09fa1dcc740cc-Paper.pdf

[7] Castañeda, L. (2014). Cosmological parameter estimation from weak lensing. The case of Ω_m, $\sigma 8$. *Proceedings of the First Astrostatistics School: Bayesian Methods in Cosmology.* https://doi.org/10.48550/arXiv.1409.1274

[8] Cranmer, K., Brehmer, J. & Louppe,G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences. 117*(48). 30055-30062. https://doi.org/10.1073/pnas.1912789117

[9] CS231n: Deep Learning for Computer Vision(2022). *Convolutional Neural Networks for Visual Recognition*. Stanford University. https://cs231n.github.io/convolutional-networks/

[10] Doran, G. (2013). Characterizing Interference in Radio Astronomy Observations through Active and Unsupervised Learning. *JPL Technical Report Server*. http://hdl.handle.net/2014/44018

[11] Drew, P.J. & Monson, J.R. (2000). Artificial neural networks. *Surgery. 127*(1), 3-11. https://doi.org/10.1067/msy.2000.102173

[12] Foody, G. (2017). Impacts of Sample Design for Validation Data on the Accuracy of Feedforward Neural Network Classification. *Applied Sciences, 7*. 888. MDPI AG. http://dx.doi.org/10.3390/app7090888

[13] Galilei, G. & Van Helden, A. (1998). *Sidereus nuncius*. Venice, 1610. Octavo.

[14] Han, J., Kamber, M. & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Elsevier.

[15] Hertz, J., Krogh, A. & Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*. CRC Press. https://doi.org/10.1201/9780429499661

[16] Hezaveh, Y., Levasseur, L. & Marshall, P. (2017). Fast automated analysis of strong gravitational lenses with convolutional neural networks. *Nature. 548*. 555–557. https://doi.org/10.1038/nature23463

[17] Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*. https://doi.org/10.48550/arXiv.1502.03167

[18] Kavlakoglu, E. (2020). IBM. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?*. https://ibm.co/3O1fQal

[19] Krizhevsky, A., Sutskever, I. & Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[20] Krogh A. (2008). What are artificial neural networks?. *Nature Biotechnology. 26*(2), 195-197. https://doi.org/10.1038/nbt1386

[21] Mohammed, M., Badruddin Khan M. & Basier, E. (2016). Machine Learning: Algorithms and Applications. *CRC Press*. https://doi.org/10.1201/9781315371658

[22] National Aeronautics and Space Administration. *LAMBDA. Education & Graphics*. https://lambda.gsfc.nasa.gov/education/

[23] Nwankpa, C., Ijomah, W., Gachagan, A. & Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *arXiv*. https://doi.org/10.48550/arXiv.1811.03378

[24] Pack, L., Littman, M.L. & Moore, A.W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*. https://doi.org/10.1613/jair.301

[25] Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning. In: Proceedings of Machine Learning Research. 28*(3). 1310-1318. https://proceedings.mlr.press/v28/pascanu13.html

[26] Planck Colaboration: N., Aghanim, Y., Akrami, M., Ashdown, J., Aumont, C., Baccigalupi, M., Ballardini, A.J., Banday, R.B., Barreiro, N., Bartolo, S., Basak, R., Battye, K., Benabed, J.P., Bernard, M., Bersanelli, P., Bielewicz, J.J., Bock, J.R., Bond, J., Borrill, F.R., Bouchet, F., Boulanger, ... A., Lewis et al. (82 additional authors not shown) (2018). Planck 2018 results. VI. Cosmological parameters. *arXiv*. https://doi.org/10.48550/arXiv.1807.06209

[27] Richards, J.W., Homrighausen, D., Freeman, P.E., Schafer, C.M. & Poznanski, D. (2012). Semi-supervised learning for photometric supernova classification. *Monthly Notices of the Royal Astronomical Society. 419*(2). 1121–1135. https://doi.org/10.1111/j.1365-2966.2011.19768.x

[28] Robson, B.A. (2019). *Redefining Standard Model Cosmology*. BoD–Books on Demand.

[29] Rumelhart, D.E. & McClelland, J.L. (1987). Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press. 318-362.

[30] Sarker, I.H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*. https://doi.org/10.1007/s42979-021-00592-x

[31] Sharma, S., Sharma, S. & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Science, 4*(12), 310-316. https://doi.org/10.33564/IJEAST.2020.v04i12.054

[32] Spergel, D.N., Verde, L., Peiris, H.V., Komatsu, E., Nolta, M.R., Bennett, C.L., Halpern, M., Hinshaw, G., Jarosik, N., Kogut, A., Limon, M., Meyer, S.S., Page, L., Tucker, G.S., Weiland, J.L., Wollack, E. & Wright, E.L. (2003). First-Year Wilkinson Microwave Anisotropy Probe (WMAP)* Observations: Determination of Cosmological Parameters. *The Astrophysical Journal Supplement Series. 148*(1). https://doi.org/10.1086/377226

[33] Szandała, T. (2021). Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. In: Bio-inspired Neurocomputing. *Studies in Computational*

Intelligence, 903. Springer.
https://doi.org/10.1007/978-981-15-5495-7_11

[34] Tegmark, M., Strauss, M.A., Blanton, M.R., Abazajian, K., Dodelson, S., Sandvik, H., ... & York, D.G. (2004). Cosmological parameters from SDSS and WMAP. *Physical review D. 69*(10).
https://doi.org/10.1103/PhysRevD.69.103501

[35] Villaescusa-Navarro, F., Anglés-Alcázar, D., Genel, S., Spergel, D.N., Li, Y., Wandelt, B., Nicola, A., Thiele, L., Hassan, S., Zorrilla Matilla, J.M., Narayanan, D., Dave, R. & Vogelsberger, M. (2021). Multifield Cosmology with Artificial Intelligence. *arXiv*.
https://doi.org/10.48550/arXiv.2109.09747

[36] Villaescusa-Navarro, F., Genel, S., Anglés-Alcázar, D., Thiele, L., Dave, R., Narayanan, D., Nicola, A., Li, Y., Villanueva-Domingo, P., Wandelt, B., Spergel, D.N., Somerville, R.S., Zorrilla Matilla, J.M., Mohammad, F.G., Hassan, S., Shao, H., Wadekar, D., Eickenberg, M., Wong, K.W.K, ... Vogelsberger, M. (2022). The CAMELS Multifield Data Set: Learning the Universe's Fundamental Parameters with Artificial Intelligence. *The Astrophysical Journal Supplement Series. 259*(2).
https://doi.org/10.3847/1538-4365/ac5ab0

[37] Vossen, J., Feron, B. & Monti, A. (2018). Probabilistic Forecasting of Household Electrical Load Using Artificial Neural Networks. *2018 IEEE International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*.
https://doi.org/10.1109/PMAPS.2018.8440559

[38] Wagstaff, K.L., Daubar, I.J., Doran, G., Munje, M.J., Bickel, V.T., Gao, A., Pate, J.& Wexler, D. (2022). Using machine learning to reduce observational biases when detecting new impacts on Mars. *Icarus*.
https://doi.org/10.1016/j.icarus.2022.115146

[39] Wang, S. (2003). Artificial Neural Network. In: Interdisciplinary Computing in Java Programming. *The Springer International Series in Engineering and Computer Science*, 743. Springer.
https://doi.org/10.1007/978-1-4615-0377-4_5

[40] Wang, Q., Ma, Y., Zhao, K. & Tian, Y. (2022). A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science. 9*(2). 187–212.
https://doi.org/10.1007/s40745-020-00253-5

[41] White, S.D.M., Efstathiou, G. & Frenk, C.S. (1993). The amplitude of mass fluctuations in the Universe. *Monthly Notices of the Royal Astronomical Society. 262*(4). 1023–1028.
https://doi.org/10.1093/mnras/262.4.1023

[42] Wikibooks. Artificial Neural Networks/Error-Correction Learning.
https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Error-Correction_Learning

[43] Yatawatta, S. & Avruch, I.M. (2021). Deep reinforcement learning for smart calibration of radio telescopes. *Monthly Notices of the Royal Astronomical Society. 505*(2). 2141–2150.
https://doi.org/10.1093/mnras/stab1401

[44] Zhang, Q.J. & Gupta, K.C. (2000). Neural Network Structures. In: *Neural Networks for RF and Microwave Design*. Artech House.
https://www.ieee.cz/knihovna/Zhang/Zhang100-ch03.pdf

[45] Zhang, Y. (2010). *New Advances in Machine Learning*. InTech.

[46] Zhang, Z. & Sabuncu, M.R. (2018). Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. In: *Advances in neural information processing systems. 31*.
https://doi.org/10.48550/arXiv.1805.07836

[47] Zhu, D., Yao, H., Jiang, B. & Yu, P. (2018). Negative log likelihood ratio loss for deep neural network classification. In: *Proceedings of the Future Technologies Conference*. 276-282. Springer, Cham.
https://doi.org/10.48550/arXiv.1804.10690