
Politecnico di Torino

Politecnico di Torino

Master's degree in Aerospace Engineering
A.a. 2022/2023
December 2022

# Training of a Neural Network for fuel-optimised space trajectories

Supervisor:
prof. Lorenzo Casalino

Candidate:
Ginevra Berni

# Acknowledgements

*To my family*

*Who has supported me every step of the way.*

*And to my friends*

*Who have been with me till the end.*

Computational resources provided by hpc@polito, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it).

# Contents

**Introduction** **1**

**1 Artificial Intelligences** **3**

1.1 Historical evolution . . . . . 3

1.2 Artificial Intelligences in spacecraft guidance . . . . . 6

1.3 Purpose . . . . . 7

**2 Artificial Neural Network** **11**

2.1 ANNs origins . . . . . 11

2.2 Technical Aspects . . . . . 13

**3 Training method** **17**

3.1 Database . . . . . 17

3.2 Backpropagation algorithm . . . . . 20

3.3 Levenberg-Marquardt method . . . . . 22

**4 ANNs development** **25**

4.1 ANNs implementation . . . . . 25

4.2 ANNs Architectures . . . . . 27

**5 Results** **31**

5.1 Presentation of the outputs . . . . . 31

5.2 Analysis on neurons per layer . . . . . 35

5.3 Analysis on training epochs . . . . . 38
5.4 Analysis on hidden layers . . . . . 41
5.5 Performance of the neural networks . . . . . 43

**6 Conclusions** **47**

**Bibliography** **49**

**List of Figures** **54**

**List of Tables** **55**

# Introduction

> "Aeronautical engineering tests do not define the goal of their field as making machines fly so exactly like pigeons that they can fool other pigeons."
> *Stuart Russel* [1]

As expressed by Stuart Russell in the opening quote, one of the main problems concerning Artificial Intelligence (AI) which has been discussed since its creation, lies in its definition. This problem can mostly be traced to the definition of intelligence itself. The Oxford Dictionary defines Artificial Intelligence as the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

Alan Turing, who can be considered the founder of Artificial Intelligence, describes in one of his articles the Turing test, which was created in order to understand whether a machine is "intelligent" or not. This test consists in a human interrogator interacting vocally with a machine by asking some questions. If this interrogator cannot tell whether he is talking to a human or to a machine, the test is successful, and the machine can be considered intelligent. Despite the importance of this test, it has not been considered a fundamental step for the creation of AIs themselves, as scientists and computer engineers focused more on creating an intelligence rather than imitating a human being.

AIs are often used nowadays, and they spread over a wide range of possible applications. Search engines use AIs to rank contents on the internet for web searches, identifying the most relevant pages to propose them as first results. Social media often exploit them to filter their contents with image recognition, managing to distinguish object, animals and people in pictures, but also by providing the user with posts they are interested in based on past google searches and recently bought articles. On many

platforms, subtitles can be automatically generated thanks to Artificial Intelligences, which are able to transcribe speech into text through language processing[2].

Curiously, now that AIs have been created and used in applications such as Google, recommendation systems and self-driving cars, an interesting effect was observed, often called the AI effect. It looks like the daily use of Artificial Intelligences creates such a normalcy about them that they are not considered intelligent anymore[3].

This work has been structured into five chapters. In chapter 1 applications of Artificial Intelligences in the field of space engineering are presented, together with the aim of this project. In chapter 2 a brief explanation of Artificial Neural Networks and their functioning can be found. The method used to train the Neural Networks is introduced in chapter 3, whereas the chosen architectures are presented in chapter 4. Finally, in chapter 5 the results obtained by the Neural Networks are shown and compared, to understand which architecture allows to obtain the most accurate output.

# 1 | Artificial Intelligences

## 1.1 | Historical evolution

The term Artificial Intelligence, or AI, was coined by computer and cognitive scientist John McCarthy in 1956 with the Dartmouth Summer Research Project on Artificial Intelligence, to refer to the academic discipline that had previously been called computer simulation.

The actual start of Artificial Intelligence studies is difficult to date, and tracing the history of Artificial Intelligence can be quite complicated. Often, its start is considered in the 20th century with the creation of the first computing machines, and especially with the work done by the mathematician Alan Turing, even though Charles Babbage had already tried building a machine, called difference engine, that would have been able to tabulate polynomial functions. He ultimately failed for lack of funding. Other times instead, AI studies are considered much older: philosophers such as Aristotle and Plato, by trying to understand the functioning of the human brain, were already laying the foundation of this work.

What can be said for certain is that imitations of human behaviour and intelligence have been fascinating people for a long time. Already in Homer's work, the description of how Hephaestus, the god of fire and blacksmiths, created attendants looking just like people and even learning by imitation is perfect evidence of this. Later on, works exploring an artificially created life were written, although often presenting an abundance of horror elements, such as Hoffmann's The Sandman and Mary Shelley's Frankenstein[4, 5]. Even though Frankenstein focuses on the moral responsibilities on the part of the scientist of creating such an intelligent creature, there is no denying its belonging to the horror genre. Isaac Asimov instead, writing his stories during the 20th centuries, in a period when AIs were already studied, provides a computer-

oriented version of AIs, more focused on them being machines and less focused on the life creation aspect that had much fascinated authors in the past. Asimov describes his robots as machines that have, deep within their code, three fundamental laws of robotics preventing them from harming human beings. In his short story Runaround, he explores these apparently simple rules and, while showing useful and positive aspects of AIs, he warns the reader of the dangers deriving from their use[6].

As previously mentioned, the proper start of Artificial Intelligence is to be considered during World War II, when Alan Turing created a machine, called The Bombe (Figure 1.1), which was able to decipher encrypted messages sent by the German army. A first period known as AI spring followed his work. Russell's propositional logic, based on a true/false dualism, could be easily implemented with binary 0 and 1. Together with Sherrington's theory on synapses, these concepts were used by neurologist Warren McCulloch and mathematician Walter Pitts to prove that Turing computation could be applied to both human and machine intelligence.

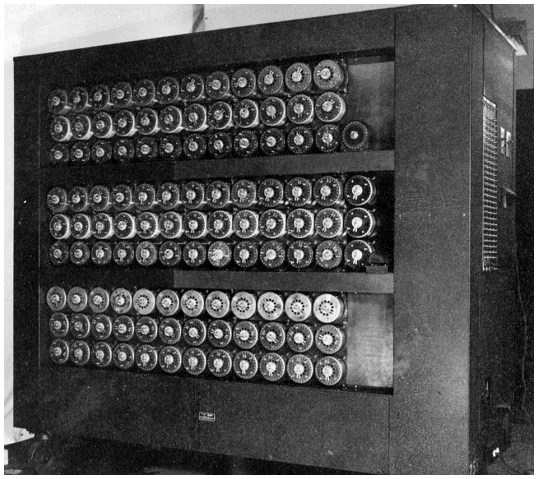

Figure 1.1: *The Bombe, machine developed by Alan Turing during World War II.*

Notorious milestones in the AI research fields are the Logic Theory Machine, which managed to prove eighteen of Russell's logical theorems, and the General Problem Solver, which achieved a vaster application field. Russell's logical theorems, proved in *Principia Mathematica*, are a set of theorems on the foundations of mathematics, for instance using and/or logical operators as an algorithmic basis[7].

The two machines previously mentioned are both Good Old-Fashioned Artificial Intelligences (GOFAIs), meaning that they use tools such as logic programming, semantic nets and frames. In the same years connectionism was encouraged as well,

using networks of logical neurons. Parallel-processing systems were only achieved later, in 1958, when Frank Rosenblatt partially implemented his Perceptron Machine, able to classify images. The critiques on this work, which admittedly had a few limitations, can be considered the start of the period called AI winter, when fundings for research on this field were not available anymore.

The continuous work on GOFAIs was meanwhile successful, as it brought to the creation of the famous Deep Blue, which managed to defeat the current world champion Garry Kasparov at chess (Figure 1.2). The AI winter lasted until 1986, when neural networks came back on the scene thanks to Parallel Distributed Processing (PDP), and symbolic AI was challenged.


Figure 1.2: *Garry Kasparov playing chess against Deep Blue.*

The name GOFAI was first used as a derogatory term to show limitations on what this type of AI could do, claiming that the complexity of the human mind could not be captured by such simple logic rules and logical agents. This problem came to be called the qualification problem, and was mostly sustained by philosophers like John Haugeland and Hubert Dreyfus[1].

Besides GOFAIs, also called classical AIs, other types of Artificial Intelligences exist. The main ones are Artificial Neural Networks (ANNs), evolutionary programming, cellular automata (CA) and dynamical systems. Artificial Neural Networks, which will be analysed in depth in chapter 2, are composed of interconnected units, each one capable of computing only one thing. These AIs are practical to implement pattern recognition and learning. Evolutionary programming uses algorithms to reach optimisation trying to imitate biological evolution, whereas a cellular automaton is a mathematical model used to describe discrete systems. The latter uses a grid of cells

whose state changes according to the state of the neighbouring cells and, same as dynamical systems, it is often used to describe development of living organisms.

Nowadays, contrary to what happened in the AI winter, thanks to the enhancement of computers capabilities, research on AIs focuses mainly on neural networks and connectionism.

## 1.2 | Artificial Intelligences in spacecraft guidance

Quite often it has been remarked that AIs can work very well on a certain topic, but if transferred to another field they can have a rather poor performance. In recent years, with the development and improvement of ANNs, attempts have been made to use AIs in the space related sector, and in particular a focus has been placed on the guidance and control field. Evolutionary algorithms, using heuristic rules, have been often exploited to search for optimal solutions. This technique has been especially used to solve problems like interplanetary combinatorial optimisation. These are problems with a finite number of possibilities, where different options concerning the visiting order of the celestial bodies are present. The astronomical objects order needs to be studied while still looking for the optimal trajectory; this way non continuous variables can be analysed as well[8].

Nevertheless, to improve results for this last case, search trees have appeared to be more successful. Indeed, they are characterised by decision nodes, which allow to explore only the most promising branches.

The use of Machine Learning (ML) is instead more connected to the topic dealt with in this work, even though, for what concerns application fields, it is not as developed as the ones mentioned beforehand. One of the main problems for ML use in the aerospace sector is the necessity of a huge amount of training data; despite that, recently ANNs rose in popularity, especially when using a large number of hidden layers and implementing supervised learning. The characteristic of ML that permits to adapt to unknown situations, as shown by Reinforcement Learning, means that these AIs can be used for autonomous navigation.

ANNs have been sometimes developed in the aerospace sector to solve problems such as pose estimation, which focuses on finding the position and orientation of a spacecraft by acquiring and analysing images from the surrounding environment. More

often though, they have been used to find global optimal trajectories for spacecrafts; this presents the considerable advantage to not require a deep knowledge of the topic and the method by its user, which is the opposite of what happens with traditional optimisation methods[9].

To present some examples, deep neural networks have already been studied and implemented in previous works to compute the landing point of mass varying spacecrafts and rockets, assuming perfect knowledge of the spacecraft state[10], and to smooth the functioning of traditional techniques by executing control during their procedure, reacting this way to unpredicted events[11]. As explained by Gaudet and Furfaro[12], a spacecraft can use images acquired during the descent itself to compute its own landing spot with a remarkable precision (in the order of meters). For what concerns interplanetary trajectories instead, ANNs can guide a low-thrust spacecraft through an optimal trajectory, becoming independent from operators instructions during this process[13].

## 1.3 | Purpose

In the aerospace field, where the objective is to allow an aircraft or a spacecraft to fly, weight has always been one of the main parameters to keep under control. Its reduction often has a big impact on a design, also due to the snowball effect, a process loop based on the fact that carrying less weight implies needing less fuel for the mission, and therefore having less weight. A smaller tank for the fuel also means that more space is available for other systems, and that overall the cost of the mission could decrease remarkably. In this perspective, the project here presented revolves around computing on MATLAB the optimal fuel mass needed for a spacecraft, through the creation and the training of an Artificial Neural Network.

The spacecraft rides along fuel-optimised trajectories through space to go from one planet to another, with a guidance and control system. In case of big deviations from the planned orbit, a new optimised trajectory needs to be computed. When traditional methods to implement guidance and control are used to find the new path, a lot of time is required both for the computations themselves and for data transmission. Indeed, the problem to be solved is usually quite complicated and either cannot be resolved in real-time or it requires the intervention of experts to redefine its parameters[13]. For this reason, the computation cannot happen onboard but it takes place in ground stations

equipped to do mission control. Therefore, the spacecraft needs to send data to the Earth concerning its position and state, so that the ground station computes an answer and then sends the results back to the spacecraft. As expected, for certain missions taking place far away from Earth, this process can require a lot of time and be overall inefficient.

An alternative method has therefore been proposed, by using an Artificial Intelligence that has been trained on the ground for a specific trajectory, and then brought onboard. This AI can compute, for any state of the spacecraft, the minimum amount of fuel needed to reach the destination, which corresponds to the one used for an optimal trajectory. Its computation times are generally much shorter than the ones required by traditional methods. This way, the AI has the advantage of not only being on board, cutting all data transmission waiting times, but also to be used in real-time.

With a more in depth work, this could be extended to computing all commands necessary to move the spacecraft along the optimal trajectory, equipping it with a guidance and control system and granting it the ability to make decisions and to become completely independent from communication links with the Earth. In recent years a new method aiming to both steer the spacecraft and optimising the trajectory has been often considered. The lack of (or even just a less used) communication system would imply many advantages, such as weight reduction, smaller volume, less power needed onboard the spacecraft and fuel saving. Moreover, real-time answers to trajectory deviations would increase chances of success for the mission, shortening response times in case a problem arises suddenly. For instance the spacecraft would not have time to deviate too much from its path if the trajectory is adjusted right away.

Consequently, the aim of the work is to implement this ANN on MATLAB applying it to our case study, which is a low-thrust trajectory from the Earth to Venus orbit. A database exists for this case, containing low-thrust fuel-optimised trajectories for the design mission and for a lot of possible states where all parameters are slightly deviated compared to the foreseen state of the spacecraft. Perturbations on these parameters could be easily explained and hardly avoided during the mission, for instance due to a thruster misalignment, environmental disturbances, model uncertainties or actuators failures, resulting in a different position at a certain time, or in a different mass of the spacecraft compared to the expected one[9]. An ANN can then be trained to compute the minimum fuel required for these deviated states of the spacecraft.

In this project, different ANNs architectures are analysed and compared, while always using the same training algorithm and training dataset. This allows to explore the influence of the ANN architecture on its outputs and especially on the accuracy obtained. The aim is to identify the best architecture between the ones implemented, and to try and find a pattern between ANNs characteristics and their behaviours in order to be able to improve the results obtained in future analyses. An accurate result would imply the possibility of using such a system to execute guidance and control on a spacecraft.

# 2 | Artificial Neural Network

## 2.1 | ANNs origins

Throughout history, many philosophers and scientists have wondered over the functioning of the human brain. From Aristotle to Kant, time and effort have been dedicated to try and understand how the human brain works. Mathematician George Boole, mostly known for the development of Boolean logic, worked on the idea that human thought processes could be reproduced through a series of mathematical equations even before the creation of AIs[3].

Early on in the research for Artificial Intelligence scientists wondered if they could manage to create intelligence by imitating the structure itself of the human brain. Contrary to the first studies on AIs, this would involve components able to operate in parallel. The fundamental cell of the central nervous system is the neuron, which is composed of dendrites, a cell body called soma, and an axon (Figure 2.1). Dendrites continuously acquire signals which are then processed in the soma by producing an activation potential. If this potential is higher than a certain threshold, an electric impulse is triggered and transmitted to the axon. The connections that transmit the impulse to another neuron are called synapses.

Already in 1943, neurophysiologist Warren McCulloch and logician Walter Pitts studied the behaviour of biological neurons and developed an artificial model that replicated it. In this model the neuron was represented as a proposition, thanks to the all-or-none law of nervous activity, and the main aspects were parallelism and high connectivity. It remains nowadays one of the most used models in ANNs architectures[14]. A few years later, the first training method was proposed, based on Hebb's rule. Donald Hebb was a psychologist which expresses the idea that repeated stimulation of a neuron increases its efficiency, thanks to the growth of the axon[15]. New

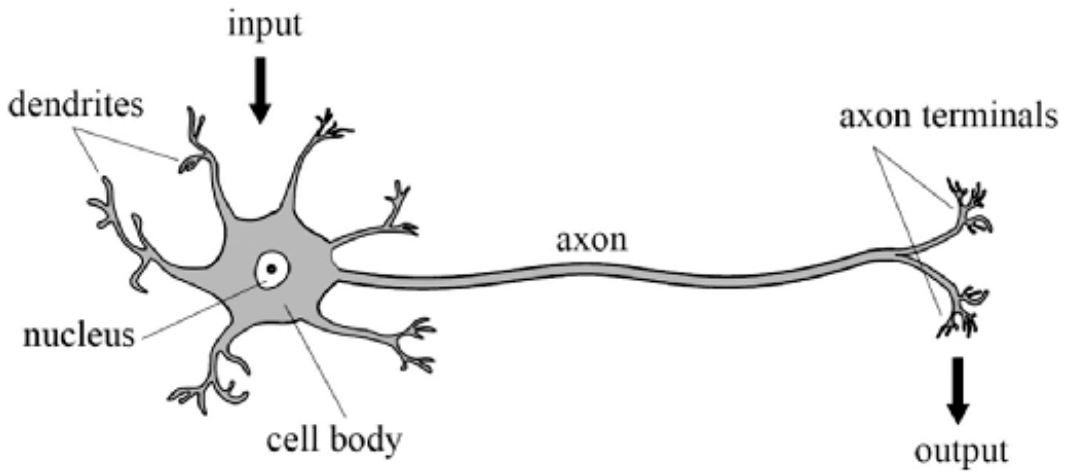


Figure 2.1: *Schematic representation of a biological neuron.*

cognitive abilities are then acquired by strengthening some connections and weakening others. Research and development of ANNs based on neurological studies continue until the end of the 1950s with success, in a period that has been afterwards called the AI spring. At the end of this decade though, limitations on neural networks were found, and research on this field had a setback. Only after a serious development in computer processing skills and on their memory capabilities, together with new findings on the biological nervous systems, Artificial Neural Networks could be studied again. In particular, more than one layer of neurons could finally be trained, determining a definitive improvement over previous works.

Nowadays, ANNs find numerous applications. They have been successfully used in the chemistry area with the aim of correlating spectra such as infrared and UV to the chemical structure of a compound, and in the biology area for the prediction of the secondary structure of proteins[16]. In the financial and economics field ANNs are relevant for stocks forecasting, thanks to their ability to handle non-linearity, and they also find applications in the pharmaceutical field, in acoustics problems, in the food industry and in the automotive and aerospace industry. In this last case, ANNs non linearity has been exploited by Cho et al. to describe aircraft dynamics, whose behaviour vary over the operating regime[17].

Overall, it is the aforementioned feature of non linearity that has made ANNs as popular and useful as they are today, allowing the description of functions and systems that were very complex to represent until now.

## 2.2 | Technical Aspects

Artificial Neural Networks generally present a structure made up of an input layer, which collects data used as inputs, a variable number of hidden layers and an output layer. These layers are composed of elements called artificial neurons, and are interlinked with artificial synapses implemented by matrices of weights.

In Figure 2.2, an example of a trained Artificial Neural Network is shown. The inputs, here represented as grey arrows on the left, are numbers describing the state of the problem that needs to be solved. The input layer, composed of grey square-shaped elements in the same number as the inputs, receives them and feeds them into the neural network. These numbers are then operated on moving from left to right according to the diagram shown.

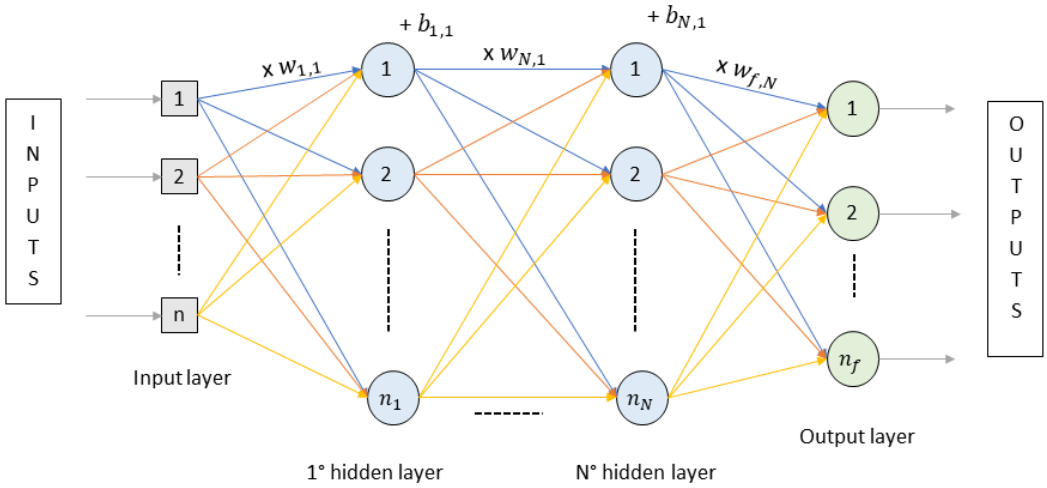


Figure 2.2: *Scheme of an Artificial Neural Network*

Each coloured arrow represents a synaptic weight (the value is here called $w$) by which the input is multiplied. These signals, still represented as numbers, are received by the round-shaped elements, called artificial neurons. All inputs arriving into the same neuron are added together. A bias value ($b$) analogous to the activation threshold in biological neurons is added to the result; then an activation function is used to limit the output from each layer to a reasonable range of values. This prevents the result of these computations from growing exponentially along the neural network, usually limiting the neurons output between -1 and 1. The output is then multiplied by the

next weight and transmitted this way to neurons in the next hidden layer. This process continues for all $N$ layers of the Artificial Neural Network, until it reaches the output layer. The output layer, here shown as a column of green circles on the right, needs to have the same dimension as the desired output, and returns the solution to the problem fed into the ANN.

Activation functions can be partially differentiable, such as when a step function (hard limiter), a bipolar step function (symmetric hard limiter) or a symmetric ramp function are used, or fully differentiable. The most used fully differentiable activation functions in ANNs are logistic functions, hyperbolic tangents, Gaussian functions and linear functions.

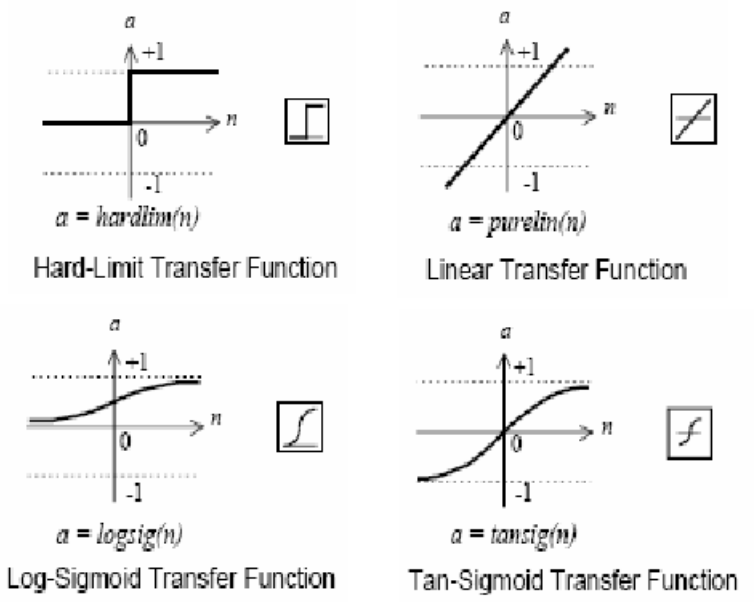


Figure 2.3: *Some examples of activation functions.*

The main features of ANNs that make them useful for our work are three:

1. their ability to adapt from experience, by changing weight and bias values with the examination of successive examples, and to extract existing relationships between several variables through the use of a learning method,
2. their ability to generalise the acquired knowledge, allowing to estimate solutions so far unknown,
3. their fault tolerance, thanks to the high number of connections, allowing them to give a valid solution even if part of the internal structure of the network is corrupted.

The most primitive neural networks only have an input layer and an output layer, and are called single-layer feed-forward neural networks. The ones used in this work, called multiple-layer feed-forward neural networks, are more complex than these, since they are made of one or more hidden layer in between the input and the output layers.

Different learning strategies have been proposed over time. Since ANNs have the ability to learn from examples, the training process usually consists of feeding the neural network a certain set of data, containing all available samples of the system behaviour. These samples are divided in three different subsets: the training subset, the test subset and the validation subset. During the training process, each use of the entire training subset is called training epoch, a different concept compared to iterations, which instead also include the use of the validation subset.

In this work, supervised learning is used. This technique consists in the neural network being fed all the inputs from the training data, as well as the corresponding outputs. At each epoch, the neural network outputs are computed and compared to the real outputs, and synaptic weights and thresholds are modified accordingly. Alternatively, unsupervised learning does not require any knowledge of the corresponding output, leaving to the network to distinguish different clusters and then to adjust weight and bias values.

# 3 | Training method

## 3.1 | Database

The Artificial Neural Networks created need to be trained. As already mentioned in chapter 2, supervised learning is chosen for our work. In order to do this, a huge number of samples for our fuel-optimized trajectories from the Earth to Venus orbit is needed. The database used contains 429316 low-thrust optimal trajectories subjected only to Sun gravity, for a spacecraft leaving the Earth on the 7th May 2005 and using an ion thruster characterised by $I_{sp} = 3800s$ and by a maximum thrust $c1 = 0.33N$. The mass of the spacecraft is $m = 1500kg$. These trajectories are computed according to the International Celestial Reference Frame, and present 100 samples equally covering the interplanetary space along each trajectory. The database is composed of 20 columns, and most values are nondimensional, or chosen so that the gravitational parameter has a value of 1. The variables contained in this database are time, six equinoctial parameters, the mass, seven costates of the Optimal Control Problem Hamiltonian, the thrust magnitude, the thrust directions, the unique trajectory id, the sample id and the value function[18].

The choice to use nondimensional parameters implies that time, which would be measured in years, is normalised based on the Earth angular velocity, computed as $\omega_T = \frac{2\pi}{1yr}$. Mass is nondimensionalised based on the aforementioned mass of the spacecraft. It is useful to note that the mass of the spacecraft is not necessarily the mass of the first sample of each trajectory, as each trajectory is created by perturbing the given values. For instance, the initial mass for the nominal trajectory is 1502,8 kg instead of the already mentioned 1500 kg. The Modified Equinoctial Elements (MEEs) instead are normalised through the use of the astronomical unit.

To generate this database, the classical approach would have been to solve a large number of optimal control problems. Instead of doing that, which would have required a huge amount of time and computational power, Izzo et al. used a method called backward generation of optimal examples. This method solves, only once, the two-point boundary value problem resulting from the application of the Pontryagin principle, and then generates more optimal trajectories at the cost of simpler numerical integrations[13].

To give an example of the trajectories and the data that this work focuses on, the 100 samples composing the nominal trajectory have been represented in two dimensions in Figure 3.1.

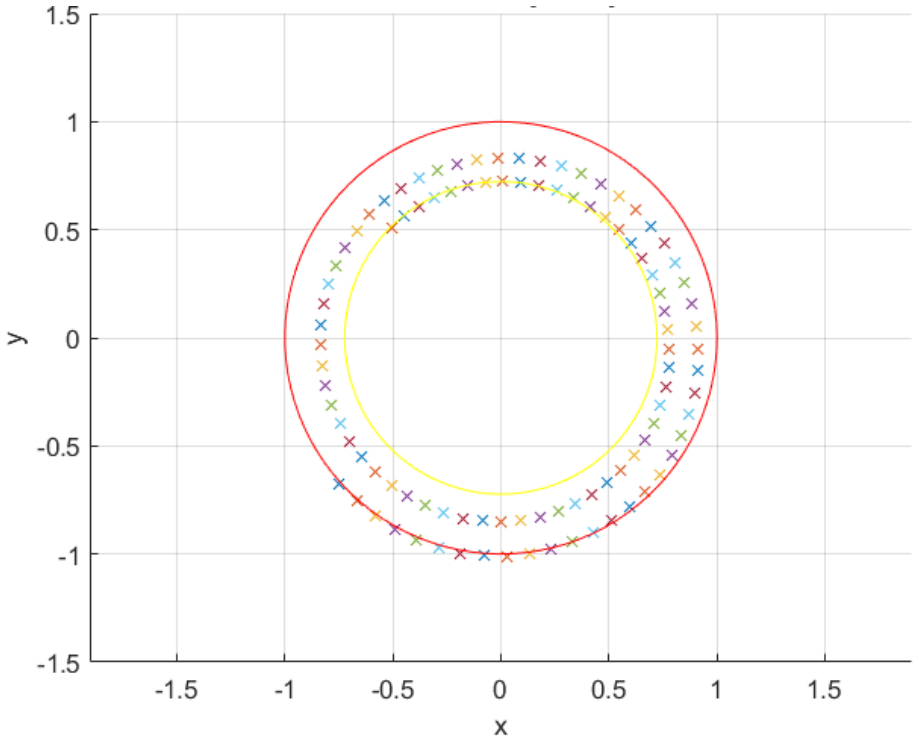


Figure 3.1: *Nominal trajectory. In red, the Earth orbit can be observed, whereas the yellow line represents Venus orbit.*

Each cross represents the position of the spacecraft at a certain time, and is obtained from the six Modified Equinoctial Elements, whose connection to the more common orbital elements is shown in Equation 3.1. The use of MEE is favoured due to the fact that the classical parameters have the right ascension of the ascending node becoming indeterminate as the inclination tends to zero, and the argument of perigee becoming indeterminate as the eccentricity tends to zero[19].

$$
\begin{aligned}
p &= a(1-e^2) \\
f &= e\cos(\omega+\Omega) \\
g &= e\sin(\omega+\Omega) \\
h &= \tan(i/2)\cos(\Omega) \\
k &= \tan(i/2)\sin(\Omega) \\
L &= \Omega+\omega+\nu
\end{aligned}
\tag{3.1}
$$

The value function for the nominal trajectory, whose meaning is explained in chapter 4, is also shown in Figure 3.2. For the moment, it is only necessary to know that it is connected to the fuel mass needed, and as expected, this value decreases when closer to Venus orbit. When a plateau in the function is observed, those points correspond to moments where the throttle is zero, and the engine is turned off. A ballistic trajectory is followed in those instants, therefore no fuel is used.

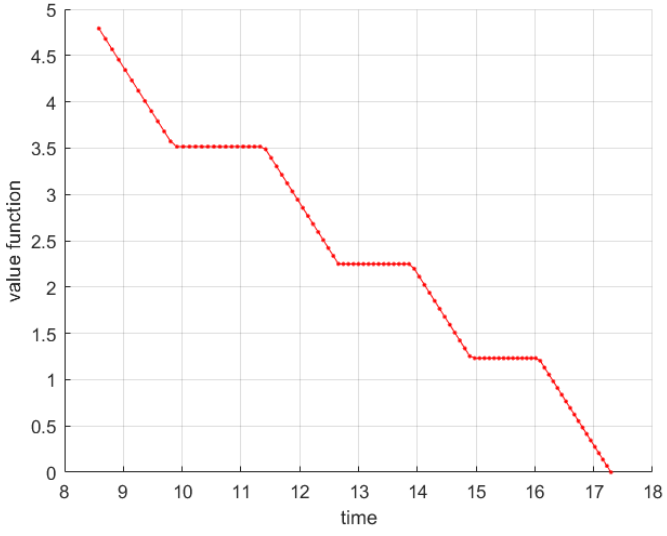


Figure 3.2: *Nominal value function.*

## 3.2 | Backpropagation algorithm

In order to understand the method used to train the neural networks, a study of the backpropagation method is first needed. The topology used, also shown schematically for a two hidden layer neural network in Figure 3.3, is the following:

- $W_{ji}^{(L)}$ are the weight matrices representing the synaptic weight that connects the $j^{th}$ neuron of layer $(L)$ to the $i^{th}$ neuron of layer $(L-1)$.
- $I_j^{(L)}$ are vectors reporting the weighted inputs for each neuron belonging to layer $(L)$

$$I_j^{(L)} = \sum_{i=0}^{n} W_{ji}^{(L)} Y_i^{(L-1)} \tag{3.2}$$

  where $n$ is the number of element for the layer L.
- $Y_j^{(L)}$ are vectors containing the outputs of the $j^{th}$ neuron of layer $(L)$, after applying the activation function $g$

$$Y_j^{(L)} = g\left(I_j^{(L)}\right) \tag{3.3}$$

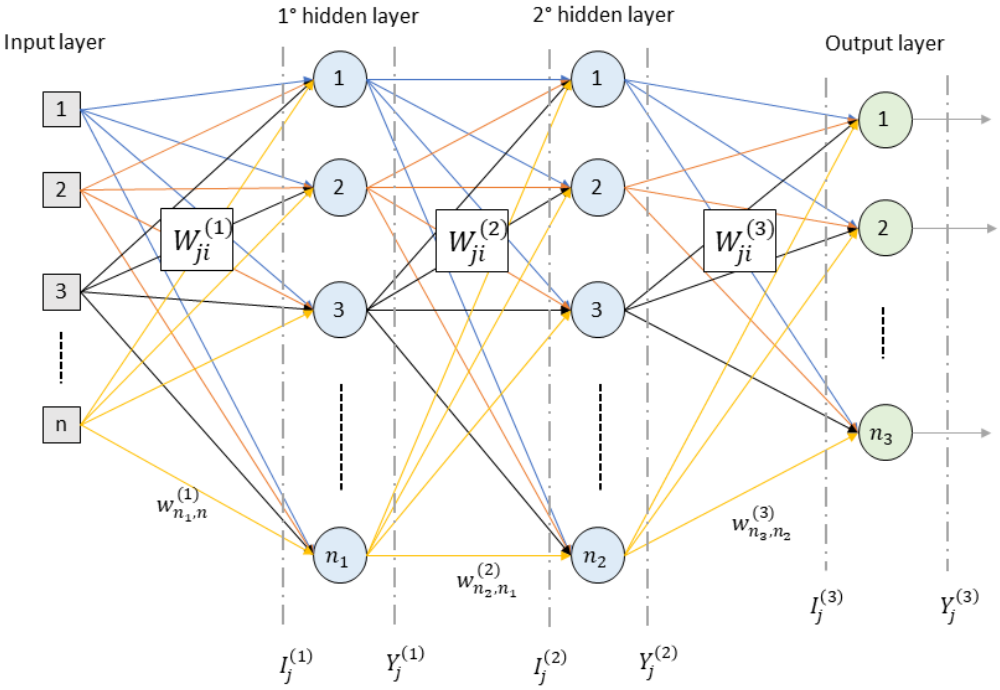


Figure 3.3: *Diagram of a two hidden layer neural network with notation.*

In the backpropagation algorithm, the squared error function (Equation 3.4) is chosen as the function to measure the distance between the results produced by the network and the desired values. Let us consider a network with N hidden layers; (N+1) denotes then the output layer, whereas $n_3$ is used to express the number of neurons in the output layer. $d_j(k)$ are the desired outputs for our neural network.

$$E(k) = \frac{1}{2} \sum_{j=1}^{n_3} \left[ d_j(k) - Y_j^{(N+1)}(k) \right]^2 \tag{3.4}$$

This function is then used to compute the performance of the neural network produced. Accordingly, using a training set composed of p samples, the performance is:

$$E_M = \frac{1}{p} \sum_{k=1}^{p} E(k) \tag{3.5}$$

The first step to train the created ANN with the backpropagation algorithm is to modify the weights of the output layer in order to minimise this error between produced and desired outputs. This is done by moving in the opposite direction of the gradient, which therefore needs to be computed. Using the definition of gradient:

$$\nabla E^{(N+1)} = \frac{\partial E}{\partial Y_j^{(N+1)}} \frac{\partial Y_j^{(N+1)}}{\partial I_j^{(N+1)}} \frac{\partial I_j^{(N+1)}}{\partial W_{ji}^{(N+1)}} \tag{3.6}$$

From Equation 3.2, Equation 3.3 and Equation 3.4, it is possible to express it as

$$\nabla E^{(N+1)} = -\left( d_j - Y_j^{(N+1)} \right) g' \left( I_j^{(N+1)} \right) Y_i^{(N)} \tag{3.7}$$

The adjustment of the weight matrix $W_{ji}^{(N+1)}$ to minimize the error in the opposite direction of the gradient is done while choosing a learning rate $\eta$ for the backpropagation algorithm. $\delta$ substitute the first two members in the previous equation.

$$\Delta W_{ji}^{(N+1)} = \eta \, \delta_j^{(N+1)} \, Y_i^{(N)} \tag{3.8}$$

The iterative process is easily obtained:

$$W_{ji}^{(N+1)}(t+1) = W_{ji}^{(N+1)}(t) + \eta \, \delta_j^{(N+1)} \, Y_i^{(N)} \tag{3.9}$$

After having adjusted the weights in the output layer, the weights of the previous layer can be modified. The output layer weights will be used to compute the output error, now also called backpropagated error. This is exactly why it is called the back-propagation algorithm: first the output layer is adjusted, then the second to last layer is adjusted, and so on and so forth, moving backwards along the network.

## 3.3 | Levenberg-Marquardt method

The backpropagation algorithm is a quite computationally demanding algorithm with a slow convergence. In order to solve these problems, some variations have been created, such as a method that uses the momentum parameter, the resilient-propagation method and the Levenberg-Marquardt method. The latter is the one that has been used to train the ANNs created, so it will be the only one explained in detail.

The Levenberg-Marquardt algorithm is a second-order gradient method based on the least squared method for nonlinear models, and it can remarkably improve the efficiency of the training process[20].

On the training set composed of p samples, the error can be expressed through Equation 3.4 and Equation 3.5, thus becoming

$$V = \frac{1}{2p} \sum_{k=1}^{p} \sum_{j=1}^{n_3} \left( d_j(k) - Y_j^{N+1}(k) \right)^2 \quad => \quad V = \frac{1}{2p} \sum_{k=1}^{p} E^T(k)\, E(k) \tag{3.10}$$

This method will not use the gradient of the squared error function to reduce the error, but an approximation of the Newton method, so the minimization of a function is given through an iterative process using the Hessian and the Jacobian matrices.

$$\Delta z = - \left( \nabla^2 V(z) \right)^{-1} \nabla V(z) \tag{3.11}$$

If $V(z)$ is a function executing $m$ quadratic functions, as in Equation 3.10, then $V(z) = \sum_{i=1}^{m} e_i^2(z)$. From this, it is possible to compute the Jacobian and the Hessian matrixes, which can be substituted in Equation 3.11. Therefore, the iterative expression of the Levenberg-Marquardt method is obtained:

$$\Delta z = \left( J^T(z)\, J(z) + \mu I \right)^{-1} J^T(z)\, e(z) \tag{3.12}$$

Computing the equation to change the weight matrices is then easy.

$$\Delta W = \left[J^T(W)\, J(W) + \mu I\right]^{-1} J^T(W)\, E \tag{3.13}$$

where $E$ is the error vector on $p$ training samples.

By using the Levenberg-Marquardt method, Hagan and Menhaj proved that the training process is in the order of magnitude of 10-100 times faster than the conjugate gradient backpropagation algorithm[21], although if $J(z)$ is ill-conditioned problems due to the convergence can arise.

# 4 | ANNs development

## 4.1 | ANNs implementation

The neural networks presented in chapter 2 are easily created on MATLAB as feedforward neural network with N+1 layers. The database from chapter 3 is used in order to train these networks through the Levenberg-Marquardt method, with supervised learning.

The inputs fed into the neural networks are time, mass and the six Modified Equinoctial Elements. Together, time and the six MEEs fix position and velocity of the considered spacecraft[19]. The numbers are given to the network without any computation beforehand, so they are presented as nondimensional values. The advantage of using nondimensional inputs is that the normalisation taking place after the first hidden layer this way is usually smoother.

The output of the neural network in this work is also called value function, because it expresses how good it is for the spacecraft to be in a given state. It is a value contained in the database and it is computed as the minimum of the functional J, expressed in Equation 4.1.

$$J(u(t),t_f) = \int_0^{t_f} \{u - \varepsilon \log[u(1-u)]\}\, dt \tag{4.1}$$

$u(t)$ represents the throttle magnitude, with values ranging from 0 to 1 due to the already mentioned normalisation. $t_f$ is the total transfer time of the spacecraft and $\varepsilon$ is the continuation parameter used to parameterise J and to make sure that the throttle never goes beyond a value of 1.

As it is possible to see, the functional J expresses a measure of how long the engine is turned on during the whole mission. It is therefore a quantity in seconds which

allows, if multiplied by the fuel flow going through the engine, to obtain the fuel mass needed for the spacecraft to conclude the journey. This means that minimising the value function corresponds to finding the minimal cost to reach the goal, which is the optimal amount of fuel for the mission. Indeed, the neural networks created are able to compute, for a certain position and state of the spacecraft, the fuel needed to get to the destination.

Using different types of initialisation for random numbers, to compute weight and bias values belonging to the first epoch, has proved to make a difference on the time needed for training and on the final performance of the neural network. A SIMD-oriented fast Mersenne Twister method has been selected to initialise these values, as it gave the best results in terms of training time and performance. Initialisation was necessary in order to always obtain the same results when executing the program, and therefore compare results from different neural networks architectures.

A memory reduction was implemented to decrease the amount of temporary storage needed by a factor of 2, in exchange for performing the computations two times sequentially on each of two subsets of the data. The activation function used was the Elliot sigmoid function, which is expressed in Equation 4.2. This function avoids exponential and trigonometric computations, meaning that less time is needed to calculate it on a simple computing hardware, compared to the more traditional hyperbolic tangent sigmoid function. The difference between these two is shown in Figure 4.1. Due to the different shape of the curves, the Elliot sigmoid function may cause the need for more training iterations, as it reaches a plateau slower than the tangent sigmoid function.

$$\sigma_E(x) = \frac{x}{1+|x|} \tag{4.2}$$

For what concerns the training process, not all data fed into the artificial neural networks is used as explained earlier, according to the Levenberg-Marquardt algorithm. Only 80% of the data from the training subset is actually used for training whereas 10% is for validation checks and the remaining 10% is for testing the neural networks. The test subset is a data set independent with respect to the training one and used to verify how good the neural networks obtained are. This way, their performance can be computed by comparing the obtained results to the real results. The validation subset is instead a data set which is used to execute validation checks. These checks consist in calculating the error of the ANNs output for this subset at each iteration, so that it

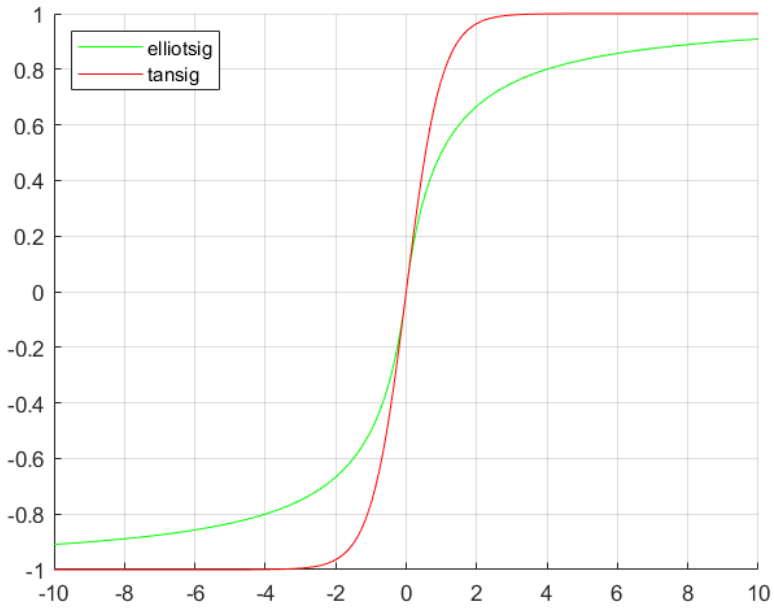


Figure 4.1: *Comparison between the Elliot sigmoid transfer function (in green) and the hyperbolic tangent sigmoid transfer function (in red).*

is possible to stop the training early. Even if the error computed on the training subset decreases at each epoch, when the one on the validation subset increases for six consecutive times, training is interrupted. This allows to avoid an over-fitting of the neural network on the training data, and to make sure that the it works well on other datasets compared to the ones used for training.

## 4.2 | ANNs Architectures

A critical decision for Artificial Neural Networks and their training is the choice of the number of hidden layers and of their artificial neurons. This is a tough choice, as it seems to depend not only on numbers of inputs and outputs, but also on the amount of training data, the amount of noise and the complexity of the function that needs to be learnt and described[16]. Hornik affirmed that a single layer is sufficient to approximate continuous functions, if using activation functions subjected to certain conditions[22]. To be safe, in this project at least two hidden layers were always implemented.

Bebis and Georgiopoulos asserted that a smaller network possesses many advantages, such as a shorter training time and short propagation delays, even though it can prove problematic during training for its tendency to identify local minima. On the opposite, bigger networks present the risk of overfitting the training data and not giving accurate results for states outside of that dataset. Moreover, they require a huge number of training examples, often not available (the number needed seems to grow linearly with the number of neurons used), and if trained with an insufficient number of examples they give poor generalisation[23].

Proof has been given over the fact that two layers neural networks can approximate any non linear function, but the number of units to be chosen for each layer is not defined and depends on the function itself and on the data used[24]. For this reason, in this work different types of ANNs architectures have been selected and trained, all with the same data.

Different architectures have been created, trained and compared in this project. At first, a neural network composed of two hidden layers was created. Each hidden layer is composed of 11 artificial neurons, as represented in Figure 4.2. Neurons per layer have been changed from 7 to 15 for different analysis.

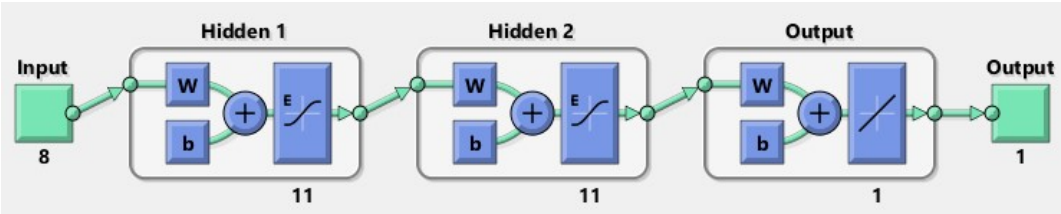


Figure 4.2: *Scheme of 2 hidden layers with 11 neurons each.*

Hidden layers were instead changed from 2 to 4. The network represented in Figure 4.3 is composed of 4 hidden layers, and still contains eleven neurons per layer.

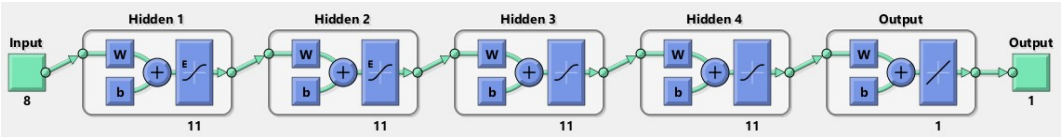


Figure 4.3: *Scheme of 4 hidden layers with 11 neurons each.*

The *fitnet* function on MATLAB was used to create these networks. Training was done on a cluster Linux Infiniband-QDR MIMD Distributed Shared-Memory, with

CPU 2x Xeon E5-2680 v3 2.50 GHz (turbo 3.3 GHz), with 12 cores being used. 3.7 TB of RAM were available.

# 5 | Results

## 5.1 | Presentation of the outputs

The feed-forward neural networks previously presented in chapter 4 have been created on MATLAB, according to the number of inputs and outputs necessary for this problem. After their creation, training has been executed on these neural networks, changing various parameters, such as the number of training epochs. For each trained network, the main parameters to be observed were the maximum absolute and relative errors computed on the outputs of the neural networks with respect to the results given by the database. Both these errors have been computed on the nominal trajectory, which being a single trajectory is composed of 100 samples, and on all test trajectories, which are 42873 trajectories. The advantage of the nominal trajectory is that the errors can be represented for each sample, whereas the test trajectories have too many samples for a graphical representation. Nevertheless, the main parameter to express the accuracy of the neural networks is the absolute error on the test trajectories, as they consist in a set of spacecraft states that deviate more from the original path, and they are not used for training. For this reason, the smaller the error is on the test trajectories, the better is the neural network.

In order to understand which neural network architectures were the most interesting to analyse, a preliminary analysis was carried out. In particular, the aim in this analysis was to reduce the computational time needed to train the network, as to be able to use a computer characterised by a CPU Intel(R) Core(TM) i7-8550U at 1.8GHz, and by a RAM with 8GB available. To be able to train a network with these computational limitations, only the first 10000 trajectories from the training dataset were used, on a network with 2 hidden layers and less than 20 neurons per layer. What was observed in this preliminary analysis was that with 20 neurons per layer a validation stop

would occur, meaning that the training was interrupted early on due to over-fitting of the neural network on the training dataset. This would often cause the network to not have a great performance. For this reason, smaller numbers of neurons per layer were analysed. With this preliminary analysis, the best network identified was the one characterised by a 2 hidden layers architecture with 11 neurons per layer, which achieved the smallest error.

Thanks to the result obtained in the preliminary analysis, the starting point for the final analysis reported in this chapter, using 342832 training trajectories, 12 cores and 3.7TB of RAM, was a neural network characterised by 2 hidden layers with 11 neurons each. The outputs obtained for this particular case are presented below, when training is executed with 100 training epochs. In particular, the maximum absolute and relative errors for both the nominal trajectory and the test trajectories are provided in Table 5.1.

Table 5.1: *Maximum relative and absolute errors for a 2 hidden layers network with 11 neurons per layer and 100 training epochs.*

| | Nominal trajectory | Test trajectories |
|---|---|---|
| Max rel err | 0.0107% | 0.1730% |
| Max abs err | 0.1505 kg | 2.5815 kg |

As expected, test trajectories have larger errors, since the test dataset is a lot larger that the nominal dataset, and as already explained its starting points often present greater deviations than the nominal case, with respect to the initial values. Nevertheless, for all considered points the maximum discrepancy between the network output and the real result is 2.58 kg of fuel.

As shown in Figure 5.1 on the left, no validation check has occurred in the 100 epochs during training, meaning that the best performance is obtained at the 100th iteration and no over-fitting of the training data has occurred. The graphic on the right instead shows the evolution of the performance of the network, computed as the mean squared error on training, validation and test datasets during each epoch. The green circle shows the point with the best performance, according to the validation check. The performance obviously decreases during training, allowing the network to obtain better results for each epoch.

The outputs obtained after the training of the neural network are shown below. In Figure 5.2, the value function of the nominal trajectory has been represented, showing both the real results extracted from the database and the outputs obtained from the neu-

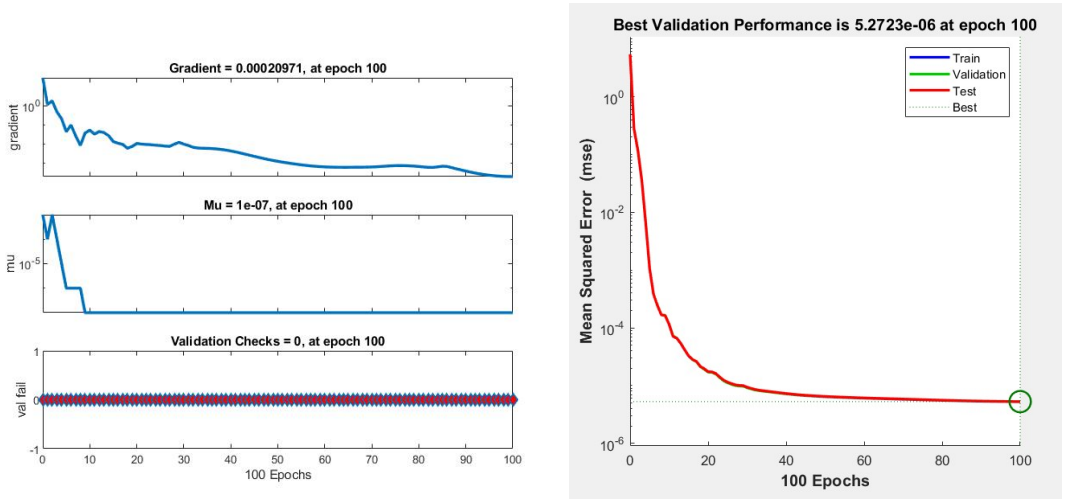


Figure 5.1: *Training for a 2 hidden layers neural network, with 11 neurons per layer and 100 epochs.*

ral network. At first sight they might seem identical, but they do present discrepancies; indeed the maximum relative error is 0.0107%, as already shown beforehand.

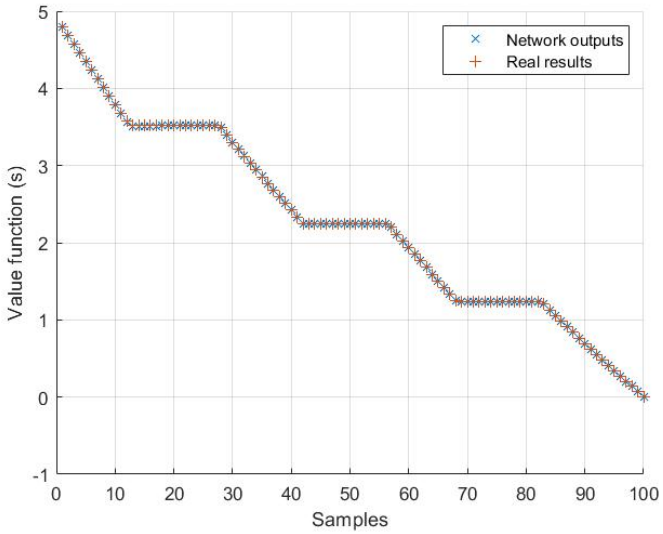


Figure 5.2: *Value function comparison between the real results and the neural network outputs.*

In Figure 5.3, the amount of fuel computed by the neural network for the nominal trajectory is represented in kilograms. As expected, it has a similar evolution compared to the value function. The first value, which is 213.26 kg, is the amount of fuel needed for the first sample of the trajectory, meaning that it is the minimum fuel needed for

the whole trajectory. On this first value the error corresponds to 0.1505 kg. The final values are very close to zero due to the fact that the spacecraft is close to its destination, and therefore does not need a lot of fuel.

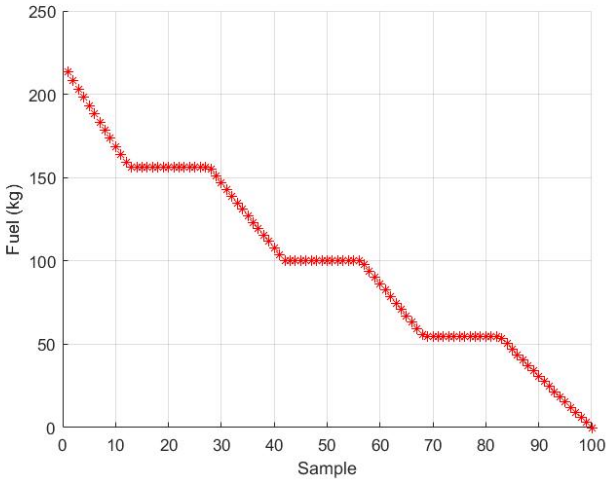


Figure 5.3: *Amount of fuel obtained with the neural network on the nominal trajectory.*

In Figure 5.4, the absolute error is shown for all samples belonging to the nominal trajectory. Greater values, including the maximum one, are found in the first samples as a larger amount of fuel is needed for those points, whereas the error decreases remarkably for the last samples, when the spacecraft is closer to destination and needs less fuel.

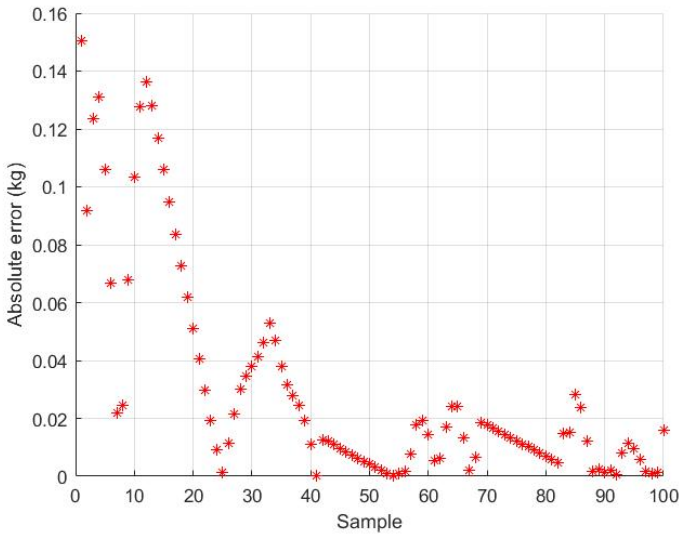


Figure 5.4: *Absolute error on the nominal trajectory.*

Abrupt changes in the slope (as it can be seen at sample 12) for the error function seem to correspond to points where the value function itself changes slope due to the presence of the plateau, where a ballistic trajectory is followed. Note that the turning point at samples 7 and 8 is only due to the absolute value applied to the error itself, and indeed it does not correspond to a plateau, as proved by the comparison with Figure 5.5. The phenomenon described can be explained as an uncertainty on the identification of the exact place where the engine is turned off.

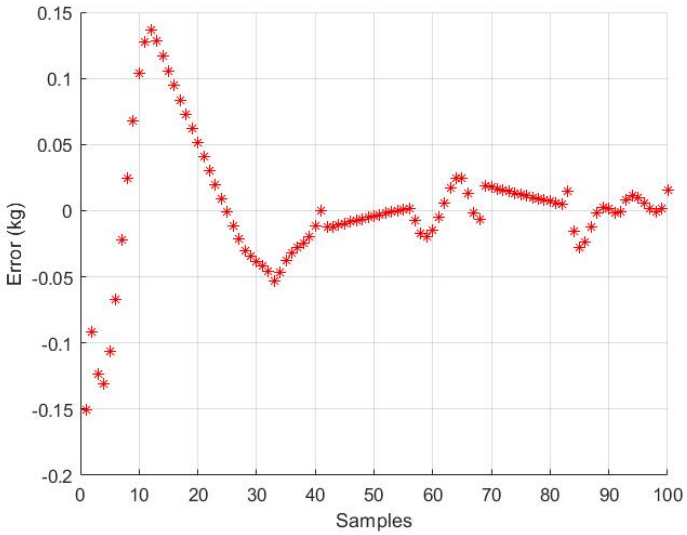


Figure 5.5: *Error (negative values also shown) on the nominal trajectory.*

In this chapter, using the neural network previously presented and its outputs as a starting point, the neural network behaviour is studied with respect to three main parameters. At first, the number of neurons per layer is modified, while maintaining the number of hidden layers at 2 and the number of training epochs at 100. Then the number of training epochs is changed, with fixed neurons per layer and hidden layers, and finally the number of hidden layers is modified to observe changes in its outputs.

## 5.2 | Analysis on neurons per layer

To analyse what happens when changing the number of neurons per layer, a comparison between two hidden layers neural networks is carried over. The number of training epochs is fixed at 100, whereas the number of neurons per layer is increased from 7 to

15. The maximum absolute and relative errors on test trajectories for these networks are shown in Table 5.2.

Table 5.2: *Results for 2 hidden layers neural networks with different numbers of neurons per layer (100 training epochs).*

| Neurons per layer | Absolute error (kg) | Relative error(%) |
|---|---|---|
| 7 | 3.1752 | 0.2154 |
| 9 | 1.9273 | 0.1395 |
| 11 | 2.5815 | 0.1730 |
| 13 | 5.1488 | 0.3427 |
| 15 | 4.0410 | 0.2732 |

No obvious behaviour can be found while changing the number of neurons per layer. The best results both in terms of absolute error and of relative error are found for the 9 neurons per layer network. A local minimum seems to be found for this value, as shown in Figure 5.6. Since the number of weights and bias values does not increase linearly with the number of neurons per layer, a diagram is also represented with the number of parameters that need to be adjusted during training on the x axis. The equation to obtain the number of parameters *wb* is Equation 5.1, where *n* is the number of neurons per layer, *hl* is the number of hidden layers and *w* is the number of weights.

$$\begin{aligned} wb &= n^{hl} + n(hl) + 9n + 1 \\ w &= n^{hl} + 9n \end{aligned} \tag{5.1}$$

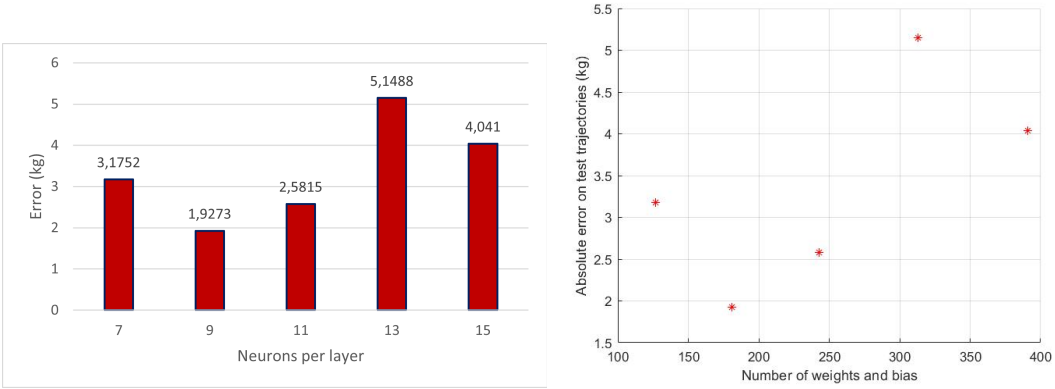


Figure 5.6: *Comparison of maximum absolute errors for 2 hidden layers networks trained with 100 epochs.*

In order to better observe the behaviour in answer to the change of neurons per layer, the same analysis has been carried out also using 300 training epochs. This has been done to make sure that the previous case did not use what might be an insufficient number of epochs compared to the complexity of the function and the amount of training data. The result shown in Figure 5.7 and in Table 5.3 is now different. The minimum absolute error on test trajectories is now achieved with the 11 neurons per layer network, reaching a value of 1.3262 kg.

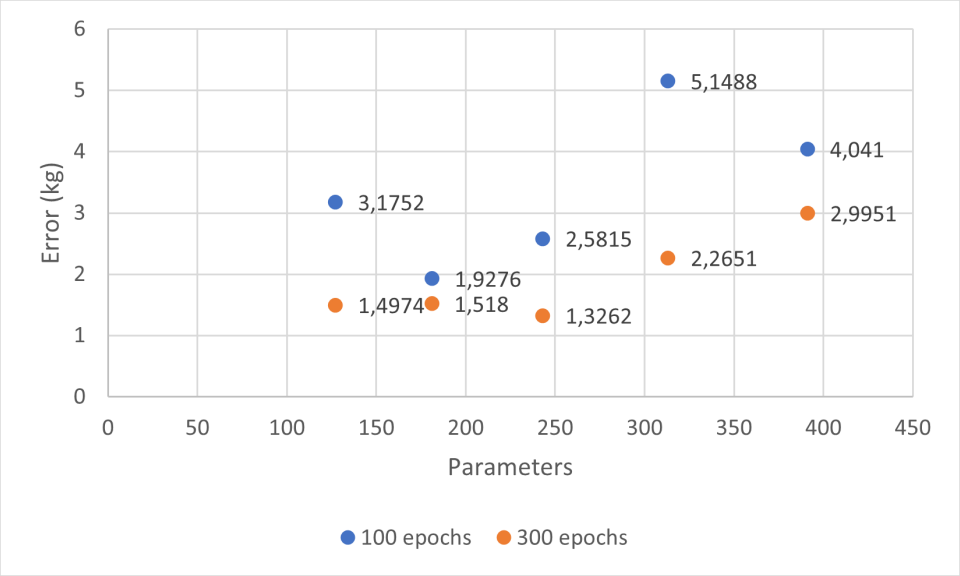


Figure 5.7: *Comparison of maximum absolute errors for 2 hidden layers networks trained with 100 and 300 epochs, as a function of number of parameters.*

Table 5.3: *Results for 2 hidden layers neural network with different number of neurons per layer (300 training epochs).*

| Neurons per layer | Absolute error |
|---|---|
| 7 | 1.4974 kg |
| 9 | 1.5180 kg |
| 11 | 1.3262 kg |
| 13 | 2.2651 kg |
| 15 | 2.9951 kg |

## 5.3 | Analysis on training epochs

In order to study the influence of the number of epochs used for the training process on the neural network output, an architecture with 2 hidden layers and 11 neurons per layer is used at first. Results from three different numbers of training epochs are compared: 100, 200 and 300 training epochs are used. This means that the performance on the training test decreases considerably, whereas it is not a certain result for what concerns the error computed on the test dataset. Nevertheless, we expect this last error to have decreased since the network is more trained. In Table 5.4 below, the outputs are displayed.

Table 5.4: *Maximum absolute errors results for 2 hidden layers networks with 11 neurons each, for different numbers of training epochs.*

| Epochs | Nominal trajectory | Test trajectories |
|---|---|---|
| 100 | 0.15153 kg | 2.5815 kg |
| 200 | 0.13277 kg | 1.9546 kg |
| 300 | 0.15549 kg | 1.3262 kg |

As expected, increasing the number of training epochs has a positive effect on the results obtained for the test trajectories, which are the ones that are considered more relevant for the overall performance of the neural network. The error on the nominal trajectory does not present a monotone behaviour, as the value increases in the last case, but being a behaviour limited to a single trajectory, this result can be disregarded compared to the test trajectories one. For the test trajectories it is possible to notice that the improvement is greater going from 200 to 300 training epochs than from 100 to 200, since the error is reduced of 0.6284 kg instead of 0.6269 kg.

In Figure 5.8, an exponential function approximating the points has been represented, as we expect the errors to continue decreasing while the number of epochs increases, but never going lower than zero. An asymptote will therefore be present along the x axis.

In Figure 5.9, the absolute errors for the nominal trajectory on 100, 200 and 300 training epochs are compared. It is interesting to notice that the maximum error appears when the greater number of epochs is used. Nevertheless, if computing for the 300 training epochs case the percentage of values that are inferior to the respective 100

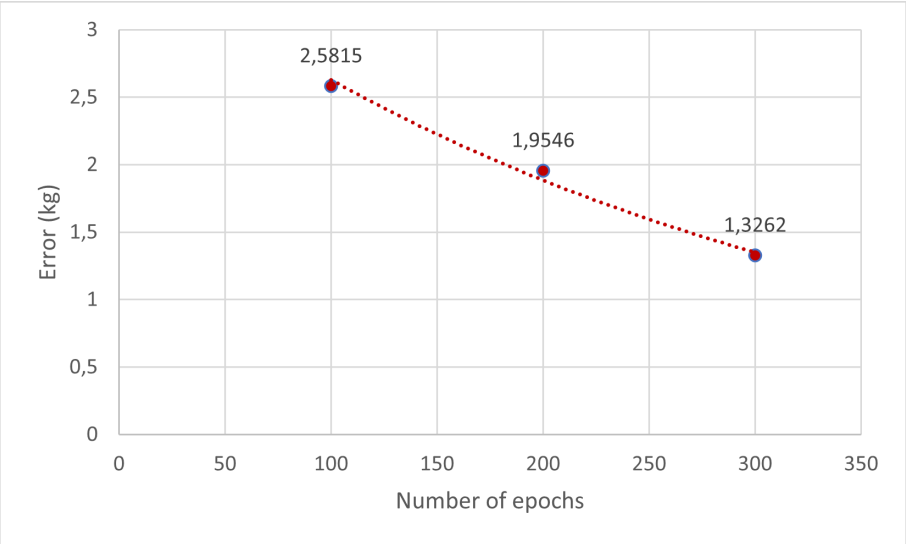


Figure 5.8: *Maximum absolute errors on test trajectories for 2 hidden layers networks with 11 neurons each with different numbers of epochs. Exponential regression displayed.*

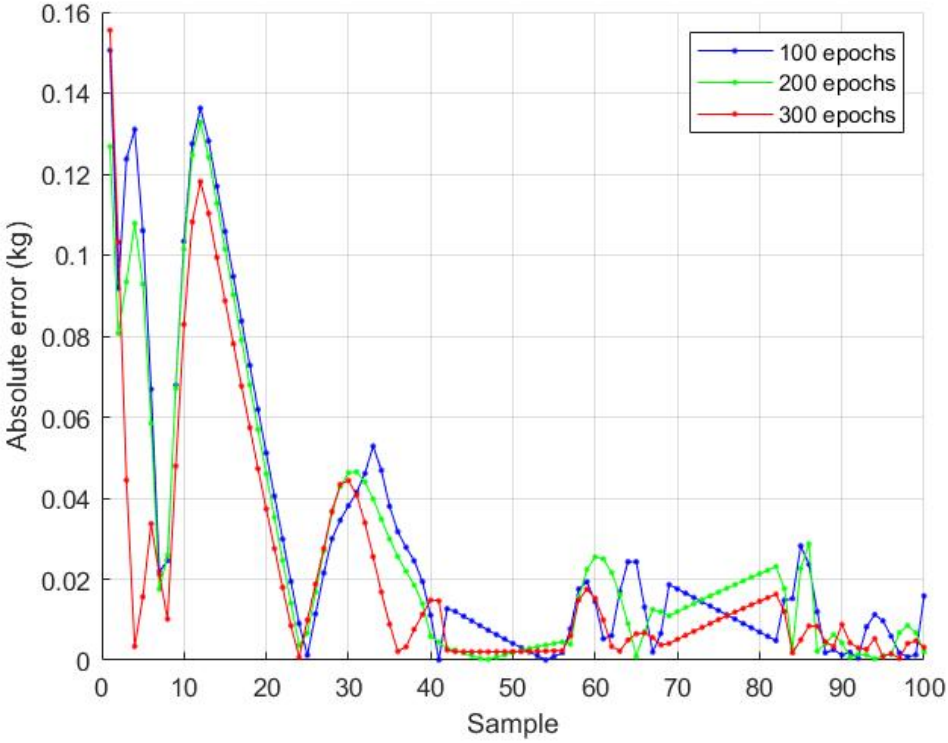


Figure 5.9: *Comparison of outputs on the nominal trajectory for different epochs used, on a 2 hidden layer network with 11 neurons per layer.*

epochs values, we obtain an improvement on 69% of cases. Comparing 100 and 200 epochs cases instead, the improvement is present in 61% of the values.

The 9 neurons per layer network architecture, with 2 hidden layers, has been analysed as well, since it allowed to obtain great results beforehand. For the increase from 200 to 300 training epochs, this time there is no improvement for what concerns the maximum test trajectories error, as shown in Table 5.5 and in Figure 5.10. An exponential function going through these points has been represented, same as the 11 neurons per layer network. Going from 100 to 200 epochs, there is still an improvement on 61% of values belonging to the nominal trajectory (Figure 5.11), and the same percentage is also found for what concerns values belonging to the test trajectories.

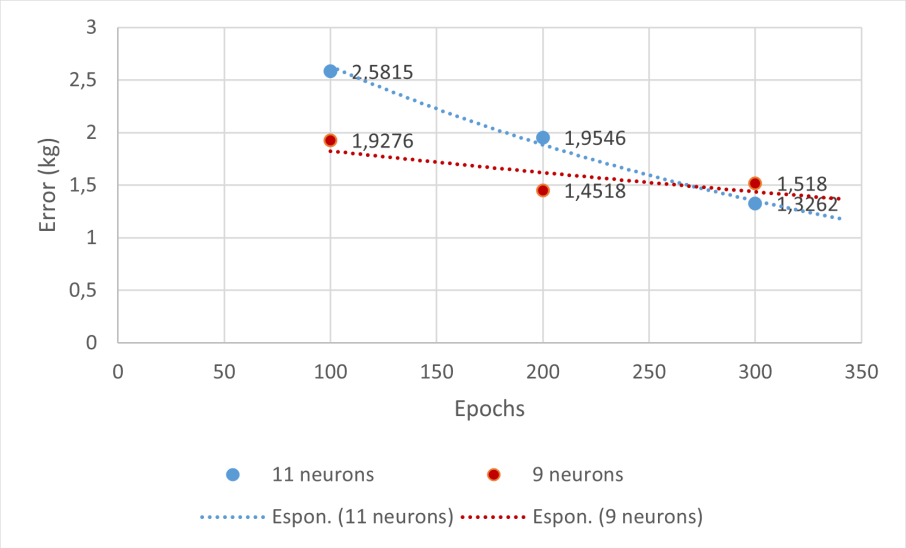


Figure 5.10: *Comaparison between 9 neurons per layer with respect to 11 neurons per layer, as a function of epochs.*

It is interesting to notice that for a high number of epochs, the 11 neurons per layer network gives better results than the 9 neurons per layer one.

Table 5.5: *Results for different number of training epochs for a 2 hidden layer network with 9 neurons per layer.*

| Epochs | Nominal trajectory error | Test trajectories error |
|---|---|---|
| 100 | 0.1473 kg | 1.9273 kg |
| 200 | 0.1402 kg | 1.4518 kg |
| 300 | 0.1260 kg | 1.3262 kg |

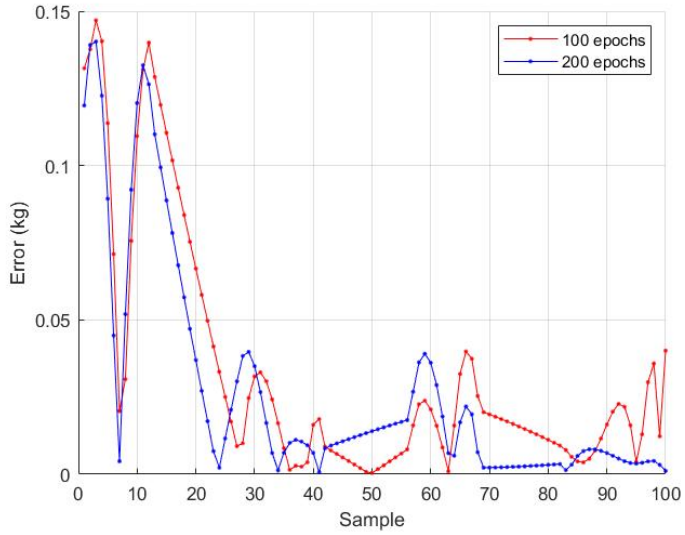


Figure 5.11: *Nominal absolute error for a 2 hidden layers network with 9 neurons per layer.*

## 5.4 | Analysis on hidden layers

Another feature that is interesting to analyse is the number of hidden layers composing the artificial neural network. Three different neural networks have therefore been compared, keeping constant the training epochs at 100 and the number of neurons per layer at 11. The three different networks have respectively 2, 3 and 4 hidden layers. The results obtained are presented in Table 5.6 and Figure 5.12.

Table 5.6: *Maximum absolute errors changing the number of hidden layers, using 11 neurons per layer and 100 training epochs.*

| Hidden layers | Nominal trajectory | Test trajectories |
|---|---|---|
| 2 | 0.1505 kg | 2.582 kg |
| 3 | 0.1411 kg | 3.068 kg |
| 4 | 0.3278 kg | 16.793 kg |

It is possible to see that the error does not decrease while increasing the number of hidden layers. A possible explanation for this can be found in the fact that, when increasing the number of hidden layers, more training epochs are probably required to reach a high accuracy, because more computations are necessary to correct all weight and bias values. These values are indeed present in a much higher number, as shown in

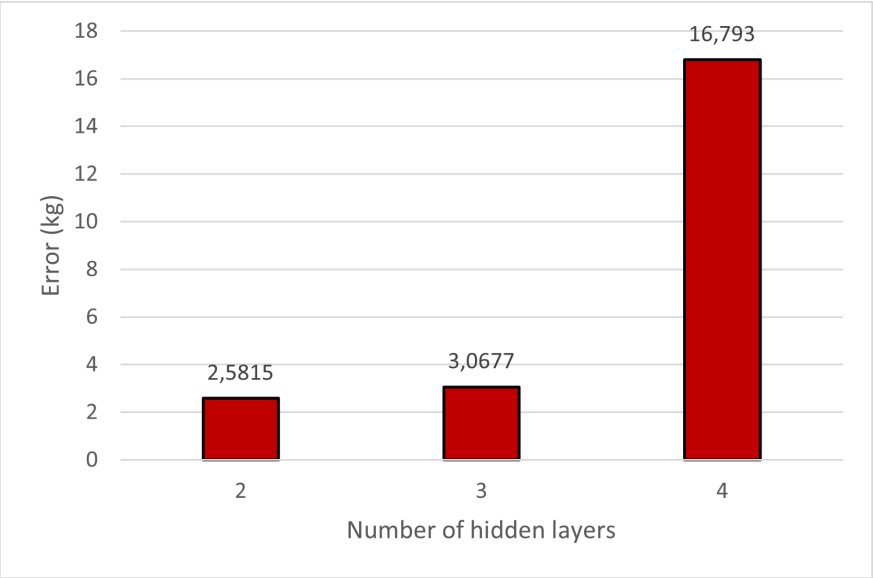


Figure 5.12: *Absolute errors for different numbers of hidden layers, using 11 neurons per layer and 100 training epochs.*

Table 5.7 and Figure 5.13. It is a reasonable assumption that the 14763 parameters for the 4 hidden layers network need more than the 100 epochs used for training to reach a high accuracy, which is probably why the maximum error is as great as 16.79 kg. The expectation is that increasing the number of iterations on this case would improve the result remarkably.

Table 5.7: *Numbers of weights and bias values for the different ANNs used, with 11 neurons per layer.*

| HL | Number of weights | Number of bias values | Number of parameters |
|---|---|---|---|
| 2 | 220 | 23 | 243 |
| 3 | 1419 | 34 | 1453 |
| 4 | 14718 | 45 | 14763 |

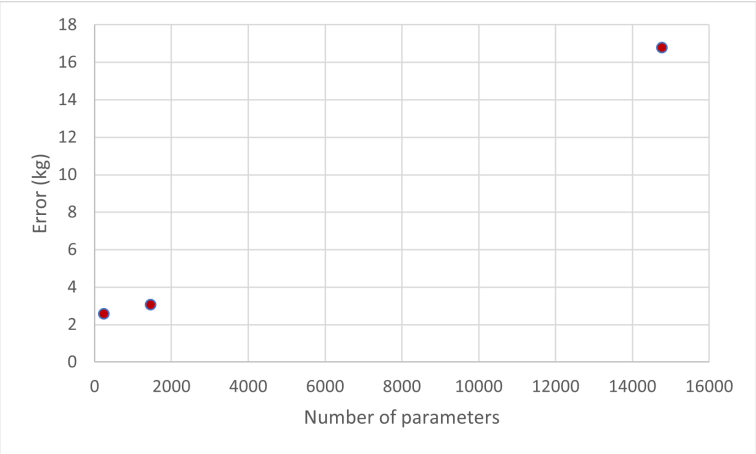


Figure 5.13: *Errors as a function of numbers of weights and bias values.*

## 5.5 | Performance of the neural networks

All neural networks analysed until now can be also studied for what concerns their performance on the test dataset, computed as shown in Equation 3.5. This result should present more as a tendency of the network to behave in a certain way, instead of the maximum error that can be found, which is what has been considered until now.

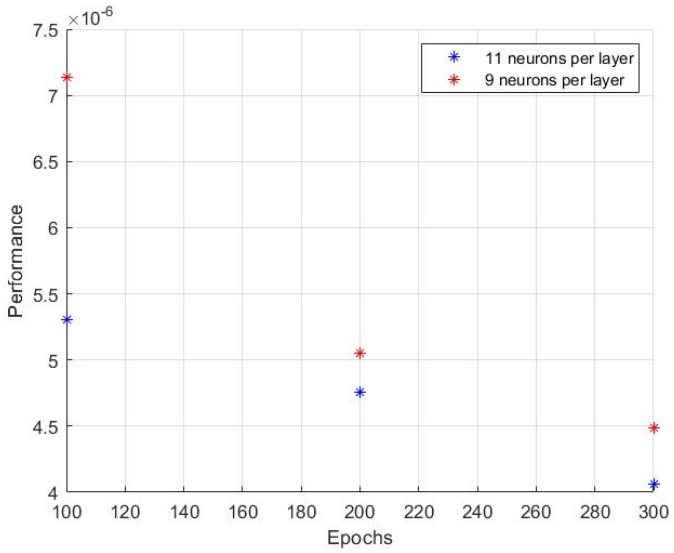


Figure 5.14: *Performance using different number of training epochs on a 11 and a 9 neurons per layer network, with 2 hidden layers.*

Since the performance on the training dataset is the parameter used by the training algorithm to improve the neural network at each epoch, we expect it to decrease when

increasing the number of epochs. This is indeed the behaviour that can be observed in Figure 5.14.

As shown in Figure 5.15, increasing the number of hidden layers does not present an advantage on the point of view of the performance. The performance increases remarkably, same as the maximum error did. The great amount of parameters to be updated at each epoch still remains the most probable cause for this behaviour, as more training epochs would be required to reach the same performance.

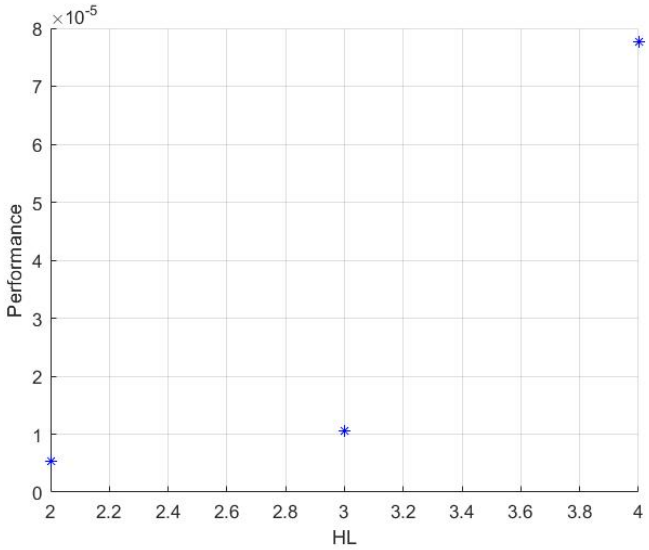


Figure 5.15: *Performance changing the number of hidden layers on a 11 neurons per layer neural network, at 100 epochs.*

An interesting result is shown in Figure 5.16, where the performance is studied as a function of the number of neurons per layer on a 2 hidden layers neural network. Neurons are increased by 2 each time, same as when the maximum error was analysed. This time, the best result is consistently found for an 11 neurons per layer neural network, instead of a local minimum at 9 neurons per layer that was found for the 100 epochs case.

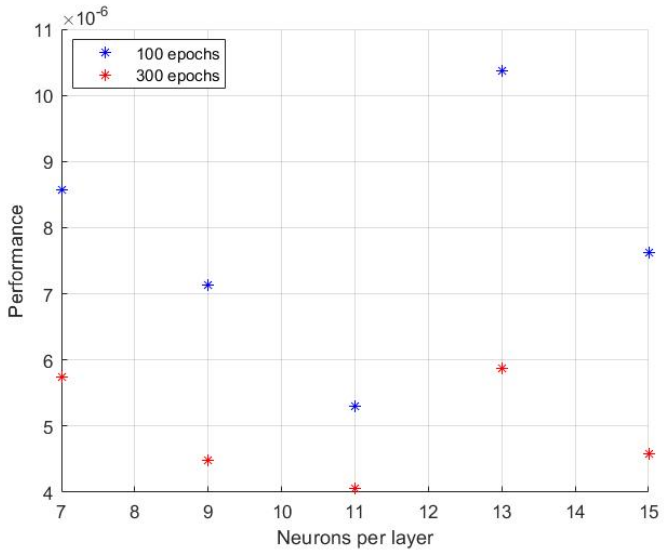


Figure 5.16: *Performance when changing the number of neurons per layer on a 2 hidden layers network, at 100 and 300 training epochs.*

# 6 | Conclusions

In this work, different Artificial Neural Networks have been analysed, comparing the maximum absolute errors computed on 42873 test trajectories. Training was done according to the Levenberg-Marquardt algorithm on 342832 trajectories, each characterised by 100 samples. Firstly, the number of neurons per layer has been increased, on a 2 hidden layers neural network, from 7 to 15 in steps of 2 using 100 training epochs. Then the number of training epochs has been changed to check its influence on the outputs, studying the 100, 200 and 300 epochs cases. Finally, the number of hidden layers has been increased from 2 to 4.

This analysis has shown that, after training is done, it is possible to obtain the amount of fuel needed from a certain point to the chosen destination in a very short time and with little computational effort. The best result was obtained for a 2 hidden layer neural network with 11 artificial neurons each and 300 training epochs (Figure 6.1), which gives a maximum absolute error on the test trajectories of 1.3262 kg out of the 1500 kg of the whole spacecraft. The maximum relative error on the initial mass is then 0.088%. On the nominal trajectory the absolute error is as small as 0.1555 kg, an order of magnitude better than on the test trajectories. This network gives the best outputs both in terms of maximum error and in terms of performance computed on the test dataset.

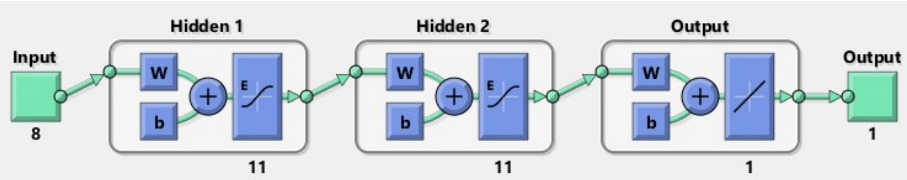


Figure 6.1: *Neural network with 2 hidden layers and 11 neurons per layer.*

A greater amount of neurons or of hidden layers does not seem to be helpful to increase the accuracy of the networks. For what concerns the number of epochs used

during training, a higher number almost always shows better results, unless a validation stop occurs.

It is interesting to notice that the same result was obtained on the preliminary analysis, where the 11 neurons per layer and 2 hidden layers were the best case, even if only 10000 trajectories were used for training. A possible explanation for this can be found in the database used and the method with which it was created.

The advantages of the neural network created are multiple. Firstly, it allows to obtain the optimal fuel needed solving problems characteristic to more traditional methods such as need for high computational power and time constraints. Moreover, no knowledge of the problem is required to use it, as it is only necessary to insert the 8 input values for the state of the spacecraft into the network.

Further possibilities arise thanks to the results obtained here. In particular, a neural network could be created in order to obtain the thrust magnitude and direction, therefore increasing the number of outputs for the neural network. This would probably mean that more iterations than what was deemed necessary in this work would be required to reach a similar accuracy. Nevertheless, the advantage would be remarkable, as this network would make the spacecraft guidance independent from ground operators. Other than independence, it would also gain the advantage of real-time implementation, which is an important achievement for space transfers, as the reaction times for the spacecraft would decrease meaningfully.

Another possibility for future works consists in combining ANNs with more traditional optimisation and control methods, which would allow to also handle constraints on variables[9]. This would allow to explore more complex problems than the ones mentioned beforehand.

# Bibliography


[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. 3rd ed. Pearson, 2009.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[3] Pamela McCorduck and Cli Cfe. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. CRC Press, 2004.

[4] M.W. Shelley. *Frankenstein, Or, The Modern Prometheus*. Frankenstein, Or, The Modern Prometheus: In Three Volumes. Lackington, Hughes, Harding, Mavor, & Jones, 1818. ISBN: 9781975957124. URL: https://books.google.it/books?id=%5C_mmw-ofzWQMC.

[5] E.T.A. Hoffmann. *The Sandman*. Re-Image Publishing, 2018. ISBN: 9783964547200. URL: https://books.google.it/books?id=xuV4DwAAQBAJ.

[6] I. Asimov. *I, Robot*. Robot novels. HarperCollins Publishers Limited, 2013. ISBN: 9780007532278. URL: https://books.google.it/books?id=vwb5mgEACAAJ.

[7] Alfred North Whitehead and Bertrand Russell. *Principia mathematica to* 56*. Cambridge University Press, 1997.

[8] Dario Izzo, Marcus Märtens, and Binfeng Pan. "A survey on artificial intelligence trends in spacecraft guidance dynamics and control". In: *Astrodynamics* 3.4 (2019), pp. 287–299.

[9] Runqi Chai et al. "Review of advanced guidance and control algorithms for space/aerospace vehicles". In: *Progress in Aerospace Sciences* 122 (2021), p. 100696. ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2021.

100696. URL: https://www.sciencedirect.com/science/article/pii/S0376042121000014.

[10] Carlos Sánchez-Sánchez and Dario Izzo. "Real-time optimal control via deep neural networks: study on landing problems". In: *Journal of Guidance, Control, and Dynamics* 41.5 (2018), pp. 1122–1135.

[11] Guanya Shi et al. "Neural lander: Stable drone landing control using learned dynamics". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9784–9790.

[12] Andrea Scorsoglio et al. "Image-based deep reinforcement meta-learning for autonomous lunar landing". In: *Journal of Spacecraft and Rockets* 59.1 (2022), pp. 153–165.

[13] Dario Izzo and Ekin Öztürk. "Real-time guidance for low-thrust transfers using deep neural networks". In: *Journal of Guidance, Control, and Dynamics* 44.2 (2021), pp. 315–327.

[14] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[15] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[16] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. "Introduction to multi-layer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[17] Jeongho Cho, José Carlos Principe, Deniz Erdogmus, and Mark A Motter. "Modeling and inverse controller design for an unmanned aerial vehicle based on the self-organizing map". In: *IEEE Transactions on Neural Networks* 17.2 (2006), pp. 445–460.

[18] URL: https://zenodo.org/record/3613772#.Y4uAfHbMJEa.

[19] MJH Walker, B Ireland, and Joyce Owens. "A set of modified equinoctial orbit elements". In: *Celestial mechanics* 36.4 (1985), pp. 409–419.

[20] I.N. da Silva, D.H. Spatti, R.A. Flauzino, L.H.B. Liboni, and S.F. dos Reis Alves. *Artificial Neural Networks: A Practical Course*. Springer International Publishing, 2016. ISBN: 9783319431628.

[21] Martin T Hagan and Mohammad B Menhaj. “Training feedforward networks with the Marquardt algorithm”. In: *IEEE transactions on Neural Networks* 5.6 (1994), pp. 989–993.

[22] K. Hornik. “Some new results on neural network approximation”. In: *Neural Networks* 6.8 (1993), pp. 1069–1072. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(09)80018-X. URL: https://www.sciencedirect.com/science/article/pii/S089360800980018X.

[23] George Bebis and Michael Georgiopoulos. “Feed-forward neural networks”. In: *IEEE Potentials* 13.4 (1994), pp. 27–31.

[24] Věra Kůrková. “Kolmogorov’s theorem and multilayer neural networks”. In: *Neural Networks* 5.3 (1992), pp. 501–506. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(92)90012-8. URL: https://www.sciencedirect.com/science/article/pii/0893608092900128.

# List of Figures


1.1 *The Bombe, machine developed by Alan Turing during World War II.* . . 4

1.2 *Garry Kasparov playing chess against Deep Blue.* . . . . . 5

2.1 *Schematic representation of a biological neuron.* . . . . . 12

2.2 *Scheme of an Artificial Neural Network* . . . . . 13

2.3 *Some examples of activation functions.* . . . . . 14

3.1 *Nominal trajectory. In red, the Earth orbit can be observed, whereas the yellow line represents Venus orbit.* . . . . . 18

3.2 *Nominal value function.* . . . . . 19

3.3 *Diagram of a two hidden layer neural network with notation.* . . . . . 20

4.1 *Comparison between the Elliot sigmoid transfer function (in green) and the hyperbolic tangent sigmoid transfer function (in red).* . . . . . 27

4.2 *Scheme of 2 hidden layers with 11 neurons each.* . . . . . 28

4.3 *Scheme of 4 hidden layers with 11 neurons each.* . . . . . 28

5.1 *Training for a 2 hidden layers neural network, with 11 neurons per layer and 100 epochs.* . . . . . 33

5.2 *Value function comparison between the real results and the neural network outputs.* . . . . . 33

5.3 *Amount of fuel obtained with the neural network on the nominal trajectory.* . . . . . 34

5.4 *Absolute error on the nominal trajectory.* . . . . . 34

5.5 *Error (negative values also shown) on the nominal trajectory.* . . . . . 35

5.6 *Comparison of maximum absolute errors for 2 hidden layers networks trained with 100 epochs.* . . . . . 36

5.7 *Comparison of maximum absolute errors for 2 hidden layers networks trained with 100 and 300 epochs, as a function of number of parameters.* 37

5.8 *Maximum absolute errors on test trajectories for 2 hidden layers networks with 11 neurons each with different numbers of epochs. Exponential regression displayed.* . . . . . 39

5.9 *Comparison of outputs on the nominal trajectory for different epochs used, on a 2 hidden layer network with 11 neurons per layer.* . . . . . 39

5.10 *Comaparison between 9 neurons per layer with respect to 11 neurons per layer, as a function of epochs.* . . . . . 40

5.11 *Nominal absolute error for a 2 hidden layers network with 9 neurons per layer.* . . . . . 41

5.12 *Absolute errors for different numbers of hidden layers, using 11 neurons per layer and 100 training epochs.* . . . . . 42

5.13 *Errors as a function of numbers of weights and bias values.* . . . . . 43

5.14 *Performance using different number of training epochs on a 11 and a 9 neurons per layer network, with 2 hidden layers.* . . . . . 43

5.15 *Performance changing the number of hidden layers on a 11 neurons per layer neural network, at 100 epochs.* . . . . . 44

5.16 *Performance when changing the number of neurons per layer on a 2 hidden layers network, at 100 and 300 training epochs.* . . . . . 45

6.1 *Neural network with 2 hidden layers and 11 neurons per layer.* . . . . . 47

# List of Tables


5.1 *Maximum relative and absolute errors for a 2 hidden layers network with 11 neurons per layer and 100 training epochs.* . . . . . 32

5.2 *Results for 2 hidden layers neural networks with different numbers of neurons per layer (100 training epochs).* . . . . . 36

5.3 *Results for 2 hidden layers neural network with different number of neurons per layer (300 training epochs).* . . . . . 37

5.4 *Maximum absolute errors results for 2 hidden layers networks with 11 neurons each, for different numbers of training epochs.* . . . . . 38

5.5 *Results for different number of training epochs for a 2 hidden layer network with 9 neurons per layer.* . . . . . 40

5.6 *Maximum absolute errors changing the number of hidden layers, using 11 neurons per layer and 100 training epochs.* . . . . . 41

5.7 *Numbers of weights and bias values for the different ANNs used, with 11 neurons per layer.* . . . . . 42