UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Evaluation of artificial neural network method applied to probe diagnostics in plasma

Bachelor Thesis

Author
David Villegas Prados

Supervisor
Xin Chen

Leganés, July 2018

Department of Aerospace Engineering
ESCUELA POLITÉCNICA SUPERIOR

Evaluation of artificial neural network method applied to probe diagnostics in plasma

Bachelor in Aerospace Engineering

Author
David Villegas Prados

Supervisor
Xin Chen

Leganés, July 2018

Abstract

Plasma diagnostics consist on predicting the plasma parameter by measuring their properties. Models to predict these parameters are based on theoretical equations, which sometimes are complicated and tedious to solve. This thesis aims to evaluate a new method to obtain these plasma parameters using a branch of artificial intelligence, the Artificial Neural Network.

To develop this thesis, the software MATLAB was used to create the neural network. Two theories were tested: Planar Langmuir Probe and Orbital Motion Limited. And two problems were solved: forward problem to obtain the properties from the parameters and the inverse problem to obtain the parameters from the properties. The later approach is used to evaluate the parameters obtained by network with experimental data from laboratory measurements.

Artificial Neural Network was implemented successfully in both forward and inverse problems with incredible accuracy. The results obtained have created a theoretical framework for plasma diagnostic using neural networks and it has been concluded that they can work with any theory presented, as long as as proper database is obtained. Regarding the simulation with experimental data, the results were not good respect the measurements taken. Although the network performed well, the database used for training and simulation had discrepancies from realistic data since it was obtained from equations. Sources of error of this last problem are found and different approaches and work to be developed in the future are proposed.

Acknowledgments

I would like to acknowledge and thank the important people who has supported me and has been by my side during the elaboration of this project and throughout this beautiful journey, my Bachelor's degree.

First of all, I would like to express my gratitude to my supervisor Xin Chen. The person who introduced me to the intriguing world of artificial intelligence. Thank you for your support, guidance, encouragement and commitment to higher standards. You inspired and motivated me.

Also, thanks to all my friends and special people on my life, who encourage me to be myself and follow my own path everyday.

Finally, infinite thanks to my family, for the unconditional support, for giving me strength everyday and the opportunity to study what I love the most.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ANN	Artificial Neural Network
EM	Electro-magnetic
LM	Levenberg-Marquardt
LP	Langmuir Probe
LSM	Least Squares Method
MLP	Multilayer perceptron
OML	Orbital-Motion-Limited

Subscripts

i	Index of neuron in the output layer
j	Index of neuron or weight in the l^{th} layer
k	Index of neuron or weight in the $l^{th} - 1$ layer
n	Index of iteration in the increment rule for the learning algorithms
p	Index of subset
q	Index of weight or bias in the whole network

Symbols

α	Learning rate or step size of the Gradient descent algorithm
β	Factor of modification of μ
δ	Sensitivity of error to a given neuron
ϵ_0	Permittivity of medium $[F/m]$
λ_D	Debye length
μ	Convergence coefficient of the Levenberg-Marquardt algorithm
$\overline{v_e}$	Electron thermal velocity $[m/s]$
$\overline{v_i}$	Ion thermal velocity $[m/s]$

σ	Activation function
a	Neuron
A_p	Probe area $[m]$
b	Bias
C	Cost function
E	Particle energy $[J]$
e	Error between output and target
e^-	Electron charge $\left[1.61 \cdot 10^{-19} C\right]$
I	Number of outputs
I_{es}	Electron saturation current $[A]$
I_e	Electron current $[A]$
I_{is}	Ionic saturation current $[A]$
I_i	Ionic current $[A]$
I_t	Total current $[A]$
J	Number of neurons in the l^{th} layer
$J^\star$	Particle angular momentum [kg m/s]
K	Number of neurons in the $l^{th} - 1$ layer
k_B	Boltzmann constant $\left[1.38 \cdot 10^{-23}\ m^2 kg/s^2/K^1\right]$
L	Number of layers in the network
L'	Lower limit of data normalization
L_p	Probe length $[m]$
m_e	Electron mass $\left[9.11 \cdot 10^{-31} kg\right]$
m_i	Argon ion mass $\left[6.63 \cdot 10^{-26} kg\right]$
n_∞	Plasma density $\left[m^{-3}\right]$
n_e	Electron density $\left[m^{-3}\right]$
n_i	Ion density $\left[m^{-3}\right]$
o	Output of the network
P	Number of subsets
Q	Number of weights plus biases in the whole network
$r^\star$	Probe placement in plasma motor $[m]$
r_p	Probe radius $[m]$

T_e	Electron temperature $[K]$
T_i	Ion temperature $[K]$
U'	Upper limit of data normalization
V_f	Floating potential $[V]$
V_p	Bias $[V]$
V_s	Plasma potential $[V]$
v_B	Bohm velocity $[m/s]$
V_{s1}	Space potential $[V]$
w	Weight
x	Vector of weight and biases
y	Target output
z	Weighted sum plus bias

Superscripts

l	Layer index

Chapter 1

Introduction

1.1 Plasma

Plasma is known as the "Fourth state of matter". Although most people are not familiarized with it, plasma is more present that one might think. It is often said that 99.99% of the universe is composed by plasma [5]. This refer to the matter that we actually know, since other types of matter such as dark matter has not been observed and are believed to occupy the majority of the universe. Although this statement may be inexact, it is a representation of an idea: Plasma is everywhere in the universe and in enormous quantities. Any star in the universe, nebulae, intergalactic medium or solar winds are composed practically of plasma. These different form of plasma cannot be observed frequently in daily life, but natural plasma can be actually observed occasionally in meteorological phenomena such as Aurora Borealis or lightnings.

But, what is plasma? Plasma is essentially an ionized gas. Ionization is a process by which a molecule is subjected to the removal or addition of electrons leading to electrically charged atoms and free electrons [22]. In contrast to gas molecules, plasma is electrically charge which will make plasma respond to electromagnetic forces. Moreover, plasma has free electrons, so it has a current flow which renders it also electrically conductive. The degree of ionization of the plasma can be different. For example, the Sun nucleus is made of dense fully ionized plasma due to its high temperature, while nebulae are low-density clouds formed of partially ionized plasma.

However, not all ionize gas can be considered as plasma, plasma shows two distinctive properties. One of them is quasi-neutrality, evaluating plasma as a whole is neutral enough so that electron and ion densities are practically equal, $n_i \approx n_e \approx n_\infty$, but not so neutral that all electromagnetic forces vanish [5]. A distinction for large scales was made since deviations from charge neutrality can be developed in shorter scales. The characteristic parameter that evaluates this concept is the Debye length λ_D. The Debye length can be defined as the distance over which significant charge separations (and electric fields) can occur in plasma [10]. Therefore, in order for a ionized to be consider as plasma $\lambda_D << L$, where L represent the dimensions of the system.

The other property is that plasma exhibits collective behavior, which means that a plasma particle

does not only respond to a stimulus by itself, but also as an interdependent response from many particles [35]. To understand this concept, imagine a group of students in a daily routine. They arrive to class, they seat themselves, listen to the lecture, take notes and finally leave the lecture. If suddenly a fire takes places (stimulus), some students will get nervous, running around and disrupting the behavior of other students. The start of the fire was the trigger for some student to react. Nevertheless it did not affect only one student, but all of them at the same time. The body language, screams and nervousness are the forms of communication between student behaviors which stimulates other students to act the same way. Returning to plasma, this form of communication correspond to long-range electromagnetic forces. When a particle is exposed to a stimulus, say an electric field, the particles exert a force on other particles even at large distance, which in turn exert a force to the other particles and so on, as a cascade effect. Because of collective behavior, plasma is often said to behave as if it had a mind of its own [5].

So the question now is how to produce plasma and how can it be implemented to create new technologies. Ionization can be obtained by heating the gas up to really high temperatures, but this is neither a practical nor efficient method. Typically, the means of plasma formation result in providing energy to free electrons. The application of energy will result in the excitation of electrons from low energy level to high level, colliding with other electrons and freeing them. This cascade process will continue during ionization [11]. Different means to free electrons are for example photo-ionization (bombardment of a gas with photons), shining radio wave (excitation of plasma by radio frequency waves) or applying an electric field to excite the electrons and produce collision cascades.

As it was seen how plasma can be defined and produced, the interest of plasma production comes from the different applications in which plasma can be used.

- One of the most common applications and something everyone is familiar is the lightning of neon lights. The working principle can be explained in a simple way: A capsule is equipped with electrodes at the edges and the inert gas. A sufficient large voltage is applied to ensure a high electric field that will excite the electrons of the gas, freeing them from their orbit. As the electrons in excited states drop to a lower energy level, photons are released. These photons are the caused of the light.
- Another usual application is for fluorescent lamps. Their functioning is a bit different from neon light. The arc tube (lamp) contains a cathode, an anode and the inert gas (typically argon). Electron current is created in the cathode by means of applying thermionic emissions from tungsten filaments. The current will ionize the gas filled with mercury vapor and inert gas. The ionization of mercury will create UV light. This light is not observable to the human eye, so phosphors are used. When the UV light strikes them, they will convert the UV light to visible light. [4]
- Chemical reactive plasma discharges are widely used to modify surface properties of materials, this application is known as plasma processing. This technology is vital for industries such as aerospace, automotive or bio-medical. Materials and surface structures can be modified in unique ways, not attainable by other commercial method. One of this process is plasma etching. This process is used for removing material from surfaces. A chemically selective process which allows to remove one type of material while laving the other materials

unaffected. [27]

- Moreover, plasma have been long studied in reactors of nuclear fusion. These reactors could be the source of electric power in the future, replacing the current fission reactors. Radiation and pollution from fission are a severe issue nowadays, which embraces the importance of the development of fusion reactors. The main advantages of nuclear fusion are that it provides clean and safe energy as well as much less nuclear waste.
- In the field of aerodynamics, recent studies have been developed regarding the improvement in the aerodynamic performance. The idea is to used dielectric-barrier-discharge (DBD) plasma actuators. All-electric actuators are ideal for active control of flow over aerodynamic bodies. DBD plasma actuators are attached to the aerodynamic body and produce a glow discharge plasma that can be used for applications involving separation control, lift enhancement, drag reduction or flight control without moving surfaces. [8, 42]
- Finally, one of the relevant application of plasma is found in the electric propulsion. Plasma is present in the three categories of electric propulsion: electrothermal, in arcjets an arc discharge is provoked to ionize and heat the gas; electrostatic, gas is broken down into ions and electrons and ions are accelerated via an electric field; and electromagnetic, where plasma is accelerated with both electric and magnetic fields [23]. In electric propulsion the input power does not come from a energy-limited exothermic reaction, like in chemical propulsion, but from a external electric power source. Then, electric propulsion is not energy-limited, but power-limited. The low thrust capabilities and the need of a power source are the major limitations for electric propulsion, it cannot be used under the effect of strong gravitational fields or rapid maneuvers. That is why it is typically used in space missions, such as satellite station keeping, orbital manouver, altittude control and inter-planetary missions. [30]

1.2 Plasma diagnostics

Plasma is being used for many applications and currently new technologies are seeking new modes to implement this state of matter. Depending upon the utilization, plasma will present different characteristics. So it seems of extreme importance to be able to recognize the characteristics of plasma to comprehend its behavior. Understanding this phenomena and being able to interpret it correctly will allow to accomplish the objective of the application. For this purpose, plasma diagnostics is vital for any plasma study.

Plasma diagnostic is referred to the techniques implemented to diagnose the *plasma parameters* by measuring different *properties*. The objective of plasma diagnostics is to deduce information about the state of the plasma from physical processes. The different parameters than can be diagnose are density, temperature, pressure or velocity distribution.

Measuring this parameters directly is not easy. Therefore, advantange is taken from the plasma properties, from which physical model are created. Depending on the property measured, different parameters can be diagnosed. The typical properties measured are the[21]:

– Magnetic measurements, sensing magnetic field.

- Plasma particle flux, for instance, current to electrode.
- Refractive index of electromagnetic waves.
- Emission of EM waves by free electrons, for example, cyclotron emission.
- Line radiation from bound electrons of atoms and ions.
- EM wave scattering, radiation scattered by plasma particles.
- Neutral atoms.
- Ions and reactions.

As it can be perceived, plasma present properties that are quite different from the usual states of matter. For that reason, understanding these physical processes involved is rather a complicated task that requires a chain of deduction. Therefore, the knowledge of plasma physics plays an elemental role in the comprehension of plasma diagnostics [21]. Generally, plasma diagnostics need a modeling process to retrieve and interpret the results since there is no analytic solution. These modelings are based in plasma physics and involve complex and tedious mathematical analysis. Even with data, the interpretation of the measured properties is rather complicated. Commonly, the parameters to be found are not linear to those obtain from the practice of diagnostics, equations are coupled and the numerical realization of these models is long and laborious.

In this thesis, the diagnostic tool to evaluate the plasma properties will be the Langmuir probe. It was invented in 1926 by Irving Langmuir, an American scientist known because of his work in the fields of engineering, physics and chemistry. He was awarded with the Nobel Price in Chemistry in 1932. One of his principal lines of study was related to plasma studies.

Langmuir probes are powerful tools due to their simple construction and relatively easy operation principle. The basic operation of the Langmuir probe consist of a conducting wire. By varying its bias with respect to plasma potential, it emits or collects electrons from the plasma, resulting in a current. These current measurements are analyzed to obtain the current law, from where plasma parameters will be found.

1.3 Artificial Neural Network

The third main branch of this thesis is the Artificial Neural Network (ANN). As it was stated, the results obtained from the practice of plasma diagnostic are highly nonlinear and the implementation of mathematical algorithm is needed. Instead of using tedious iterative procedures, ANN will serve as a much faster tool for the analysis of data and noise will be better managed.

ANNs are inspired in the way the human brain performs a task and tries to model its behavior in the form of algorithm [17]. This modeling technique is special; neural networks are not programmed to do a particular task, but they are trained for it [3]. It means that when performing one task, the computer program improves its performance as it learns from experience [32].

This concept can be understood with an example of the checkers game, as demonstrated by Arthur Samuel in 1959. Arthur developed a checkers program and he made it play against itself thousand

and thousands of times. By the gain of this experience, the program learned which were good and bad positions on the table, improving its performance each time. Eventually, the program was able to beat Arthur Samuel, although it had not played against him [44].

ANNs attempts to imitate the learning activities of biological neural networks in a simpler manner [37]. Therefore, a good understanding of their behavior will provide easier explanation when explaining its principles.

1.3.1 Biological background

The brain consists of an immense number of interconnected elements, named neurons. The main three components of the neuron are the cell body, the axon and the dendrites.

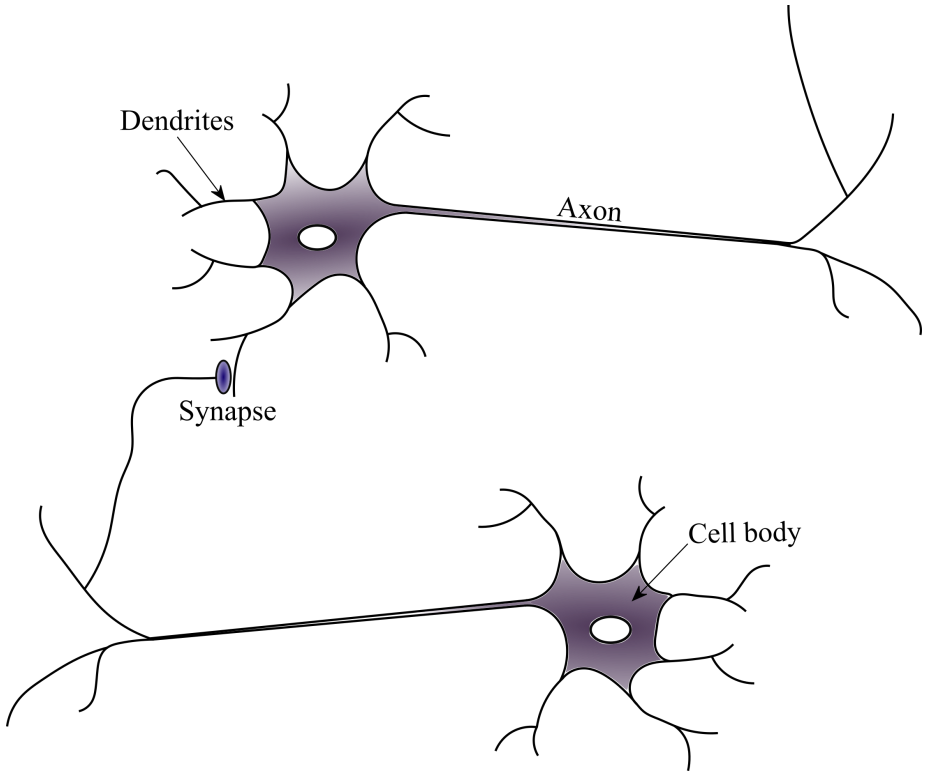

Figure 1.1: Schematic representation of two biological neurons

The cell body is the central part of the neuron. It controls and regulates the neuron functions. It receives information from the dendrites and transmits information through the axon. The point of junction between an axon and a dendrite is called a synapse. The synapse permits a neuron to pass information to another neuron via electrical signal. [2]

One neuron may have tens of thousands of synaptic connections with other neurons [41]. In this connections is where memory is stored. When learning to do some some task, neurons are activated and complex chemical processes strengthen the connection between neurons. This strength results

in information storage that will generate the memory. The more times the task is repeated, the stronger these connections will be, resulting in the automation of the task.

Depending on the task to be achieve, the strengths of these individual synapses along the neural network will be different [14]. This does not mean that for each different activity a new complete structure must to be formed. The neural network has a great plasticity, meaning that it works in such a way that past knowledge is used to learn and modify the strengths to gain other knowledge.

1.3.2 Analogy to Artificial Neural Networks

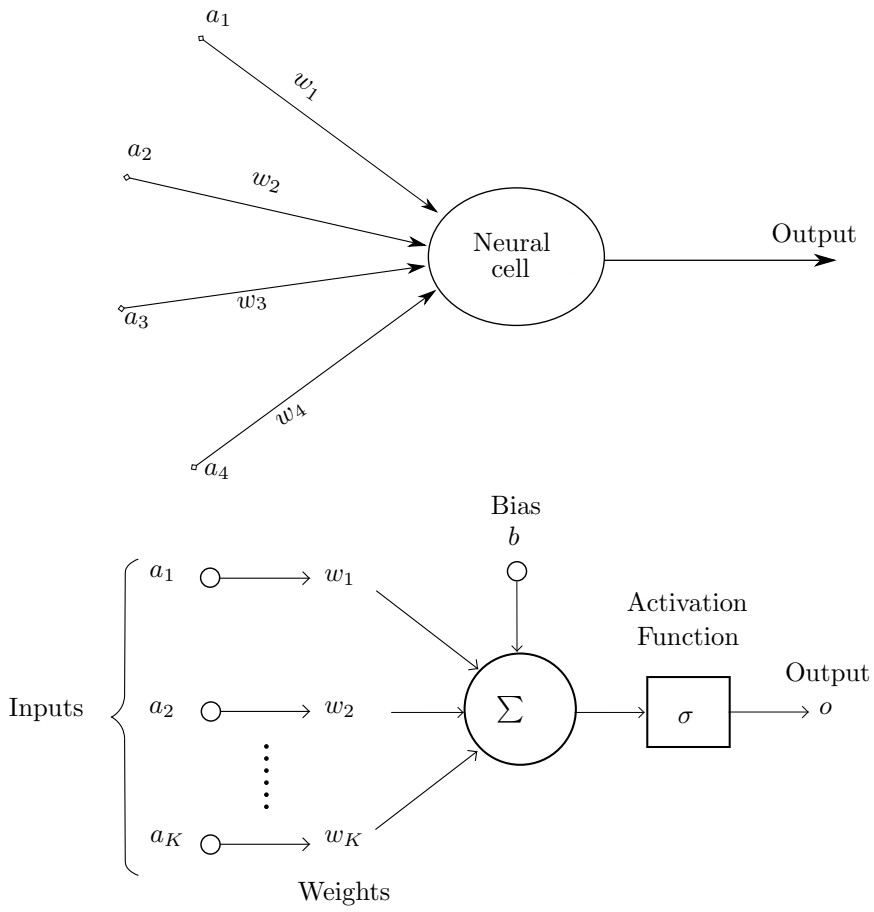

Figure 1.2: Analogy of biological neural network to artificial neural network.

An analogy to biological neural networks is shown in Figure 1.2. The inputs reflect the information coming from the axons to the dendrites. The weights resemble the synaptic strength, where

information is intrinsically stored. The neural cell is where information is processed. In ANN, this processing consist in simple calculation such as sums and multiplications. Information arrives to the neural cell, it is modified and the new information is passed to other neural cells [17]. The information is stored in the form of weight and biases. The output of one cell will be the input of another, this represents the information transmitted from neuron to neuron through the axons.

The simplified model of Figure 1.2 represents the working principle of one single neuron, called a perceptron. In a complete neural network, a neuron receives information from many neuron and transmits information to many others. This sharing process will convert the initial information into useful outputs.

However, this process is not achieved out of nothing. If data is presented to the network and no instructions are given, the network will behave as it pleases and with no objective. The network needs to know what is its function and how to accomplish the desire task.

This learning process is achieved by training the network. Analogically to biological network, ANN will modify the inter-neuron connections strengths (synaptic weights) to acquire knowledge. The weights among the network are changed so that error is minimized. The procedure to obtain this knowledge is known as *training*, and mathematically it is achieved with the *learning algorithm* [17].

The advantage of using neural networks over conventional algorithms lies in two main aspects. The first one is the simplicity. ANN uses simple computational operations. It converts succesfully a complex problem into a set of linear equations easy to solve. The other aspect is its high parallelism. In conventional algorithms if one transistor fails, the whole system fails. However, ANN has a great insensitivity to damage of few cells [13]. ANN is also able to adapt to new environments learning from previous experiences. Moreover, due to this over-determined system, neural networks are much less sensitive to noise [17]. Artificial Neural Network is used in many fields with multitude of applications. Among others these can be:

Field	**Applications**
Aerospace	Performance aircraft autopilote, flight path simulation and aircraft control systems
Defense	Target tracking, facial recognition, radar and image signal processing and noise suppression
Financial	Loan adviser, mortgage screening, portfolio trading program and price prediction
Manufacturing	Manufacturing process control, visual quality systems and machine maintenance analysis
Medical	Cancer cell analysis, optimization of transplant times, probability of a patient with a disease
Telecommunications	Real-time translation of spoken language, customer payment and data compression
Speech	Speech recognition, vowel classification and text to speech synthesis.

Table 1.1: Applications of ANN in different fields [14]

1.4 Objectives

Plasma diagnostics has different approaches for predicting the values of plasma parameters. They are based on theoretical models developed under different assumptions. The practice of plasma diagnostic is a complicated procedure that requires difficult chains of deduction. The objective of this thesis is to evaluate a new method for predicting plasma parameters, the Artificial Neural Network.

The mathematical software MATLAB will be used for the application. The goal is to establish a framework based on theoretical models of plasma diagnostics and used that framework to apply it to experimental databases.

Two theories will be implemented: Planar Langmuir Probe and Orbital-Motion-Limited. First, the ANN will be evaluated to obtain the measured property (current) from a set of parameters. This will serve as a tool to understand ANN, its performance, behavior and to find the necessary tools for its correct functioning. It will also help to build the ANN structure to optimize its efficiency and performance.

Once this basis has been cleared, the goal is to identify the plasma parameters from the properties. This is a complicated process since the number of inputs is fewer than number of outputs. The difficulties should be identified for this inverse problem and different ways to solve it, since the inverse problem represents the true activity of plasma diagnostics.

The OML theory represents a more realistic model and once the ANN has been trained with the theoretical background it will be evaluated with experimental data. The main objective of this thesis is establishing the basis and working towards the experimental frame, evaluating how it performs along the way until it can be implemented with laboratory measurements. The network should be able to predict perfectly the plasma parameters when the measured current law is given. In case of discrepancies and error in the experimental application, the sources of error as well as modifications to improve the performance should be determined. A special emphasis is done in the analysis of the experimental results. Due to the inability to obtain perfect values of the plasma parameter, it is difficult to estimate the error made with the neural network, since it does not have a concrete value to compare. Then, the results will be compared against the theory, completing an evaluation of the neural network in plasma diagnostics problems.

Chapter 2

Langmuir Probe Theories

The Langmuir probe (LP) is arguably the simplest way to measure a plasma. Its simple functioning and set up makes it really attractive. It consists of sticking a wire into the plasma and measure the current to a varying voltage. The main drawback of the method is that it is an intrusive technique, which results in an interaction of the plasma with the wire that should be taken into account. The interpretation of the I-V curves has been studied by many authors in different theoretical paper due to its complexity [6].

In this chapter, two LP theories will be studied for their later implementation into the neural network. They are: 1) LP planar theory and 2) Orbital Motion Limited Theory. A quick theoretical background on these theories will be addressed, and the equations on which the ANN data will be constructed later will be presented. Due to the nature of the LP technique, the effect of immersing an object into plasma needs to be studied. To understand the operational principle of Langmuir probe, it requires first to understand the concepts of Debye shielding and sheath.

2.1 Debye shielding and sheath

One of the fundamental characteristic that an ionized gas has to achieve to be considered as plasma is quasi-neutrality. This is met when $\lambda_D << L$, where λ_D is the Debye length and measures the capability of plasma to screen out local electric fields. It can be thought as the shielding distance or thickness sheath.

To understand this statement, let us consider a basic case as in Figure 2.1: a simple charged ball is immersed in plasma. If the ball is charged positively, it will attract electrons and if it is charged negatively it will attract ions. These charged particles will form a cloud around the ball, the shielding. Plasma outside this cloud is quasi-neutral. Without focusing on the mathematical derivation of this parameter, its value is given:

$$\lambda_D = \sqrt{\frac{\epsilon_0 k_B T_e}{n_\infty e^{-2}}} \tag{2.1}$$

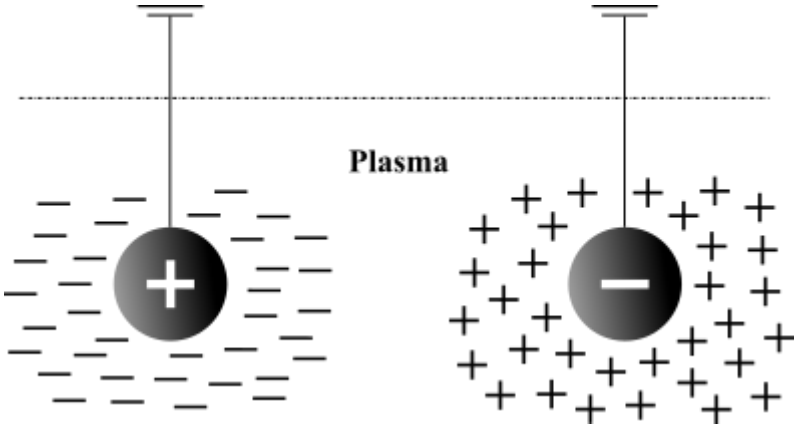

Figure 2.1: Debye shielding

Where ϵ_0 is the permittivity of the medium, $k_B T_e$ is the thermal energy, n_∞ is the density and e^- is the electron charge. From this equation it can be inferred how the shielding will change on this parameters. If the density is increased, electrons will be more compact expecting a decrease on the Debye length. Also, if the thermal energy is increased, as it was explained before, more particles will have sufficient energy to surpass the potential energy and be able to escape. This will result in the increase of the shielding thickness.

Now we know that studying plasma as a whole is neutral. Only when looking at very small volumes this conditions is violated. Nevertheless, the Langmuir probe technique is an intrusive method and the presence of the probe will disturb the plasma, so the question now is: how will plasma interact with the probe?

For this purpose, the concept of sheath must be studied. Irving Langmuir was the first person to introduce the difference between plasma and sheath: *"We shall use the name plasma to describe this region containing balanced charges of ions and electrons [...] These regions of strong field due to space charge which cover the electrodes will be referred to as sheaths"* (Langmuir, 1928) [26]. It was discover that normally non-neutral regions appeared at the plasma boundaries, these regions are known as sheaths.

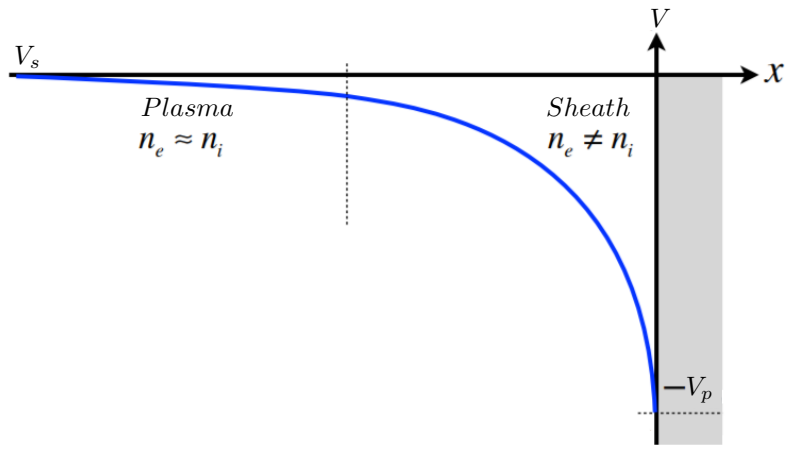

Figure 2.2: Illustration of potential drop from plasma potential to bias when $V_p < V_s$

This region represented in Figure 2.2 is macroscopic, of the order of the Debye length. The sheath formation will depend upon the potential of the probe. If the bias is at the plasma potential, no plasma sheath will be formed around the probe. On the other hand, if $V_p > V_s$ sheath will be electron rich and will expand the collecting area. However when $V_p < V_s$, electrons are repelled and a well developed positive space-charge region appears. The area is little affected by the probe voltage, resulting in almost constant ion current. [20]

2.2 Planar model for Langmuir Probe

In this section it will be discuss the theory on a single Langmuir probe. The probe is introduced into the plasma and polarize to the probe potential V_p by an external circuit. The plasma parameters are calculated from the net current to probe I_p, generated when probe potential is varied respect plasma potential V_s. The current collection to a Langmuir probe is quite simple when some conditions are met: [28]

1. Planar model: $r_p >> \lambda_D$ (probe is a plate instead of a wire)
2. Cold ions: $T_e >> T_i$
3. Electrons at or near thermal equilibrium, so that they have Maxwellian velocity distribution and are non-drifting [19].
4. Collisionless plasma.
5. Electrostatic ($\frac{dB}{dt} \approx 0$) and non-magnetized ($r_{L,e}/r_p >> 1$).
6. Quasi-neutral: $n_e \approx n_i$
7. Isotropic and homogeneous.

If this conditions are satisfied, the random thermal current passing through an imaginary surface in the plasma is given by, [28]

$T_e = 3eV$	$n_\infty = 10^{16} m^{-3}$	$r_p = 2.5mm$	$T_i = 1eV$	$V_s = 25V$	$V_p = [-20, 40]\, V$

Table 2.1: Values used to build I-V curve for planar model represented in Figure 2.3

$$I_{th} = \frac{1}{4} e^- A_p \left(n_e \overline{v_e} - n_i \overline{v_i} \right) \tag{2.2}$$

where n_i and n_e the ion and electron densities, A_p is the probe collecting area, and $\overline{v_e}$ and $\overline{v_i}$ are the electron and ion mean of the magnitude of the velocity respectively,

$$\overline{v_e} = \sqrt{\frac{8 k_B T_e}{\pi m_e}} \tag{2.3}$$

$$\overline{v_i} = \sqrt{\frac{8 k_B T_i}{\pi m_i}} \tag{2.4}$$

Because of assumption 2 and since $m_i >> m_e$, it can be inferred that the ion thermal current will be much smaller than electron thermal current. Therefore, it is deduced that $I_{the,e} >> I_{the,i}$.

A theoretical I-V curve is built using the parameters of Table 2.1. The theory behind Figure 2.3 will be explained subsequently.

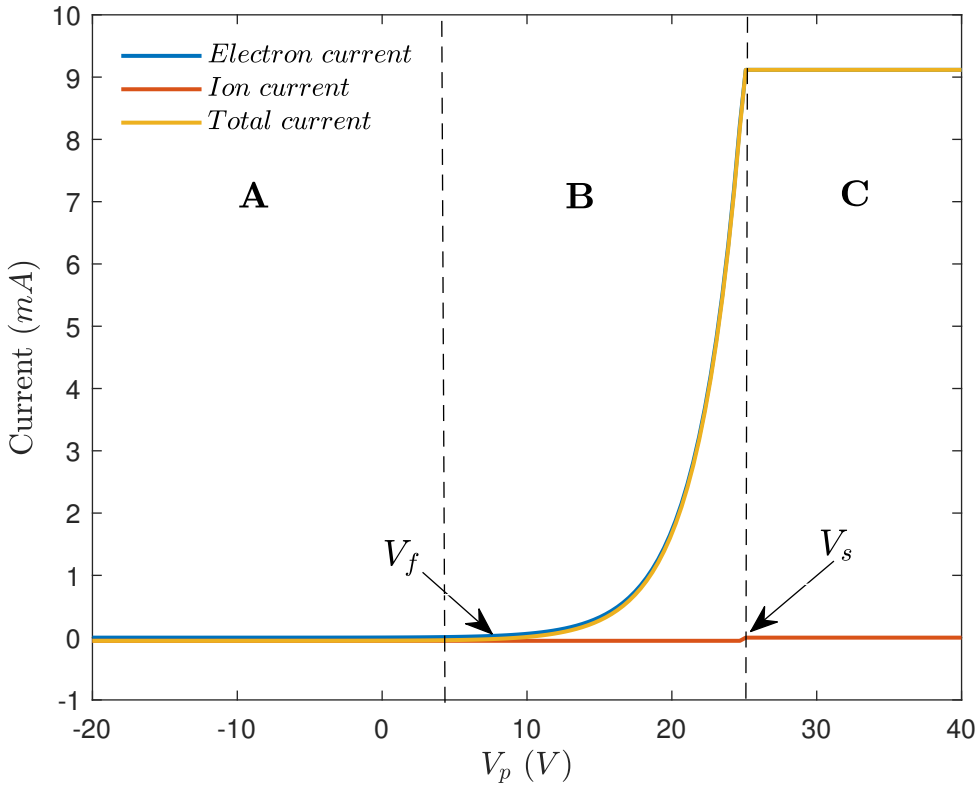

Figure 2.3: I-V curve representation for Planar LP theory and positive for electron collection

Taking the current positive when electrons are collected, the I-V curve looks like Figure 2.3. As expected from ion and electron current, three regions can be thoroughly distinguished. Region **A** happens for very negative bias voltages $V_p << V_s$ and represent the ion saturation current (I_{is}), electrons are completely repelled and ion current is saturated, meaning that it does not increase even if V_p is decreased. The region **C** represents the electron saturation current, when the probe potential is bigger than the plasma potential, $V_p > V_s$, ions are nearly completely repelled and even if V_p increases, electron current will not, since it is saturated.

Region **B** shows the current when the probe potential is decreased from V_s. The probe begins to repel electrons and only those electrons with enough initial kinetic energy can overcome the potential barrier. As V_p is decreased, electron current decreases and ion current increases. The point where the current drops to zero is known as floating potential (V_f).

2.2.1 Electron current

For region **C** the current is known as electron saturation current. This current can be calculated from Eq. 2.2, noticing that for $V_p > V_s$, the ion density around the probe will be zero, $n_i \approx 0$. Then, electon saturation current is given by, [31]

$$I_{es} = \frac{1}{4} e^{-} n_e \overline{v_e} A_p \tag{2.5}$$

and assuming the Maxwellian distribution for electrons, the electron current as a function of the bias can be obtained. Two distinguished zones can be identified for electron collection:

$$I_e(V_p) = \begin{cases} I_{es} \exp\left(\frac{e^{-}\ (V_p - V_s)}{kT_e}\right) & \text{for } V_p < V_s \\ I_{es} & \text{for } V_p \geq V_s \end{cases} \tag{2.6}$$

2.2.2 Ion current

In order to calculate the ion saturation current, the effect of a positive sheath must be taken into account. For very negative bias, $V_p << V_s$, electron current is negligible and the net current is only contributed by ion flux, resulting in a positive shielding. Langmuir stated that in order for this to happen the ions should enter the sheath region with a velocity greater than the ion sound velocity, $v_i \geq v_s$, where v_s represents the ion sound velocity. This fact was formulated later by Bohm, in the so called *Bohm criterion*. This condition requires the ions to be accelerated by an electric field in the plasma, the presheath region [38]. The speed of the ions is called the Bohm velocity, expressed as: [31]

$$v_B = \sqrt{\frac{k_b T_e}{m_i}} \tag{2.7}$$

Bohm velocity is found to be with electron temperature and obviously will be greater than $\overline{v_i}$. If ion temperature was to be used, the physical effects would not be explainable. Hence, the ionic saturation curent is calculated as, [19]

$$I_{is} = I_{Bohm} = 0.6 n_i e^- A_p \sqrt{\frac{k_B T_e}{m_i}} \tag{2.8}$$

where the factor 0.6 is due to the reduction in ion density in the presheath [31]. On the other hand, if the bias is increased respect plasma potential, ions will be repelled and current will tend to 0. Then, using the Boltzmann relation, the ion current as a function of the bias is given by, [31]

$$I_i(V_p) = \begin{cases} -I_{is} & \text{for } V_p \leq V_s \\ -I_{is} \ \exp\left(\frac{e^- \ (V_s - V_p)}{kT_i}\right) & \text{for } V_p > V_s \end{cases} \tag{2.9}$$

where the minus sign indicates that the convection used is positive for electron collection. In difference with electron current, the ion current will tend to zero much quicker as it can be observed in Figure 2.3. Then, for this ideal case it can be argued that the ion current for $V_p \geq V_s$ will be zero.

2.2.3 Total currents

The I-V curve is obtained when summing both currents: electron and ion.

$$I_t = I_i + I_e \tag{2.10}$$

The floating potential now can be calculated from previous expressions. Using Eqs. 2.6 and 2.9 [31].

$$I_i(V_f) + I_e(V_f) = 0 \tag{2.11}$$

$$I_{es} \exp\left(\frac{e^-(V_f - Vs}{k_B T_e}\right) = I_{is} \tag{2.12}$$

$$V_f = V_s + \left(\frac{k_B T_e}{e^-}\right) \ln\left(0.6 \sqrt{\frac{2\pi m_e}{m_i}}\right) \tag{2.13}$$

Figure 2.3 confirms that ion current is much smaller than electron current. Going back to Eqs. 2.5 and 2.7, I_{es} is calculated using the electron thermal velocity and I_{is} using Bohm velocity, since

ions are a lot heavier than electrons, the ion velocity will be much smaller, ergo lower saturation current. Moreover, it can be appreciated how once the bias has been increased over the plasma potential, the ions are repelled much quicker than electrons, resembling a vertical jump in ion current, reaffirming the fact that when $V_p > V_s$, $I_i \approx 0$.

For the ideal planar case, the ion and electron currents reach a fixed saturation value due to the fact that the thickness of the sheath is negligible compare to the radius of the probe, $r_p >> \lambda_D$. Meaning that the influence of the sheath around the probe can be neglected and saturation currents will hardly depend on the probe bias. If this was not the case and the thickness of the sheath will increase as bias is increased, and fixed saturations current will not be achieved. [28].

Of course this is a theoretical model, with assumed conditions that in real conditions may not be met and results will differ from ideal ones. However, this simple model will be useful to understand and start developing a simple neural network to calculate the plasma parameters.

2.3 Orbital Motion Limited Theory in cylindrical Langmuir probe

The Orbital-Motion-Limited (OML) theory was first describe by I. Langmuir and Mott-Smith [33] as an approach to model the physics of a cylindrical Langmuir probe, contrary to the planar probe described in Section 2.2.

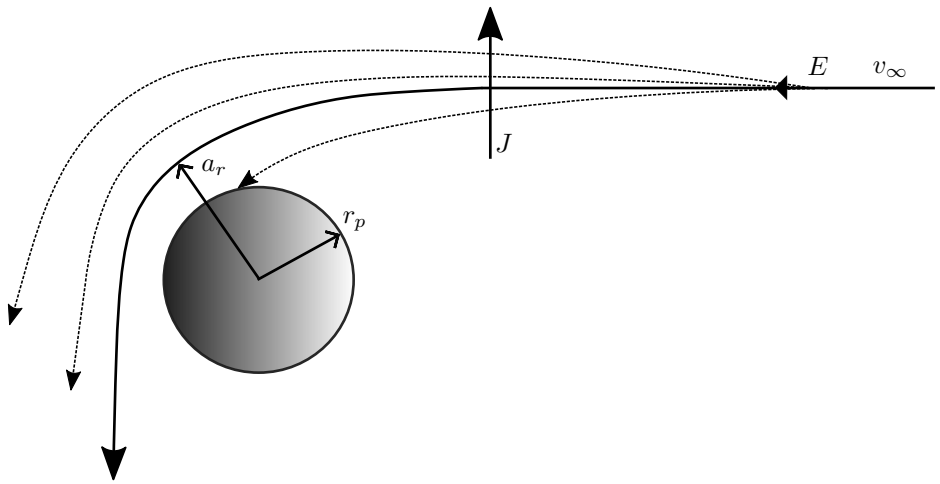

Figure 2.4: Sketch of the trajectory followed by a particle initially laying at infinity and depending on its angular momentum.

The planar probe model requires a thin sheath. For thick sheath regimes of a cylindrical probe, $r_p << \lambda_D$, the particles attracted to the probe may follow an hyperbolic trajectory around the probe [24]. Particles approach the probe with energy E at infinity, and due to the accelerating electric field they will feel an attraction force and try to hit the probe, if the distance of closest approach a_r is lower than r_p, the particle will be collected by the probe, and if $a_r > r_p$ they will miss it. From energy conservation the following equation can be expressed,

$$E = \frac{1}{2} m_\alpha \left(v_r^2 + v_\theta^2 \right) + e^- |\Delta V| \tag{2.14}$$

where $\alpha \equiv i$ for ions and $\alpha \equiv e$ for electrons, and v_r and v_θ are the radial and azimuthal velocity respectively. Also, the angular momentum for both particles is defined as:

$$J^\star = m_\alpha r v_\theta \tag{2.15}$$

Then, in Eq. 2.14, the energy can be expressed as a function of three parameters, $E = f\left(v_r^2, J, |\Delta V|\right)$.

$$E = \frac{1}{2} m_\alpha v_r^2 + \frac{J^{\star 2}}{2 m_\alpha r^2} + e^- |\Delta V| \tag{2.16}$$

Energy, E, and angular momentum, $J^\star$, are conserved so that they are constant. Also, each instant V_p is constant, meaning that ΔV is also constant. Therefore, the maximum angular momentum will be found when $v_r^2 \geq 0$.

$$J^{\star 2} = 2 m_\alpha r^2 \left(E - e^- |\Delta V| \right) \tag{2.17}$$

As the angular momentum increases, the orbit followed by the particle becomes wider, increasing the probability of missing the probe, as illustrated in Figure 2.4.

The equations describing the current collection in dimensionless form were obtained by [7] taking advantage of the conservation of the distribution functions, energy, and angular momentum [7]. The dimensionless ion and electron current (i_α) are expressed as a function of the error function (erfcx) and the three dimensionless parameters, φ_p, δ_i, and μ_i:

$$\varphi_p \equiv \frac{e^- \left(V_p - V_s \right)}{k_B T_e} \equiv \frac{e \Delta V}{k_B T_e} \qquad \delta_i \equiv \frac{Ti}{Te} \qquad \mu_i \equiv \frac{m_i}{m_e} \qquad i_\alpha = \frac{I_\alpha}{I_{the}} \tag{2.18}$$

$$i_i = \begin{cases} -\sqrt{\delta_i / \mu_i} \left[\operatorname{erfcx}\left(\sqrt{-\varphi_p / \delta_i} \right) + 2 \sqrt{-\varphi_p / (\pi \delta_i)} \right] & \text{for } \varphi_p < 0 \\ -\sqrt{\delta_i / \mu_i} \exp(-\varphi_p / \delta_i) & \text{for } \varphi_p > 0 \end{cases} \tag{2.19}$$

$$i_e = \begin{cases} \exp(\varphi_p) & \text{for } \varphi_p < 0 \\ \operatorname{erfcx}\left(\sqrt{\varphi_p} \right) + 2 \sqrt{\varphi_p / \pi} & \text{for } \varphi_p > 0 \end{cases} \tag{2.20}$$

The total current (I_t), positive for electron collection, is the sum of both ion and electron current. The currents are normalized with respect to the electron random thermal current I_{the}. I_t is then obtained according to Eqs. 2.21 and 2.22, where L_p is the probe length,

$$I_{the} = 2 \pi r_p L_p e^- n_\infty \sqrt{\frac{k_B T_e}{2 \pi m_e}} \tag{2.21}$$

$$i_t = i_e + i_i \qquad I_t = i_t I_{the} \tag{2.22}$$

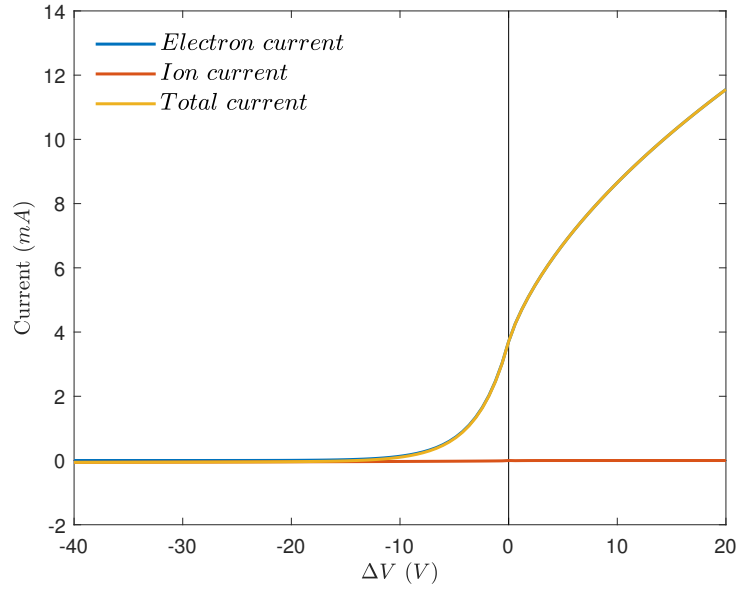

Figure 2.5: I-V curve representation for OML model (positive for electron collection)

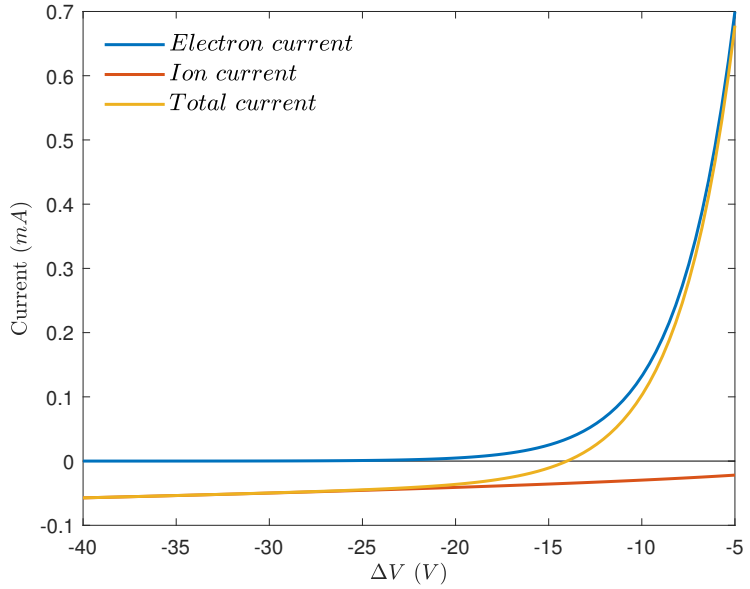

Figure 2.6: Ion current collection of the I-V curve representation for OML model (positive for electron collection)

$T_e = 4eV$	$n_\infty = 5 \cdot 10^{16} m^{-3}$	$r_p = 0.127mm$	$T_i = 1eV$	$L_p = 2mm$	$\Delta V = [-40, 20]\, V$

Table 2.2: Values used to build I-V curve for OML model represented in Figure 2.5

The I-V curve for the OML model is represented in Figure 2.5 and close-up view of ion current collection in Figure 2.6. It was calculated using the previous formulas and taking the parameters shown in Table 2.2, where $\Delta V = V_p - V_s$. In this model, there is no saturation current as in the planar probe. Since the probe is cylindrical, the sheath radius will increase when the bias is increased, leading to an expansion in the collecting area, which will result in the increase of electron current collection. Hence, no matter how much the bias is increased, the current will increase with it. When bias is decreased respect the plasma potential, the same effect will occur to the ion collection when bias is decreased as observed in Figure 2.6. It is also shown in [7] that theoretically, the plasma potential coincides with the inflection point of the I-V curve,

$$V_s \quad \rightarrow \quad \frac{\partial^2 I_t}{\partial V_p^2} = 0 \tag{2.23}$$

Chapter 3

Artificial Neural Network

This chapter aims to provide a theoretical framework on Artificial Neural Networks. ANN has been learn from scratch, and to understand their principles, assess the performance and understand the results, a good background is essential. The different components of the network, structures, working principles and mathematical scheme will be studied.

Artificial neural networks replicate analogously the human brain. The human brain is able to perform numerous task instantaneously, how much does it take to differentiate a chair from a table? Or a dog from a cat? It is instantaneously. However, for a computer this is no easy job. Expressing this type of problem in a mathematical way that the computer is able to understand would require difficult codes with no generalization for other images. Artificial neural networks were initially created to solve this kind of problems, but soon they evolved. Problems involving lots and lots of data cannot be assimilated by the human brain, such as pattern recognition or classification. The power of the ANNs lies in combining the mathematical power of a computer with a different way to approach problems inspired in how we learn and use that knowledge to keep learning.

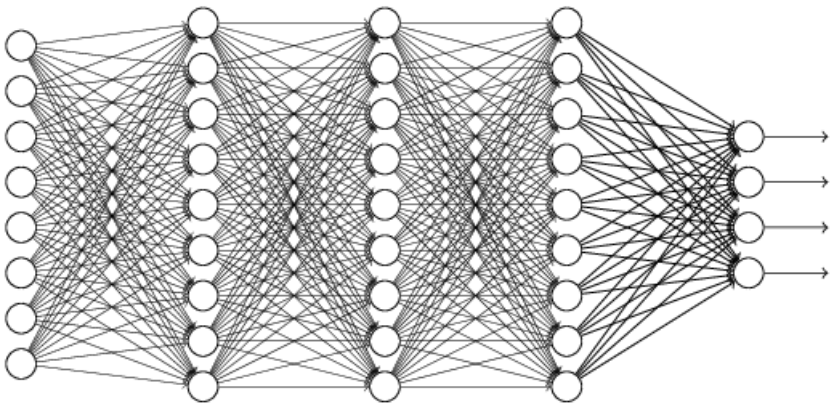

Figure 3.1: Representation of the architecture of a artificial neural network (Image Credit: Mathworks)

The simplest form of a neural network is the neuron, known also as neural cell. The neuron can be thought as a mathematical function, but for simplicity, it is better to think of it as a node that simply holds a number. The nodes are distributed along the network into three different layers: input layer, output layer and hidden layer.

The **input layer** is the first contact with the outside world. This layer contains the neurons where input data is stored. The input layer will have as many nodes as input variables and no computation takes place in these nodes, they just pass the information to the hidden layer. On the other hand, the **output layer** contain the neurons where the result is shown, as it name indicates, it represents the output that the network has yielded. It will have as many nodes as output variables.

All the process happens in the **hidden layer**. Essentially, its goal is to recognize the relations between input and output. The inputs are processed in the hidden layer to extract information from it to predict the output. This process is not ideal and not always works as expected. It will depend on many different aspects, such as handling and pre-processing of data, architecture of the network and the complexity of the problem itself.

3.1 Neural network structures

The neural network do not have a predefined structure, it can adopt any desired architecture. The only condition for an architecture to work as a neural network is that it has an input and output layer, with the corresponding weights connecting them. This types of network are names ***single layer*** network. The outputs are solely a linear combination of the inputs, so it cannot treat problems with some kind of non-linearity [36].

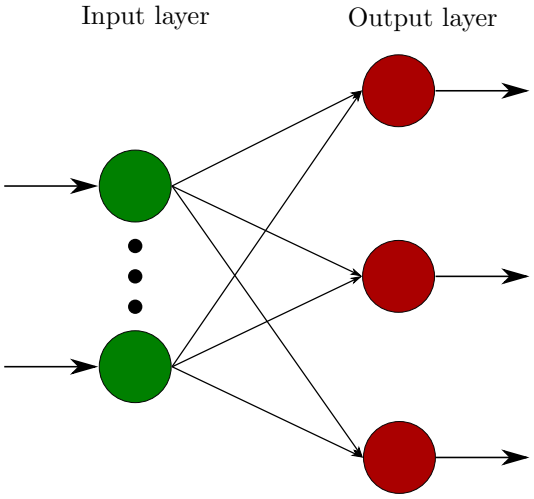

Figure 3.2: Sketch of architecture of single layer artificial neural network

When a neural network has at least one hidden layer, it is called ***multilayer network*** (or multilayer perceptron, MLP). MLP can have any number of hidden layers with any number of neurons

in each of them. Hidden neurons have the ability to enrich the network, subtracting more complex relations between input and output data [17]. The number of neurons in each layer as well as the number of layers is determined depending on the problem. Generally, adding more neurons will help to improve the performance of the network with the drawback of being more time-consuming. However, adding more neurons not always improves the performance since no more information can be extracted from the input [40].

Adding layers means adding dimensionality to the network and has a similar effect than adding neurons. Generally, if the number of hidden layers is increased the accuracy will improve since different relations can be extracted. Nevertheless, sometimes increasing the number of layers much more above the sufficient number might have the opposite effect, decreasing the performance and over-fitting the training set, losing every ability of generalization.

In Figure 3.2 and 3.3, the inputs of each layer are the output response of the preceding layer. The neurons are always connected with forward connections. This kind of neural network with no loops are called ***feedforward network*** and will be the type treated during the simulation. Moreover, it can be also said that the networks are fully connected, since one neuron in one layer is connected to every neuron in the next layer. If some of this links are missing, it it said to be partially connected.

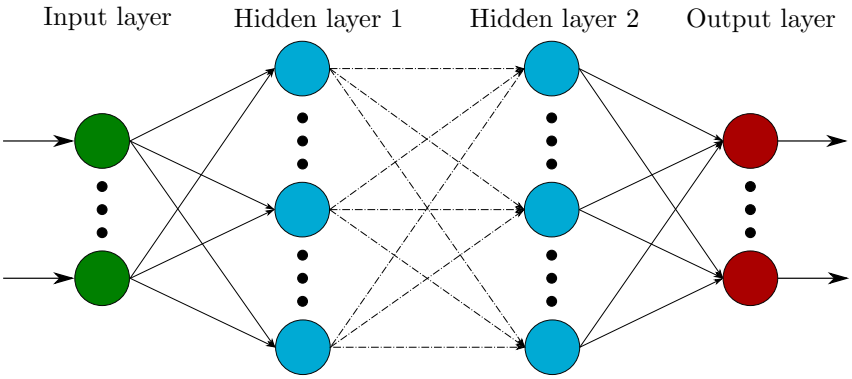

Figure 3.3: Sketch of architecture of multilayer layer artificial neural network

3.2 Principals of ANN

To understand how the neural network operates, let's consider the following notation to represent the neurons and the connecting weights between them.

$\boldsymbol{a}^{(l)} \equiv$ vector of neurons in the l^{th} layer of a network with L layers.
$w_{j,k}^{(l)} \equiv$ each weight connecting the k^{th} node in the $l^{th} - 1$ layer with the j^{th} node on the l^{th}

The weight, w, can be any number and they are always multiplied by the preceding node value. If the weight value is very large, it means that the neurons connected by this weight are of great importance and this importance decreases as it magnitude becomes smaller.

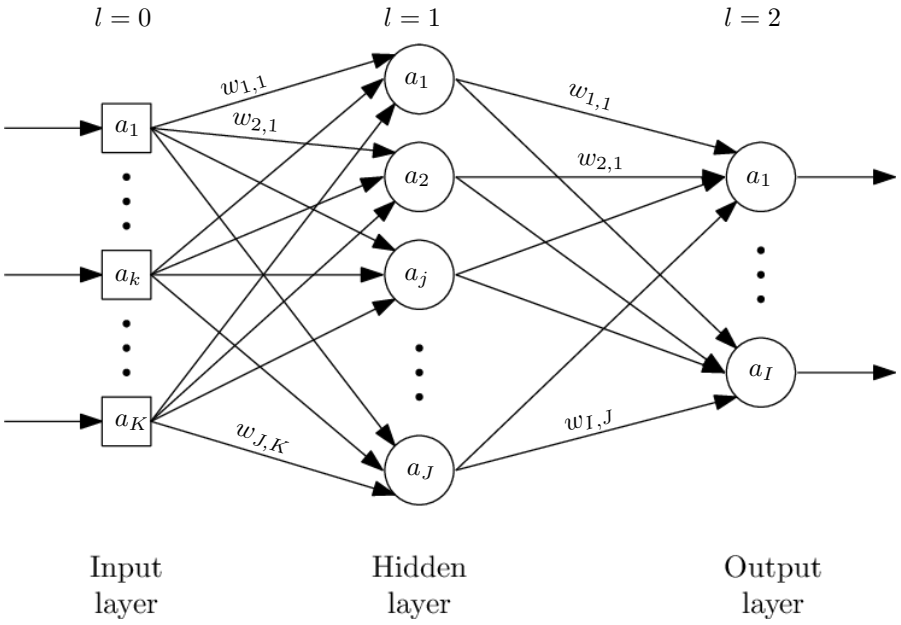

Figure 3.4: Notation of neural network with one hidden layer

In Figure 3.4, a representation of a three layer network (1 input, 1 hidden, and 1 output) is shown to understand the notation chosen. Index i will refer exclusively to the output nodes, where the total number of output neurons is given by I. The indexes j and k are referred respect to the layer that is being inspected. For example, if the datum is in layer $l = 1$, the index of layer 1 will be j and the index of layer 0 will be k. If we are looking at layer 5, this layer will have index j and layer 4 will have index k. To begin the problem, consider a neural network with a single hidden layer as represented in Figure 3.4. The weighted sum, denoted by $z_j^{(l)}$, is computed for each node as follows,

$$z_j^{(l)} = \sum_{k=1}^{K} w_{j,k}^{(l)} a_k^{(l-1)} = w_{j,1}^{(l)} a_1^{(l-1)} + w_{j,2}^{(l)} a_2^{(l-1)} + \cdots + w_{j,K}^{(l)} a_K^{(l-1)} \tag{3.1}$$

where K is the number of neurons in the $l^{th} - 1$ layer. $\boldsymbol{z}^{(l)}$ will be a vector containing the weighted sum in each neuron of the l^{th} layer. Sometimes, the weighted sum is desired to be activated meaningfully (positive sum) when the value is over certain threshold. For this purpose, a bias b is added to the weighted sum before the the next step. Therefore, in each neuron the following computation is performed,

$$z_j^{(l)} = \sum_{k=1}^{K} w_{j,k}^{(l)} a_k^{(l-1)} + b_j^{(l)} \tag{3.2}$$

The result after this computation can be anything, so it is usual that, depending on the problem, this sum is desired to be classified or squished into some specific range. It is a way to give meaning to the values obtained, and it is accomplished with activation functions.

3.2.1 Activation functions

The activation functions, $\sigma(z)$, are applied to the weighted sum plus the bias and have two objectives: to squish it into a desired range and to obtain a measure of how positive the relevant weighted sum is. Therefore, as it will be seen, if the bias is much more negative than the weighted sum, when the activation function is applied it will yield a value closer to the lower bound of the function, meaning that the neuron is not meaningfully activated. Then, the bias works as an indicator of what value the neuron should have to be "important". There are several activation functions that can be used.

Step function

The step function is a piece-wise constant function that classifies the output into two different values: 0 or 1. The output is classified depending on some threshold value that will be seen later, called the bias (b). If the weighted sum is bigger than the bias, the activation will yield 1 and if the weighted sum is lower than the bias, 0. This activation function is used in problems were only two possibles answers are possible, i.e true or false, yes or no, 1 or 0...

$$\sigma(z) = \begin{cases} 0 & \text{if} \quad \boldsymbol{w} \cdot \boldsymbol{a} + b < 0 \\ 1 & \text{if} \quad \boldsymbol{w} \cdot \boldsymbol{a} + b \geq 0 \end{cases} \tag{3.3}$$

If the value of the first neuron of the hidden layer as to be computed, $b = b_1^{(1)}$ is the threshold value, $\boldsymbol{w} = [w_{1,1}, w_{1,2}, ..., w_{1,K}]^{(1)}$ is the vector of weights from every neuron in the input layer to the first neuron in the hidden layer and $\boldsymbol{a} = [a_1, a_2, ..., a_K]^{(0)^T}$ the input vector.

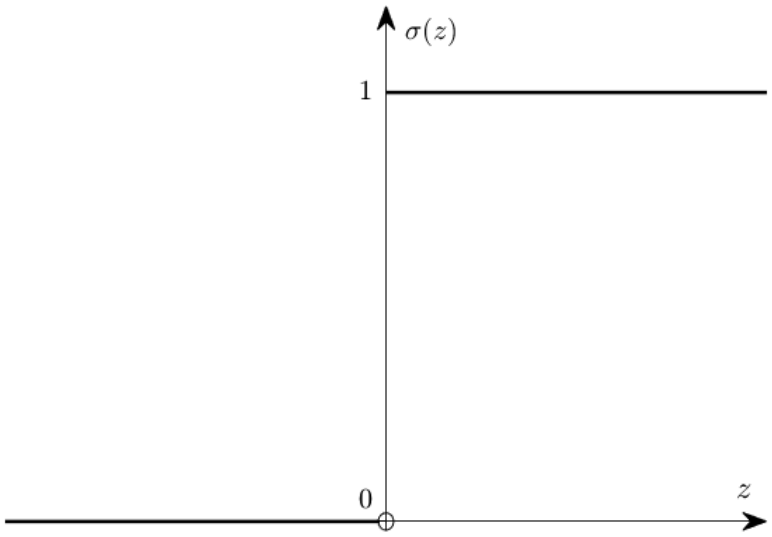

Figure 3.5: The step function

Linear function

The linear activation function gives outputs that are proportional to the input. The advantage of this activation function is that is not bounded between two values, its range is from $-\infty$ to ∞. So that problems requiring outputs outside of the range $[-1, 1]$ should have a linear activation function in the output layer.

$$\sigma(z) = Az \tag{3.4}$$

Where A is the constant of proportionality. This activation function presents a great disadvantage due to its linearity. The basic principle of the neural network is the linear combination (weighted sum) between each layer. If the activation function is linear, the output will always be a linear combination of the input, resulting impossible to solve non-linear and complex problems.

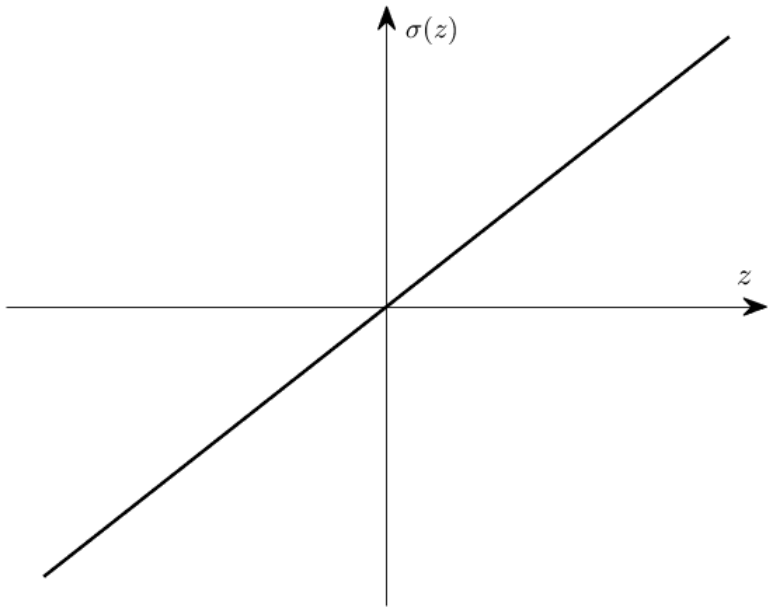

Figure 3.6: The linear function

Sigmoid function

The sigmoid function is non-linear with a "S" shape. The problem of linear combinations between layers is removed respect to the linear function. The range of this function is between 0 and 1 and it can take any value in between. Therefore, it can be used for classifier problems since it tends to bring the output to either side of the curve.

$$\sigma(z) = \frac{1}{1 + \exp^{-z}} \tag{3.5}$$

The problem that arises with this kind of activation function is the poor response when the values of the input (z) are either to big or too small. The gradient of the function in both sides becomes smaller and smaller until it is almost flat. Then, at the sides of the curve, a big increase of z will result in a small change in the output, decreasing the ability to predict value efficiently. However, for classification problem this is not an issue since the distinction between 1 and 0 is clear. The sigmoid is one of the most used activation functions.

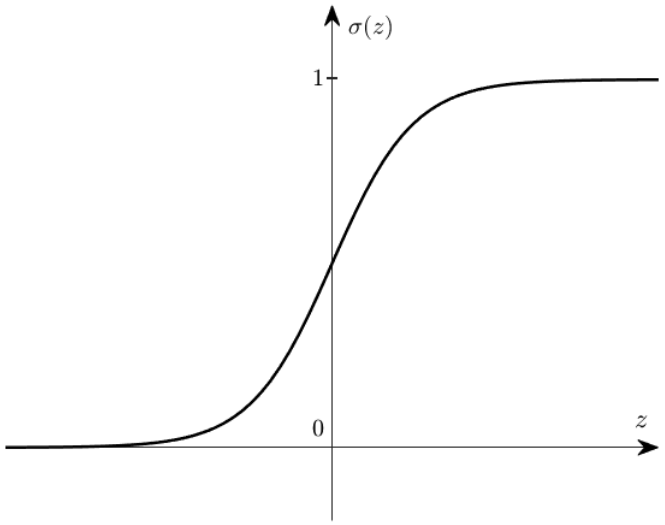

Figure 3.7: The sigmoid function

Hyperbolic tangent function

The hyperbolic tangent function is very alike to the sigmoid function . Their characteristic are quite similar with the difference that the range goes from -1 to 1.

$$\sigma(z) = \tanh(z) = \frac{2}{1 + e^{-2z}} - 1 \tag{3.6}$$

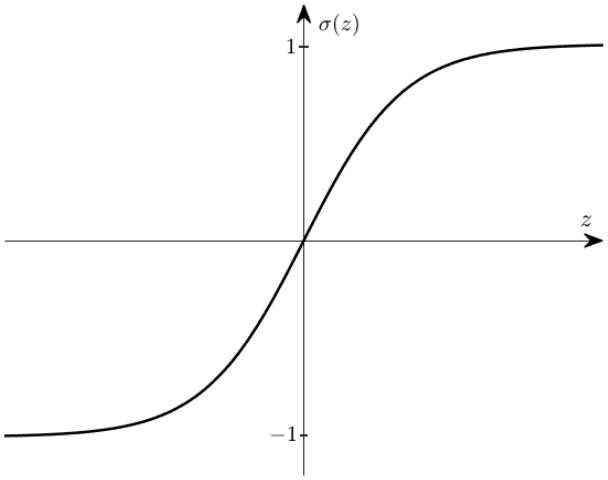

Figure 3.8: The tangent hyperbolic function

3.2.2 Matrix notation

The value of each neuron in the network can be finally computed as

$$a_j^{(l)} = \sigma(z_j^{(l)}) = \sigma\left(\sum_{k=1}^{K} w_{j,k}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right) \tag{3.7}$$

Since this process is repeated in each neuron and for each layer transition, tons of data are managed, and it is useful to express the weights and biases in matrix form. The whole transition from one layer to another can be expressed with, the matrix $\boldsymbol{W}^{(l)}$ containing all the weights from one layer to another, $\boldsymbol{b}^{(l)}$ the biases added, and $\boldsymbol{a}^{(l-1)}$ being the preceding vector of neuron that multiplies the weights.

$$\boldsymbol{W}^{(l)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,K} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{J,1} & w_{J,2} & w_{J,3} & \dots & w_{J,K} \end{bmatrix} \tag{3.8}$$

$$\boldsymbol{a}^{(l-1)} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix} \qquad \boldsymbol{b}^{(l)} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_J \end{bmatrix} \tag{3.9}$$

Then, for example, the full transition from the input layer to the first hidden layer is given by the following computation, where $\boldsymbol{a}^{(1)}$ is the vector containing the values for each neuron in the first hidden layer.

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_J^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} & \dots & w_{1,K}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} & \dots & w_{2,K}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{J,1}^{(1)} & w_{J,2}^{(1)} & w_{J,3}^{(1)} & \dots & w_{J,K}^{(1)} \end{bmatrix} \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_K^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_J^{(1)} \end{bmatrix} \right) \tag{3.10}$$

Then for the whole network the following expressions hold.

$$\boldsymbol{z}^{(l)} \equiv \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{3.11}$$

$$\boldsymbol{a}^{(l)} = \sigma^{(l)} \left(\boldsymbol{z}^{(l)} \right) \tag{3.12}$$

The activation function can be different for any layer if the problem requires so, which is the reason behind the upper-index l applied to it.

3.3 Learning of the ANN

It has been studied the forward computation that the neural network does. The inputs are propagated to calculate the values of nodes of the first hidden layer. Then, the ones of the next hidden layers (in case there are more than one) can be computed and so on until the output layer is obtained. Nevertheless, to obtain a good mapping between input and output, something else is needed. All the weights and the biases cannot be set at the desirable value by hand one by one. That is an impossible task. Hence, what is usually done is to initialize randomly the weights and biases and then they are continuously being adjusted by a process called **the training**.

If no training is performed to the network, it will just give random outputs. To achieve such training, data needs to fed to the network so that it has something to learn from. The **training data** consist of a series of inputs with their corresponding outputs or target values.

The cost function (also called the loss function) is defined. A function comparing the difference between output given by the network and the target value associated to the input data. This means that to train the network properly, the target or "desired" output value must be fed to the network so that it has something to learn from, to compare how it has performed, and if it does badly, recalculate the weights and biases to improve its performance [34]. For a given training data (combination of inputs with their corresponding targets), the cost function used is the mean

square error function (mse), and it is defined as, [43]

$$C(\boldsymbol{x}) = \frac{1}{2P}\sum_{p=1}^{P}\sum_{i=1}^{I} e_{p,i}^2 \tag{3.13}$$

where

$\boldsymbol{x}$ is the vector of weights and biases of the whole network

$$\boldsymbol{x} = \left[w_{1,1}^{(1)}, w_{1,2}^{(1)}, \ldots, w_{J,K}^{(1)}, b_1^{(1)}, \ldots, b_J^{(1)}, w_{1,1}^{(2)}, \ldots, b_1^{(2)}, \ldots, w_{j,k}^{(l)}, \ldots, b_j^{(l)}, \ldots, b_J^{(L)}\right]^T \tag{3.14}$$

$$\boldsymbol{x} = [x_1, x_2, x_3, x_q, \ldots x_Q]^T \qquad q = 1, 2, \ldots, Q \tag{3.15}$$

p is the index of training subsets, from 1 to P, where P is the number of training subsets (combination of one set of inputs with their corresponding targets). The training set is composed of numerous training subsets.

i is the index of outputs, from 1 to I, where I is the number of outputs in each training subset.

$e_{p,i}$ is the training error in output i applied in training subset p, and it is defined as

$$e_{p,i} = y_{p,i} - o_{p,i} \tag{3.16}$$

where y is the desired output or target value and o the actual output, $\boldsymbol{o} \equiv \boldsymbol{a}^{(L)}$. [43]

The mse function defined resulted as the sum of the square difference between output and target over all the training set and divided by the number of these training subsets. The training error for the given training subset can be expressed as a vector as $\boldsymbol{e}(\boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{o}$, where both $\boldsymbol{y}$ and $\boldsymbol{o}$ are column vectors with size equal to the number of outputs. Let us express the vector $\boldsymbol{E}(\boldsymbol{x})$, containing the training error of the whole training set [18],

$$\boldsymbol{E}(\boldsymbol{x}) = \left[\boldsymbol{e}_1^T(\boldsymbol{x})\ \boldsymbol{e}_2^T(\boldsymbol{x})\ \ldots \boldsymbol{e}_P^T(\boldsymbol{x})\right]^T \tag{3.17}$$

Notice that the vector $\boldsymbol{E}(\boldsymbol{x})$ is a column vector of size $P \cdot I \times 1$. Now, the cost function can be expressed with vectors instead of summation,

$$C(\boldsymbol{x}) = \frac{1}{2P}\boldsymbol{E}^T(\boldsymbol{x})\boldsymbol{E}(\boldsymbol{x}) \tag{3.18}$$

It can be inferred that the cost function becomes small when the output value is closer to the target one, as it is expected. Consequently, to improve the performance, the cost function needs to be minimized, which is the core idea of the learning process, finding suitable weights and biases to obtain a good mapping.

3.3.1 Gradient descent & Gauss-Newton

The learning algorithm used in the thesis is the *Levenberg-Marquardt* (LM) algorithm. This algorithm is a mid point between two algorithms, the Gradient Descent and the Gauss-Newton, which will be explained before introducing LM.

Gradient Descent

The gradient descent or steepest descent is a first order algorithm, since it uses the first-order derivative of the total error function for finding the local minimum of such function. The gradient is defined as the first-order derivative of the cost function [18],

$$\nabla C = \frac{\partial C(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{1}{P}\frac{\partial \boldsymbol{E}^T(\boldsymbol{x})}{\partial \boldsymbol{x}}\boldsymbol{E}(\boldsymbol{x}) = \frac{1}{P}\boldsymbol{J}^T\boldsymbol{E}(\boldsymbol{x}) \tag{3.19}$$

where the Jacobian matrix is introduced as

$$\boldsymbol{J} = \frac{\partial \boldsymbol{E}(\boldsymbol{x})}{\partial \boldsymbol{x}}$$
$$= \begin{bmatrix}
\frac{\partial e_{1,1}}{\partial w_{1,1}^{(1)}} & \frac{\partial e_{1,1}}{\partial w_{1,2}^{(1)}} & \cdots & \frac{\partial e_{1,1}}{\partial w_{J,K}^{(1)}} & \frac{\partial e_{1,1}}{\partial b_{1}^{(1)}} & \cdots & \frac{\partial e_{1,1}}{\partial b_{J}^{(1)}} & \cdots & \frac{\partial e_{1,1}}{\partial w_{j,k}^{(l)}} & \cdots & \frac{\partial e_{1,1}}{\partial b_{j}^{(l)}} & \cdots & \frac{\partial e_{1,1}}{\partial b_{J}^{(L)}} \\
\frac{\partial e_{1,2}}{\partial w_{1,1}^{(1)}} & \frac{\partial e_{1,2}}{\partial w_{1,2}^{(1)}} & \cdots & \frac{\partial e_{1,2}}{\partial w_{J,K}^{(1)}} & \frac{\partial e_{1,2}}{\partial b_{1}^{(1)}} & \cdots & \frac{\partial e_{1,2}}{\partial b_{J}^{(1)}} & \cdots & \frac{\partial e_{1,2}}{\partial w_{j,k}^{(l)}} & \cdots & \frac{\partial e_{1,2}}{\partial b_{j}^{(l)}} & \cdots & \frac{\partial e_{1,2}}{\partial b_{J}^{(L)}} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial e_{1,I}}{\partial w_{1,1}^{(1)}} & \frac{\partial e_{1,I}}{\partial w_{1,2}^{(1)}} & \cdots & \frac{\partial e_{1,I}}{\partial w_{J,K}^{(1)}} & \frac{\partial e_{1,I}}{\partial b_{1}^{(1)}} & \cdots & \frac{\partial e_{1,I}}{\partial b_{J}^{(1)}} & \cdots & \frac{\partial e_{1,I}}{\partial w_{j,k}^{(l)}} & \cdots & \frac{\partial e_{1,I}}{\partial b_{j}^{(l)}} & \cdots & \frac{\partial e_{1,I}}{\partial b_{J}^{(L)}} \\
\frac{\partial e_{2,1}}{\partial w_{1,1}^{(1)}} & \frac{\partial e_{2,1}}{\partial w_{1,2}^{(1)}} & \cdots & \frac{\partial e_{2,1}}{\partial w_{J,K}^{(1)}} & \frac{\partial e_{2,1}}{\partial b_{1}^{(1)}} & \cdots & \frac{\partial e_{2,1}}{\partial b_{J}^{(1)}} & \cdots & \frac{\partial e_{2,1}}{\partial w_{j,k}^{(l)}} & \cdots & \frac{\partial e_{2,1}}{\partial b_{j}^{(l)}} & \cdots & \frac{\partial e_{2,1}}{\partial b_{J}^{(L)}} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial e_{p,i}}{\partial w_{1,1}^{(1)}} & \frac{\partial e_{p,i}}{\partial w_{1,2}^{(1)}} & \cdots & \frac{\partial e_{p,i}}{\partial w_{J,K}^{(1)}} & \frac{\partial e_{p,i}}{\partial b_{1}^{(1)}} & \cdots & \frac{\partial e_{p,i}}{\partial b_{J}^{(1)}} & \cdots & \frac{\partial e_{P,2}}{\partial w_{j,k}^{(l)}} & \cdots & \frac{\partial e_{p,i}}{\partial b_{j}^{(l)}} & \cdots & \frac{\partial e_{p,i}}{\partial b_{J}^{(L)}} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial e_{P,I}}{\partial w_{1,1}^{(1)}} & \frac{\partial e_{P,I}}{\partial w_{1,2}^{(1)}} & \cdots & \frac{\partial e_{P,I}}{\partial w_{J,K}^{(1)}} & \frac{\partial e_{P,I}}{\partial b_{1}^{(1)}} & \cdots & \frac{\partial e_{P,I}}{\partial b_{J}^{(1)}} & \cdots & \frac{\partial e_{P,I}}{\partial w_{j,k}^{(l)}} & \cdots & \frac{\partial e_{P,I}}{\partial b_{j}^{(l)}} & \cdots & \frac{\partial e_{P,I}}{\partial b_{J}^{(L)}}
\end{bmatrix} \tag{3.20}$$

The gradient of the cost function will have column vector form with size equal to the total number of weights plus biases in the network. For the gradient descent, the increment rule in every weight and bias is written as, [18]

$$\boldsymbol{x}_{n+1} - \boldsymbol{x}_n = \Delta \boldsymbol{x} = -\alpha \nabla C \tag{3.21}$$

with n being the index of iterations and α is a constant named the learning rate or step size. By applying this rule repeatedly, the cost function C will decrease, improving the chances to arrive to a local minimum [34]. This method can be understood graphically as shown in Figure 3.9. Starting at some point of a given error function C, the gradient is calculated to find the steepest path to another point in the function, decreasing the error value. This process is repeated until a local minimum is reached. It is important to notice how depending on the initial value, the algorithm may converge to a local minimum, but not to the absolute minimum of the function. In neural networks, this problem cannot be thought graphically, since the cost function depends on many variables and the dimension of the problem is extremely high.

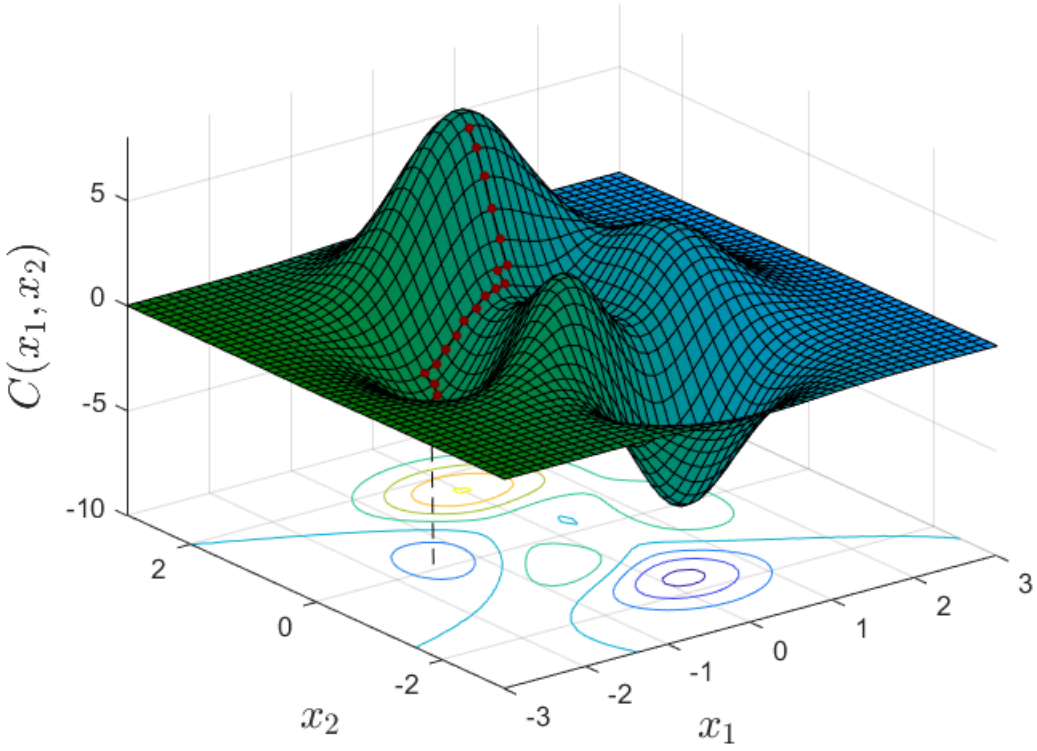

Figure 3.9: Sketch of an example of gradient descent algorithm

Gauss-Newton

This method uses second-order derivatives in order to update the weights and biases. That means calculating the Hessian matrix of the total error function. The increment $\Delta \boldsymbol{x}$ given by the Newton's method is given by, [18]

$$\Delta \boldsymbol{x} = -\frac{\nabla \boldsymbol{C}}{\nabla^2 \boldsymbol{C}} \tag{3.22}$$

$$\nabla^2 \boldsymbol{C} = \frac{1}{P}\frac{\partial \boldsymbol{E}^T(\boldsymbol{x})}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{E}(\boldsymbol{x})}{\partial \boldsymbol{x}}\boldsymbol{E}(\boldsymbol{x}) + \frac{1}{P}\boldsymbol{E}(\boldsymbol{x})\frac{\partial^2 \boldsymbol{E}^T(\boldsymbol{x})}{\partial \boldsymbol{x}^2} = \frac{1}{P}\left[\boldsymbol{J}^T\boldsymbol{J} + \boldsymbol{E}(\boldsymbol{x})\nabla^2\boldsymbol{E}(\boldsymbol{x})\right] \tag{3.23}$$

Calculating the the Hessian matrix, $\boldsymbol{H} \equiv \nabla^2\boldsymbol{C}$ can be very complicated, so the Gauss-Newton method assumes that $\boldsymbol{E}(\boldsymbol{x})\nabla^2\boldsymbol{E}(\boldsymbol{x}) \approx 0$ [18], so that,

$$\boldsymbol{H} \approx \frac{1}{P}\boldsymbol{J}^T\boldsymbol{J} \tag{3.24}$$

Here, only the Jacobian matrix needs to be calculated, which through the back-propagation method is much less complex than the Hessian matrix [9]. Therefore, the increment in each weight and bias can be computes as

$$\Delta\boldsymbol{x} = -\frac{\boldsymbol{J}^T\boldsymbol{E}(\boldsymbol{x})}{\boldsymbol{J}^T\boldsymbol{J}} = -\left[\boldsymbol{J}^T\boldsymbol{J}\right]^{-1}\boldsymbol{J}^T\boldsymbol{E}(\boldsymbol{x}) \tag{3.25}$$

Now, the Hessian is easy to calculate, since it is based on first order derivatives. However, when this formula is repeated iteratively to decrease the error function, it may result in large step sizes, which would revoke the main assumption on which this method is based [39]. This problem is interpreted mathematically as if $\boldsymbol{J}^T\boldsymbol{J}$ is not invertible [43]. This problem is solved by the LM algorithm.

3.3.2 Levenberg-Marquardt

The Levenberg-Marquardt algorithm is a modification of the Gauss-Newton. It introduces an extra term to approximate the Hessian matrix:

$$\boldsymbol{H} = \frac{1}{P}\left[\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}\right] \tag{3.26}$$

where $\boldsymbol{I}$ represents the identity matrix with same size as $\boldsymbol{J}^T\boldsymbol{J}$ and μ is called a combination coefficient and it is always positive [43]. Therefore, $\Delta\boldsymbol{x}$ is expressed as

$$\Delta\boldsymbol{x} = -\left[\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}\right]^{-1}\boldsymbol{J}^T\boldsymbol{E}(\boldsymbol{x}) \tag{3.27}$$

The LM algorithm was introduced as a mid-point between gradient descent and Gauss-Newton. This fact will depend on the value of μ. If the coefficient is really small, $\mu \approx 0$, Eq. 3.27 approximates Eq. 3.25. If μ is a very large value, $\mu\boldsymbol{I} >> \boldsymbol{J}^T\boldsymbol{J}$, Eq. 3.27 is approximated as Eq. 3.21, where $\alpha \approx 1/\mu$ [43]. If an step results in an increase in $\boldsymbol{E}(\boldsymbol{x})$, the coefficient μ is multiplied

by a factor β. And when a step reduces $\boldsymbol{E}(\boldsymbol{x})$, μ is divided by β, so that convergence is improved [18].

3.3.3 Back-propagation algorithm

From Eq. 3.27, it can be inferred that the key for updating the weights and biases of the network is to calculate the Jacobian matrix. The terms that need to be calculated are of the form:

$$\frac{\partial e_{p,i}}{\partial x_q} \quad \rightarrow \quad \frac{\partial e_{p,i}}{\partial w^l_{j,k}}, \ \frac{\partial e_{p,i}}{\partial b^l_j} \tag{3.28}$$

For this purpose the back-propagation algorithm is implemented. The idea behind the algorithm is to evaluate how sensible the error is to each weight and bias in the network and update them according to this evaluation. This is done by calculating the sensitivity of the error to the weights and bias connecting the last hidden layer to the output layer, and back-propagating this error up to the first connections. We begin calculating the gradient of a given error respect to a given weight or bias of the the vector $\boldsymbol{x}$,

$$\frac{\partial e_{p,i}}{\partial w^{(l)}_{j,k}} = \frac{\partial e_{p,i}}{\partial a^{(l)}_j} \frac{\partial a^{(l)}_j}{\partial z^{(l)}_j} \frac{\partial z^{(l)}_j}{\partial w^{(l)}_{j,k}} \tag{3.29}$$

$$\frac{\partial e_{p,i}}{\partial b^{(l)}_j} = \frac{\partial e_{p,i}}{\partial a^{(l)}_j} \frac{\partial a^{(l)}_j}{\partial z^{(l)}_j} \frac{\partial z^{(l)}_j}{\partial b^{(l)}_j} \tag{3.30}$$

The first two terms are equal for both gradients respect to weight and bias and represent the *sensitivity* of the error to the neuron j in layer l. It is defined as [15]

$$\delta^{(l)}_j \equiv \frac{\partial e_{p,i}}{\partial z^{(l)}_j} \tag{3.31}$$

From Eq. 3.11, $\frac{\partial z^{(l)}_j}{\partial w^{(l)}_{j,k}} = a^{(l-1)}_k$ and $\frac{\partial z^{(l)}_j}{\partial b^{(l)}_j} = 1$. Then, it can be shown that Eqs. 3.29 and 3.30 become:

$$\frac{\partial e_{p,i}}{\partial w^{(l)}_{j,k}} = \delta^{(l)}_j a^{(l-1)}_k \tag{3.32}$$

$$\frac{\partial e_{p,i}}{\partial b^{(l)}_j} = \delta^{(l)}_j \tag{3.33}$$

Moreover, according to [15] it can be also shown that the sensitivities follow the following recurrence relation:

$$\delta_k^{(l-1)} = F\left(z_k^{(l-1)}\right) w_{k,j}^{(l)} \delta_j^{(l)} \tag{3.34}$$

Equation (3.34) expresses the idea of the error back-propagation, the sensitivity of the $l-1$ layer is calculated according to the sensitivity of the next layer, where $\boldsymbol{F}$ is a matrix containing the derivatives of the activation function evaluated at each weighted sum (z_k) of the $l^{th}-1$ layer

$$\boldsymbol{F}\left(\boldsymbol{z}^{(l-1)}\right) = \begin{bmatrix} \sigma'\left(z_1^{(l-1)}\right) & 0 & \dots & 0 \\ 0 & \sigma'\left(z_2^{(l-1)}\right) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma'\left(z_k^{(l-1)}\right) \end{bmatrix} \tag{3.35}$$

and

$$\sigma'\left(z_k^{(l-1)}\right) = \frac{\partial a_k^{(l-1)}}{\partial z_k^{(l-1)}} = \frac{d\sigma\left(z_k^{(l-1)}\right)}{dz_k^{(l-1)}} \tag{3.36}$$

The terms inside this matrix actually corresponds to the second term of the chain rule in Eqs. 3.29 and 3.30, $\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$, but applied to the $l^{th}-1$ layer instead of the l^{th} layer. The first term of those two equations, $\frac{\partial e_{p,i}}{\partial a_j^{(l)}}$, is related to Eq. 3.16, where $o_j = a_j^{(L)} = w_{j,k}^{(L)} a_k^{(L-1)} + b_j^{(L)}$. Since the difference between target and output given by the network is known, the recurrence of Eq. 3.34 is initialized in the output layer.

$$\delta_j^{(L)} = \frac{\partial e_{p,i}}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = -F\left(o_j\right) \tag{3.37}$$

Now, Eq. 3.37 is introduced in Eq. 3.34 and the sensitivity in the previous layer is calculated back-propagating the error using the corresponding weight in the L^{th} layer. Once the sensitivity is known, using Eqs. 3.32 and 3.33, their corresponding value in the Jacobian matrix can be obtained. This process is repeated until $l = 1$, calculating the sensitivity related to each weight and bias and using Eqs. 3.31 and 3.32, the whole Jacobian matrix is computed.

3.3.4 Summary of the learning algorithm

The complete learning algorithm can be summarize in the following steps, expressing every variable as matrix form:

1. **Present the input to the neural network and propagate** *the values until the output layer is known according to*

$$\boldsymbol{a}^{(l)} = \sigma\left(\boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}\right)$$

2. **Compute the sum square error over all inputs**

$$C(\boldsymbol{x}) = \frac{1}{2P}\sum_{p=1}^{P} \boldsymbol{e}_p^T \boldsymbol{e}_p$$

3. **Calculate each value of the Jacobian matrix** *computing the following equations in order*

$$\boldsymbol{\delta}^{(L)} = -\boldsymbol{F}\left(\boldsymbol{o}\right)$$
$$\boldsymbol{\delta}^{(l-1)} = \boldsymbol{F}\left(\boldsymbol{z}^{(l-1)}\right)\left(\boldsymbol{W}^{(l)}\right)^T \boldsymbol{\delta}^{(l)}$$
$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\delta}^{(l)}\boldsymbol{a}^{(l-1)}$$
$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$$

4. *Once the Jacobian matrix is found,* **obtain the increment in every weight and bias,** $\Delta\boldsymbol{x}$*, of the network*

$$\Delta\boldsymbol{x} = -\left[\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}\right]^{-1}\boldsymbol{J}^T\boldsymbol{E}(\boldsymbol{x})$$

5. **Recompute the sum square error** *using* $\boldsymbol{x} + \Delta\boldsymbol{x}$*. If the total error is smaller than the previous one, divide* μ *by* β*, let* $\boldsymbol{x} = \boldsymbol{x} + \Delta\boldsymbol{x}$ *and go back to step number 1. If the total error is bigger than the preceding one, multiply* μ *by* β *and repeat step number 3.*

6. **The algorithm convergence** *has finished when the norm of the gradient,* $\nabla C = frac1P|$ *boldsymbol*$J^T\boldsymbol{E}(\boldsymbol{x})|$*, is less than a user defined value or when the sum square error is reduced up to a user determined goal.*

Chapter 4

Simulation, Results and Discussion

In this chapter, the simulations using ANN and the results are presented. Part of the work consist on the construction of the network architecture and parameters setting. The network structure will be chosen according to a performance analysis in order to assure good results. But first a background on the software is introduced as well as the pre-simulation set-up needed to prepare the network. Once the network structure and components are set up, the data is needed. Data needs to be fed to the network with certain conditions that will be evaluated and analyzed in the data pre-processing section. Finally, the network simulations on both theories will be performed. The training performance and the results accuracy will be discussed. For each LP theory, two approaches are carried out: A forward simulation, where the goal is to obtain the IV curve given the plasma parameters. This method will serve as a tool to establish the network capacities and guide the problem to the interesting one, the inverse mapping. The underlying purpose of the thesis is to evaluate the performance of ANN in this particular problem, where the plasma parameters have to be obtained from the current law. The inverse mapping will be later performed with experimental data and its implementation will be analyzed.

4.1 MATLAB Toolbox

The neural network is created using the mathematical software *MATLAB*. This software offers a simpler way to study ANNs than other software. MATLAB offers two different options to use artificial neural network:

1. Predefined networks with the architecture already created, which are ready to be used. Different type of networks (depending on the problem) are available where only the input data is needed to be given to the toolbox. All the different parameters used, which will be explained later on, are the standard ones and only the number of neurons and hidden layer can be changed.

2. Different functions associated to the neural network can be used to create a customized network. All the parameters can be set and changed to optimize the network. The main three function associated to this options are:

   - *feedforwardnet*: This function initializes and creates the network architecture. The number of layers and the number of neurons in each of them can be set as desired, as well as the initial value of the weights and bias. Note, that this function is used specifically to create feed-forward network, which is the type that will be used in this thesis.
   - *train*: This function reproduces the learning algorithm explained in Section 3.3. The complete set of input data is fed to the network and the back-propagation algorithm is performed to update the weights and biases to reduce the cost function.
   - *sim*: This function is implemented once the learning has been completed. With the weights and biases optimized, the network can be simulated for a given input using this function.

Both options were tried and **option number 2** was selected. Option number 1 is based on straight forward problem where little pre-processing and post-processing is needed, since it lacks of maneuverability and controllability. Thus, option number 2 is used to create the neural network, so that it is possible to change the network parameters to adapt it to the problem presented and be able to control every step of the process.

The first step when training the network is to feed the data containing each training subset. According to the index notation of the ANN, the input data includes the inputs with size $K^{(0)} \times P$ and target data $I \times P$. It must be noticed that the number of columns should be always the same, meaning that each training subset has the same size, P, which represents the number of training subsets and it is divided by MATLAB into three categories.

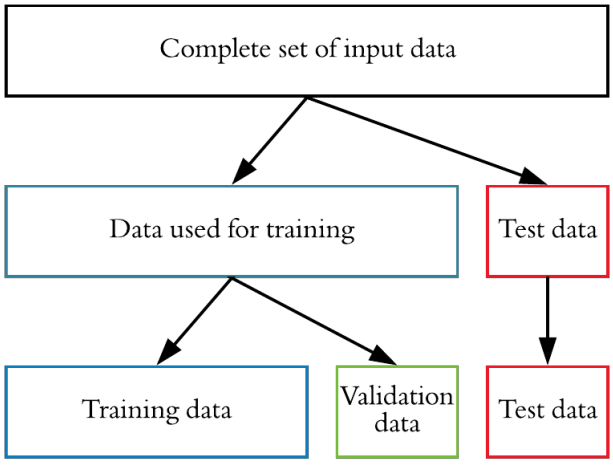

Figure 4.1: Division of the input data into different categories

From the complete input data given by the user, the division is made randomly by MATLAB, regrouping the data into training, validation and testing set. They all have different purposes and the standard percentage for each division is represented in Figure 4.2, where the whole pie represents the 100% of the input data.

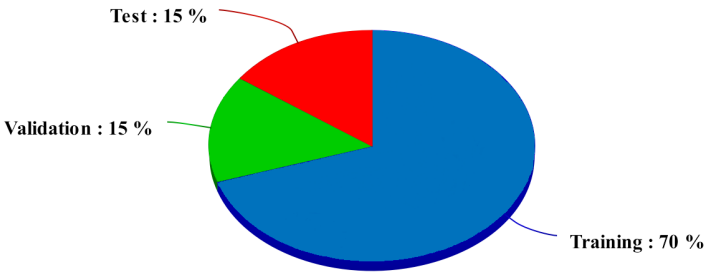

Figure 4.2: Standard percentages of training, validation and test set respect to the complete set of input data

Training set: This data is used exclusively to update the weights and biases. Once the training set is fed, the cost function is computed and using the back-propagation algorithm, $\Delta \boldsymbol{x}$ is updated repeatedly.

Validation set: They are used as an unbiased evaluation for the network performance. Using the updated weights and biases, the cost function associated to the validation set is calculated. If after an iteration, the cost function in the training set has decreased but the cost function of the validation data has increased, it means that the network is adjusting to much to the training data, losing its ability of generalization. This problem is known as over-training, and results in over-fitting issues. Validation sets are used to stop the training when possibility of over-fitting is found. That means that during six consecutive iterations $C(\boldsymbol{x})_{training}$ had been decreasing, but $C(\boldsymbol{x})_{validation}$ had been increasing.

Testing set: These subsets are used only to assess the performance of the network of an unbiased input data.

To understand better the concept of validation and why it is of great importance, the following problem is consider: The function $y^\star = x^{\star 2} + x^{\star 3}$ wants to be represented in the interval $[-2, 2]$ using a neural network with one hidden layer with 10 neurons. The network will be trained using values of $x^\star$ with the corresponding $y^\star$ plus some added noise, $\Delta y^\star$. Once trained, it will be simulated with $x^\star$ and the results will be compared with the theoretical $y^\star$ plus $\Delta y^\star$. The two plot of $y^\star$, with and without noise are represented in Figure 4.3.

Figure 4.4 shows the difference in results when the division into validation data has been performed (left) and when it has been not (right). It can be observed how when validation division did not take place, the output approximates the function $y^\star$ with noise and when validation division was performed, it approximates to the theoretical function. This is the over-fitting problem described, the network tries to replicate the data given, without any consideration whether noise might be added or not, losing one of the powerful advantages of ANN, the generalization. If the problem

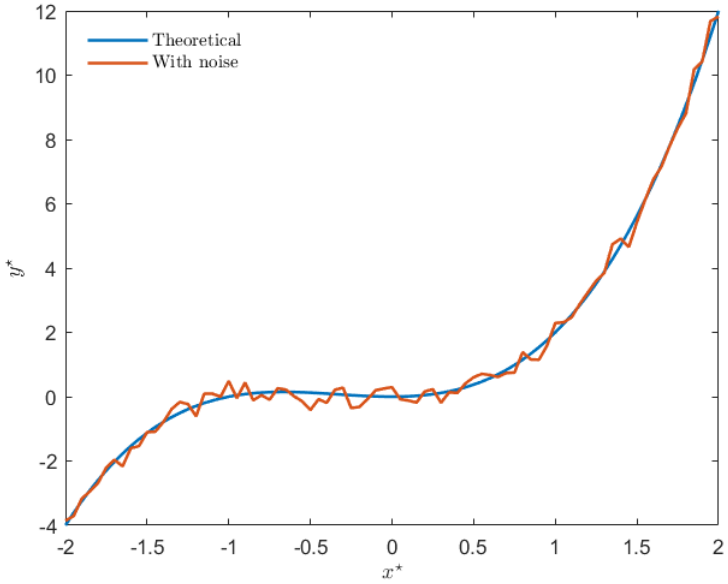

Figure 4.3: Function $y^\star = x^{\star 2} + x^{\star 3}$ with and without noise

is to reproduced exactly the input data, the problem of over-fitting does not exist, but at the minimum sign of eluding from this data, the network will no perform well.

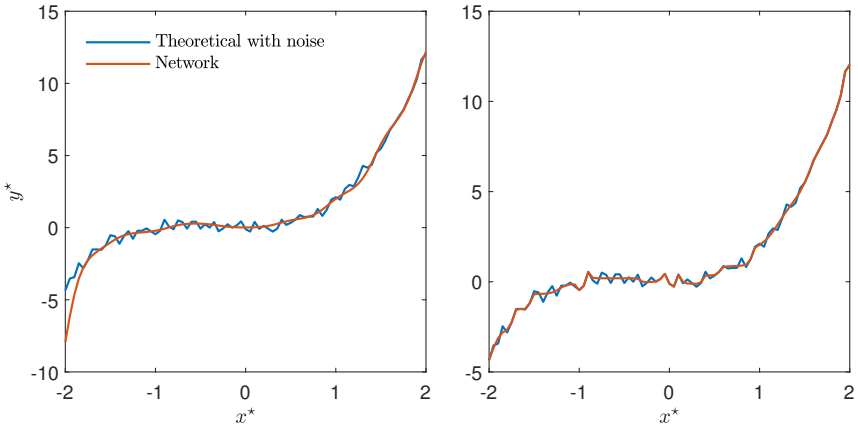

Figure 4.4: Comparison between output of a network trained with (left) and without (right) validation set

MATLAB has the different network parameters initially set as the standard ones, but sometimes it is helpful to change them in order to optimize the performance of the specific problem, the main parameters are:

- *Architecture*: It includes the number of layers and neurons in each of them. The size will depend on the complexity of the problem and should be optimized by the user. The standard configuration is 1 hidden layer with 10 neurons, but it will be changed to improve the performance.
- *Training algorithm*: The training algorithm is referred to the mathematical technique used to update the weight and biases. The back-propagation Levenberg-Marquardt algorithm explained in Sections 3.3.2-3.3.4 is used.
- *Performance function*: The performance function corresponds to the cost function used. Different ones exist, such as "sum square error" or "cross entropy cost function", but the mean square error defined in Eq. 3.13 is used.
- *Transfer function*: The transfer function (or activationcan be different for every layer. It will depend on the problem to be solved. Regression problems like the one encountered use two different transfer functions: sigmoid or hyperbolic tangent function for the hidden layers because of their ability to give a wide range of bounded values, and a linear function in the output layer so that final output is not bounded.
- *Epochs*: The epochs are equivalent to an iteration. A epoch is constituted by the propagation of the input data to obtain the output, compute the cost function and the back-propagation to update the weights and biases. They can be set to any value desired, taking into considerations that when the epochs are increased so will the time required to train the network.
- *Goal*: It is a parameter used by the training algorithm to establish a stopping criteria in the training. The goal is the minimum value of the cost function, $C(\boldsymbol{x})_{training,min}$, which is accepted by the user as a good enough performance.
- *Minimum Gradient*: Same objective as the goal parameter, ∇C_{min} is the minimum value of the gradient below which the network stops training.
- *β factor*: Parameter by which μ is divided or multiplied depending on the current performance respect to the proceeding. β can be a different number when it multiplies and when it divides. The standard values are: $\beta_{decrease} = 0.1$, $\beta_{increase} = 10$.
- *Validation checks*: Stopping criteria to avoid over-fitting. If after an epoch, the training cost function has decreased, but the validation cost function has increased, both respect their previous value, a validation check rises. If the validation check rises a number of consecutive times, the training stops. The standard number of validation checks is 6.

4.2 Data pre-processing

The neural network begins with the input data and ends with it. This statement underlies the importance of the data fed to the network, the network is going to learn and improve from the input data and at the end of the day, it will compare the results with the target data given. It is clear that the way data is managed will play a key role in the training process, ergo in the network performance. The goal of data pre-processing is to feed the network the best way possible to improve its accuracy.

One of the most important steps of data pre-processing is the scaling of data. Scaling the data will have a considerably beneficial improvement if it is performed within the boundaries of the activation function. Especially, if the data managed has several order of magnitude. This statement can be understood with a simple example:

One of the plasma parameters is plasma density, this property usually can have values ranging from $10^{15}m^{-3}$ *to* $10^{17}m^{-3}$*. How does this high order of magnitude affect the problem? It affects greatly when the activation functions are applied. For the hyperbolic tangent function, really high or low values of the weighted sum* (z_j) *approximate 1 or -1 respectively, meaning that big changes in* z_j *result in very little deviation of* $a_j^{(l)}$*. Below it is observed how although the value of z is doubled, the result is practically the same. This is obviously an issue, since the network will find really complicated to differentiate the importance of each input and assign a proper weight to it.*

$$a_j^{(l)} = \tanh(z_j) = \tanh(7) = 0.9999983369$$
$$a_j^{(l)} = \tanh(z_j) = \tanh(14) = 0.9999999999$$

If the data managed exceeds the activation function boundaries, scaling should be performed. Moreover, the problem evaluated has several parameters each one with different order of magnitude, which makes it impossible to the network to perform well if the data is simple thrown out to the network.

Therefore, each variable needs to be delimited by the same values. In order to do so, the following procedure is done: Considering X_n an input parameter, $X_{n,min}$ will be the minimum value given to this variable in the whole data set and $X_{n,max}$ the largest one. Then, each variable is corrected linearly with its maximum and minimum value.

$$L' = m \cdot X_{n,min} + b \tag{4.1}$$

$$U' = m \cdot X_{n,max} + b \tag{4.2}$$

Where L' is the lower boundary of the normalized variable and U' the upper. Solving these two equations the new normalized values for each variable will be,

$$X'_n = \frac{(U' - L')\, X_n - U' X_{n,min} + L' X_{n,max}}{X_{n,max} - X_{n,min}} \tag{4.3}$$

where X_n is the value of the of the parameter, X'_n is the new scaled value and, U' and L' are 1 and -1, respectively (hyperbolic tangent function limits). This procedure is performed to all of the variables, inputs and target. The network should be fed with the scaled values and trained with them, producing scaled results too. To obtain the real magnitude of the parameter, the opposite procedure is done, obtaining X_n given X'_n,

$$X_n = \frac{(X_{n,max} - X_{n,min})\, X'_n + U' X_{n,min} - L' X_{n,max}}{(U' - L')} \tag{4.4}$$

Another key aspect to improve the network performance is finding a good set of input data with the corresponding target data. That means obtaining a set of inputs which hold some kind of relation with the outputs, these inputs will be called *morphological parameters* [16]. The relation between these parameters and the output can be of any kind, but they must represent accordingly the relation input-target.

4.3 Planar LP theory ANN simulation

The artificial neural network simulation will be performed for the planar LP theory. To solve mapping and function approximation problems, the feed-forward networks are commonly used [12]. For the development of this model, a constant ion temperature will be assumed, $T_i = 1eV$. The fixed network parameters will be:

1. Training algorithm: Levenberg-Marquardt with mean square error performance function.
2. Transfer function: Hyperbolic tangent for hidden layer and linear for output layer (typical feed-forward architecture [9])
3. Maximum epochs: 1000
4. Goal: 1e-09
5. Minimum gradient: 1e-07

4.3.1 Forward simulation

The forward simulation is useful to obtain an initial guess for complicated numerical models and to understand the network behavior with the data presented. The equations presented on the Langmuir probe section form a simplified model to calculate the current measured as a function of the bias and plasma parameters. Then, the problem encountered is the following: $I_t = f(T_e, n_\infty, r_p, \Delta V)$. The forward simulation will consist of calculating the corresponding

I-V curve to a set of given plasma parameters using the neural network. That means that the simulation input should have the form shown in Table 4.1.

$4 \times l_{\Delta V}$				
T_e	T_e	T_e	...	T_e
r_p	r_p	r_p	...	r_p
n_e	n_e	n_e	...	n_e
ΔV_1	ΔV_2	ΔV_3	...	ΔV_{end}

Table 4.1: Representation of the simulation input data of the forward planar problem

The morphological parameters are four: T_e, n_∞, r_p and ΔV and the output associated to each column is just one value of I_t, obtaining the complete I-V curve associated to that ΔV. To train the network an input matrix $4 \times P$ is formed and a target matrix $1 \times P$, where P will be given by,

$$P = l_{T_e} \times l_{n_e} \times l_{r_p} \times l_{\Delta V} \tag{4.5}$$

where $l_{T_e} = 20$ is the number of values of electron temperature in the range of $[1 - 10]\, eV$, $l_{n_\infty} = 20$ is the number of values of plasma density in the range of $\left[10^{15} - 10^{17}\right]\, m^{-3}$, $l_{r_p} = 20$ is the number of values of probe radius in the range of $[0.001, 0.01]\, m$, and $l_{\Delta V} = 35$ is the number of bias voltage minus plasma potential in the range of $[-50, 20]\, V$.

The input $4 \times P$ matrix is created by multiple matrices like Table 4.1. Each of them with different values of T_e, n_∞ and r_p. At the end of the day, each column of the P subsets will have a different combination of the 4 inputs. The target matrix will be the current calculated for each subset combination (one column of the input matrix). The data given to train the network will look like Table 4.2.

	P							
	$4 \times l_{\Delta V}$				$4 \times l_{\Delta V}$			
$T_{e,1}$	$T_{e,1}$	...	$T_{e,1}$	$T_{e,2}$	...	$T_{e,2}$	...	$T_{e,l_{T_e}}$
$r_{p,1}$	$r_{p,1}$	...	$r_{p,1}$	$r_{p,1}$	...	$r_{p,1}$	...	$r_{p,l_{r_p}}$
$n_{e,1}$	$n_{e,1}$	...	$n_{e,1}$	$n_{e,1}$	...	$n_{e,1}$	...	$n_{e,l_{n_e}}$
ΔV_1	ΔV_2	...	ΔV_{end}	ΔV_1	...	ΔV_{end}	...	ΔV_{end}
$I_{p,1}$	$I_{p,2}$	...	$I_{p,l_{\Delta V}}$	$I_{p,l_{\Delta V}+1}$	...	$I_{p,2 \cdot l_{\Delta V}}$	...	$I_{p,P}$

Table 4.2: Representation of training data structure containing the input data and the target data for the forward planar simulation

The network was trained with different structures (number of neurons and layers) to analyze which gives the best performance. Five configurations were tried: 1) 1 hidden layer with 10 neurons, 2)

1 hidden layer with 20 neurons, 3) 1 hidden layer with 30 neurons, 4) 2 hidden layers with 10 and 5 neurons respectively (10/5), and 5) 2 hidden layers with 10 neurons each (10/10) .

Network structure (neurons layer 1/neurons layer 2)	10	20	30	10/5	10/10
R	0.99743	0.99965	0.99991	0.99999	1
b (±)	4.6e-03	5.9e-04	1.6e-04	5.5e-06	2.3e-06

Table 4.3: Network training performance parameters (R and bias) describing the relation between target output and network output

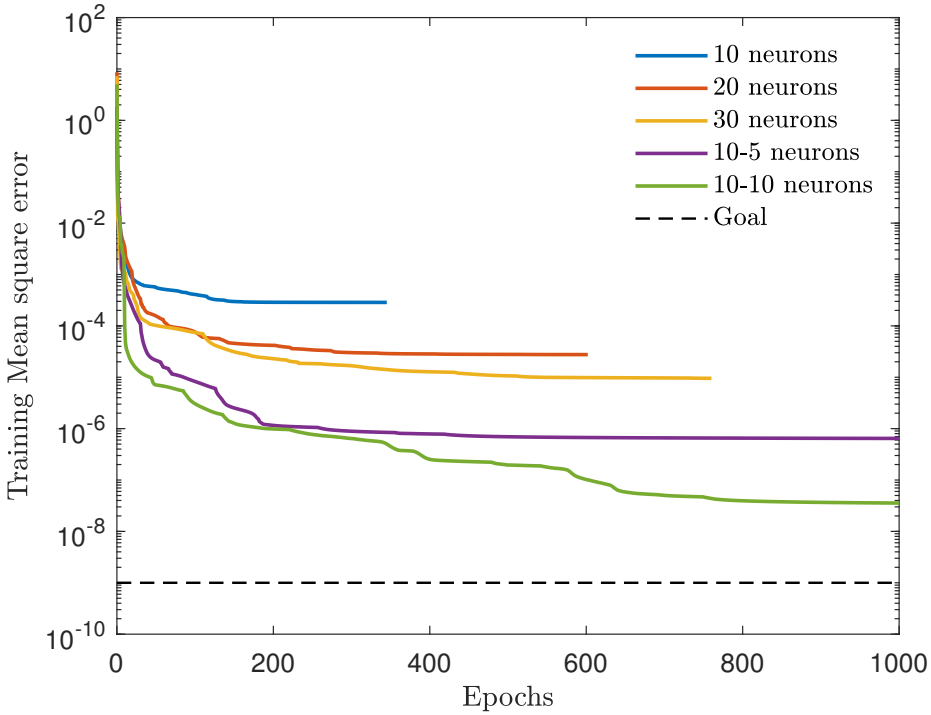

Figure 4.5: Training performance convergence for different types of network structures for the forward planar case

Table 4.3 contains two training performance parameters, the coefficient of correlation (R) between target and network output and the bias (b), used to build an empirical relations between them.

$$\text{outputs} \approx \text{targets} + b \qquad \rightarrow \qquad o \approx y + b \tag{4.6}$$

The coefficient of correlation is a metric used to measure the linear relatedness between output and target. When the value of R is closer to 1, it means that the network has find a set of weights and biases which gives a perfect, or almost perfect, linear relation between the target and the network output [29]. This linear relation can be also expressed as a linear function shown in Eq. 4.6, where the bias represents the approximate uncertainty made.

Two facts should be in mind. These parameters are a mean representation for all the training subsets, meaning that for some subsets the performance might be better and for other worse, but the mean value gives an overall idea of the linear relation. Moreover, it must be noted that these training performance parameters are a representation of the accuracy state of the network for the scaled dimensionless data, not to the real values of total current, since the network is trained with scaled data.

Figure 4.5 shows the convergence of the cost function (mse) with the number of epochs. Structures containing only one hidden layer fail to reach the goal performance and the maximum number of epochs, stopping the training due to reaching the maximum number of validation checks. For a single hidden layer, if the number of neurons is increased, the performance will improve, but at a slow rate as it is inferred from the performance difference between 1), 2) and 3). To improve the performance, one more layer is added, showing clearly its beneficial effect. Each layer created adds its own level of non-linearity that cannot be contained in a single layer, obtaining better results when highly non-linear problems are encountered.

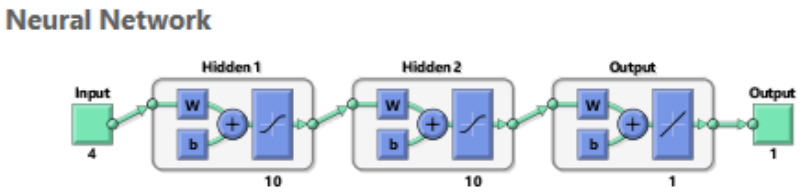

Figure 4.6: Sketch representation of the network structure with two hidden layers with 10 neurons each, a linear output and particularized for the forward LP planar problem (4 inputs)

From the performance parameters and the mean square error, the best solution is a two hidden layer network with 10 neurons each. The number of layers and neurons could be increased as much as the programmer desires, but for the problem to be solved, this structure is good enough and of relatively short training duration. It will be the structure used from now on in every case solved (Figure 4.6).

For the structure chosen, other training characteristics can be analyzed to have an overall view of the training process. In Figures 4.7, 4.8 and 4.9, the gradient of the cost function, the μ parameter and validation checks are plotted against the epochs. From Figure 4.7, the gradient decreases with the number of iterations as the cost function is finding its way to a local minimum. It is compared with the gradient of the structure of 30 neurons, which is observed to be larger, meaning that the cost function is further from reaching the minimum. The local minimum is assumed to be reached when the gradient fails below the minimum gradient.

The μ parameter is directly associated to the gradient. As the gradient becomes smaller, local minimum is closer and μ decreases to not overshoot the local minimum. Then, the μ parameter represents the error convergence, bigger jumps in μ may result in faster convergence, but they can also affect negatively, by overshooting the minimum and never reaching convergence. If the gradient increases, μ will increase to converge to the minimum faster. Figure 4.9 shows that stop by validation did not happen. Although some validation checks occurred, there was not six

consecutive checks, stopping the training by reaching the maximum number of epochs. This was not the case of 30 neurons where the training stopped because of 6 consecutive validation checks (during 6 consecutive epochs) near the 800 epoch.

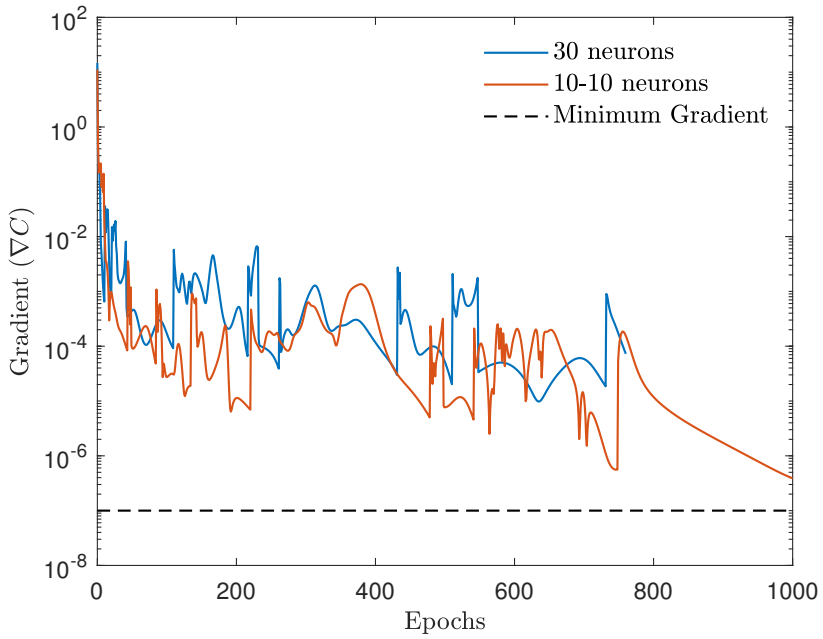

Figure 4.7: Cost function gradient at each epoch during training the forward planar LP case

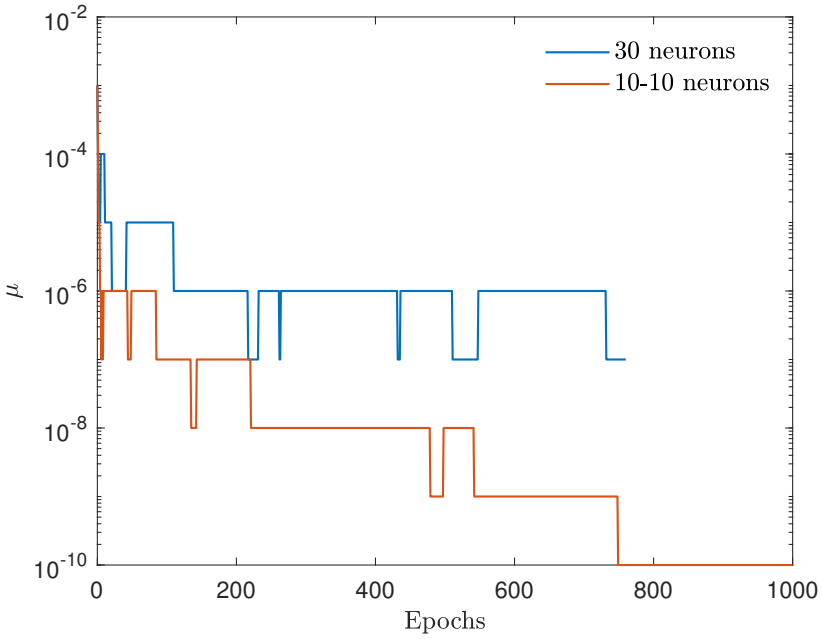

Figure 4.8: μ parameter at each epoch during training the forward planar LP case

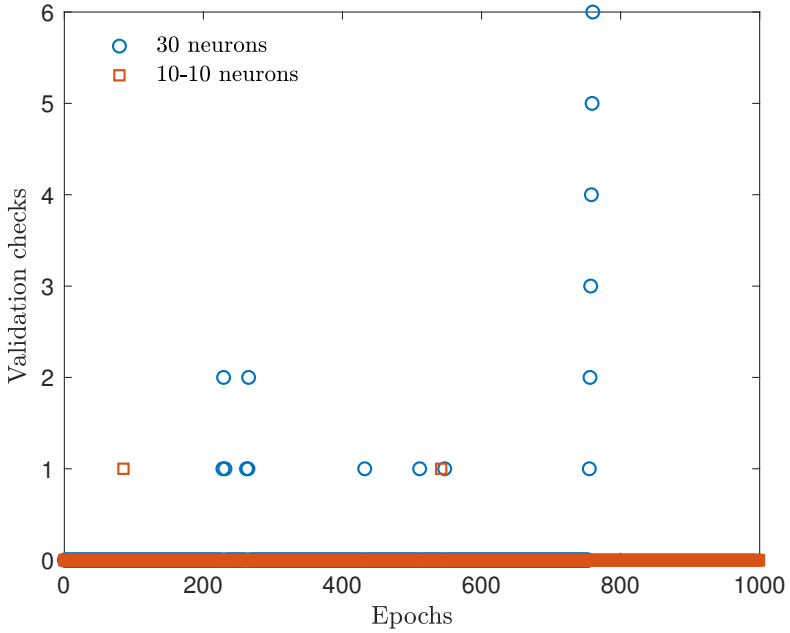

Figure 4.9: Validation checks at each epoch during training the forward planar LP case

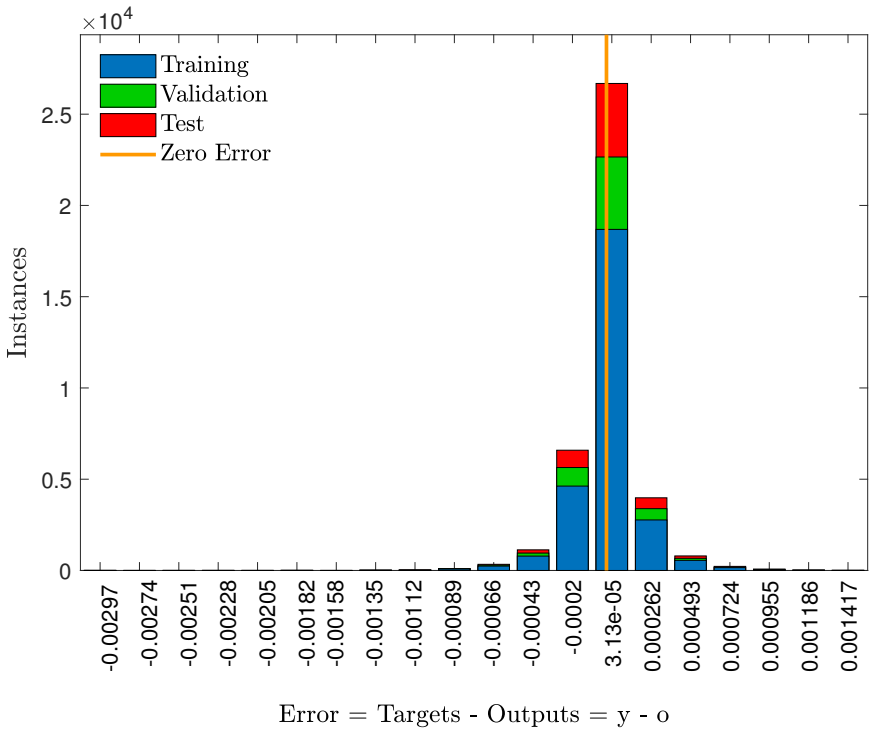

Figure 4.10: Histogram illustration of the error between target and output for every subset trained for the forward planar LP case

Figure 4.10 represents an overall view of the post-training performance. An histogram containing the post-training error made in each of the subsets. The input data is divided into three sets (training, validation and test). The x axis is the error made, calculated as the target output minus network output. The y axis corresponds to the instances or number of subsets associated to these errors. The orange zero error line, would be the ideal output, where the output is equal to the target. Two features of this plot are important. The error of the outer bars of the plot, they represent the maximum error (above and below target) that has been made within the subsets provided. The closer these outer values are to 0, the better performance the network will have. The other feature is the concentration of the errors. Ideally, all the subsets should be concentrated in the zero line. Therefore, subsets around the zero error line is an indicator of good performance. From the figure, it can be expected quite good performance when the simulations are performed. Note that the error is given respect to the normalized data, not actual values of plasma parameters.

In order to test the network, an input like Table 4.1 is fed to the network with values T_e, r_p and n_e, chosen randomly and different from the ones used in the grid of Table 4.2, and a ΔV vector. In Table 4.1 each column of this input matrix will be forwardly propagated to obtain a single value of current, obtaining a current vector with $l_{\Delta V}$ values. The network was simulated for 4 different cases, all of them with different parameters except ΔV, which is a 40 value equally spaced vector from $-50V$ to $20V$. To compare the differences, the relative error in percentage is calculated using Eq. 4.7, where $\boldsymbol{o}$ is the output of the network and $\boldsymbol{y}$ the actual one.

$$|Error| = |1 - \frac{\boldsymbol{o}}{\boldsymbol{y}}| \tag{4.7}$$

	Simulation 1	Simulation 2	Simulation 3	Simulation 4
T_e (eV)	8	5	3	1.5
n_∞ $(1/m^3)$	10^{16}	$5 \cdot 10^{17}$	$2 \cdot 10^{15}$	$5 \cdot 10^{16}$
r_p (m)	0.005	0.003	0.008	0.0025

Table 4.4: Values used for each simulation in the forward LP planar case

From Figures 4.11 and 4.12 it can be observed that results are quite accurate. As the error is shown in percentage respect the target value, it does not reach 1% of error. In each of the four graphs a peek is observed. This peek occurs around floating potential. Around that point, current is almost zero, and when very small order of magnitudes are handled, small differences in those order of magnitudes will show a bigger deviation in the error percentage. Also, it can be said that the network has the most trouble to predict the electron saturation current, as observed in Simulations 1 and 4 of Figure 4.11. Although the error in percentage is smaller, the difference in values is the biggest.

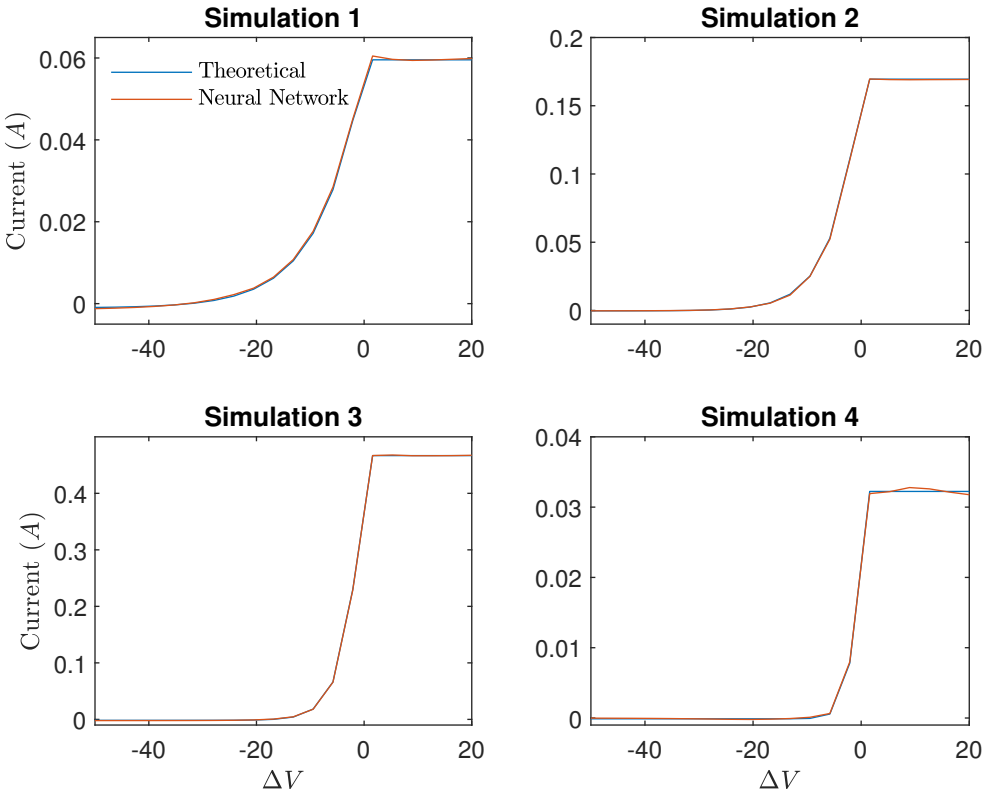

Figure 4.11: Comparison between I-V curves obtained theoretically and simulated with the neural network for the values of 4.3

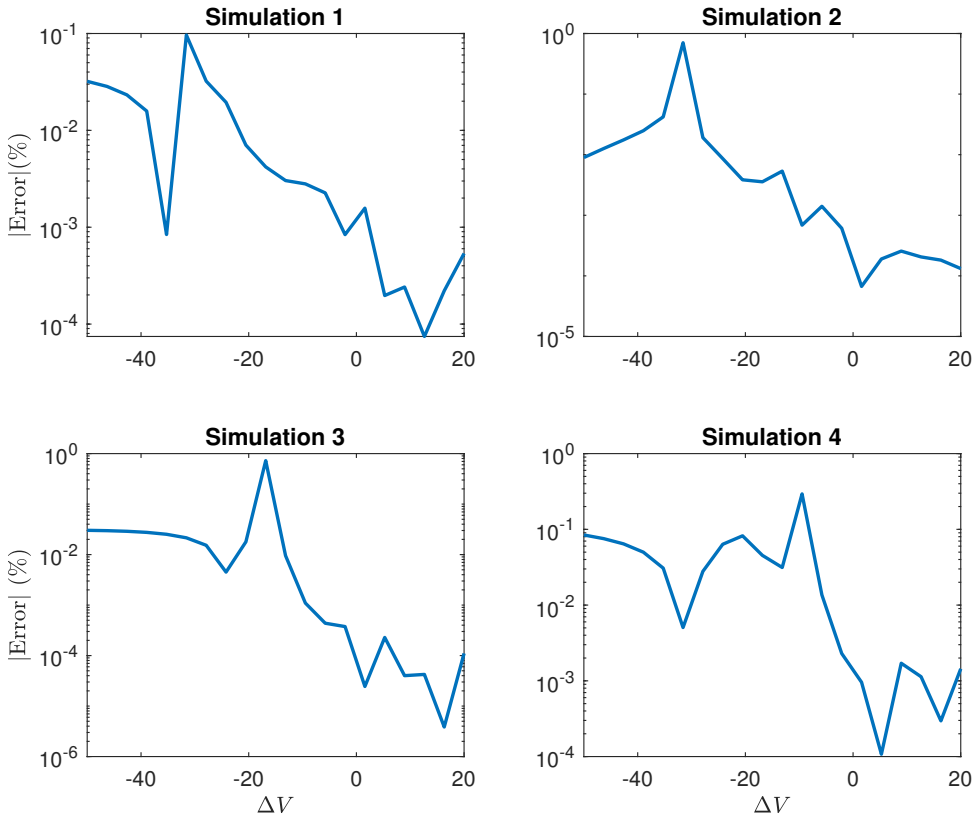

Figure 4.12: Relative error between I-V curves for each simulation performed of the forward planar LP case

4.3.2 Inverse mapping

The previous problem was performed to understand how the network works, the goal with artificial neural network is to obtain the plasma parameters given the current measure by the LP. Then, the problem complicates from the previous one. Analytically, there is one equation for the total current and two unknown plasma parameters (T_e and n_∞). The difficulty is to find a quantitative way to represent a curve by ANN inputs. Two approaches were found to work well for this problem

Discrete approach

The discrete approach consist of feeding directly the current values of the IV curve. In this way, the number of inputs will be larger than outputs. The drawback of this method is that it needs certain standardization of the current values chosen as inputs, since they all need to be associated to a certain ΔV and only currents associated to the ΔV values can be used. In vector form the input and output will be:

1. Input: The network is fed directly with the values of the current associated to the given value of ΔV and the selected probe radius, such that:

$$\boldsymbol{a}^{(0)} = [I_p(\Delta V_1), I_p(\Delta V_2), I_p(\Delta V_3), \ldots, I_p(\Delta V_{end}), r_p]^T \tag{4.8}$$

   where ΔV should be the same vector as the one defined for the training data. It should have the same number of values and the same values at each vector position. The key for the correct functioning is that ΔV remains the same through the entire training and simulation. The total number of inputs is $l_{\Delta V} + r_p$.

2. Outputs: The output is composed of two values, the calculated T_e and n_∞ by the network, such that:

$$\boldsymbol{o} = [T_e, n_\infty]^T \tag{4.9}$$

The neural network was trained using P training subsets, given by Eq. 4.10, where $l_{T_e} = 30$, $l_{n_\infty} = 30$ and $l_{r_p} = 30$. The ranges of the plasma parameters are the same as in the forward simulation.

$$P = l_{T_e} \times l_{n_\infty} \times l_{r_p} \tag{4.10}$$

The complete data set is built as shown in the Table 4.5. Each column represents the vector shown in Eq. 4.8 with the target values. The three parameters (T_e, n_∞, r_p) are varied to obtain the grid for training.

P

$I_p(\Delta V_1)$	$I_p(\Delta V_1)$	$\dots$	$I_p(\Delta V_1)$	$I_p(\Delta V_1)$	$\dots$	$I_p(\Delta V_1)$	$\dots$	$I_p(\Delta V_1)$
$I_p(\Delta V_2)$	$I_p(\Delta V_2)$	$\dots$	$I_p(\Delta V_2)$	$I_p(\Delta V_2)$	$\dots$	$I_p(\Delta V_2)$	$\dots$	$I_p(\Delta V_2)$
$I_p(\Delta V_3)$	$I_p(\Delta V_3)$	$\dots$	$I_p(\Delta V_3)$	$I_p(\Delta V_3)$	$\dots$	$I_p(\Delta V_3)$	$\dots$	$I_p(\Delta V_3)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
$I_p(\Delta V_{end})$	$I_p(\Delta V_{end})$	$\dots$	$I_p(\Delta V_{end})$	$I_p(\Delta V_{end})$	$\dots$	$I_p(\Delta V_{end})$	$\dots$	$I_p(\Delta V_{end})$
r_{p_1}	r_{p_1}		r_{p_1}	r_{p_2}	$\dots$	r_{p_2}	$\dots$	$r_{p,l_{r_p}}$

$T_{e,1}$	$T_{e,2}$	$\dots$	$T_{e,l_{T_e}}$	$T_{e,1}$	$\dots$	$T_{e,l_{T_e}}$	$\dots$	$T_{e,l_{T_e}}$
$n_{e,1}$	$n_{e,1}$	$\dots$	$n_{e,1}$	$n_{e,1}$	$\dots$	$n_{e,1}$	$\dots$	$n_{e,l_{n_e}}$

Table 4.5: Training data structure (inputs with the corresponding targets) for the inverse planar problem

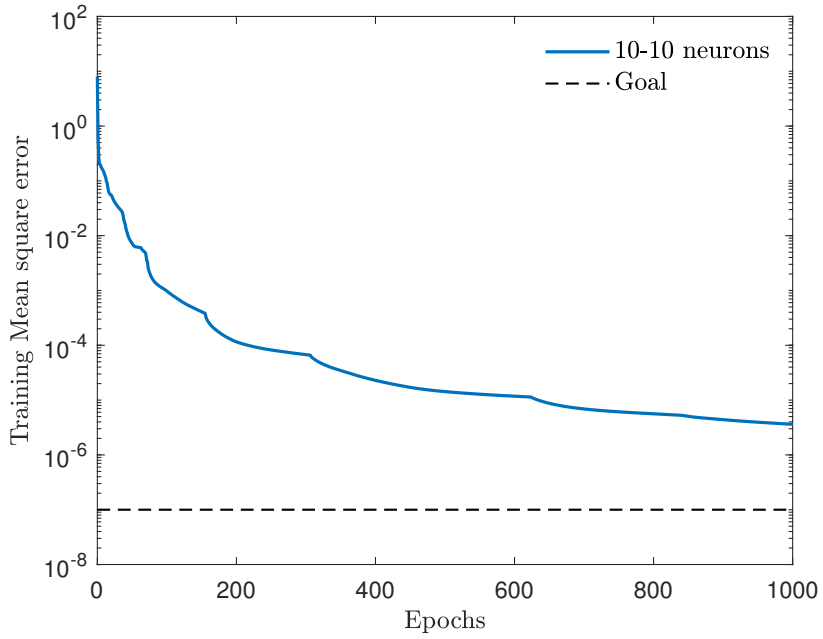

Figure 4.13: Cost function convergence at each epoch during training the inverse planar LP case

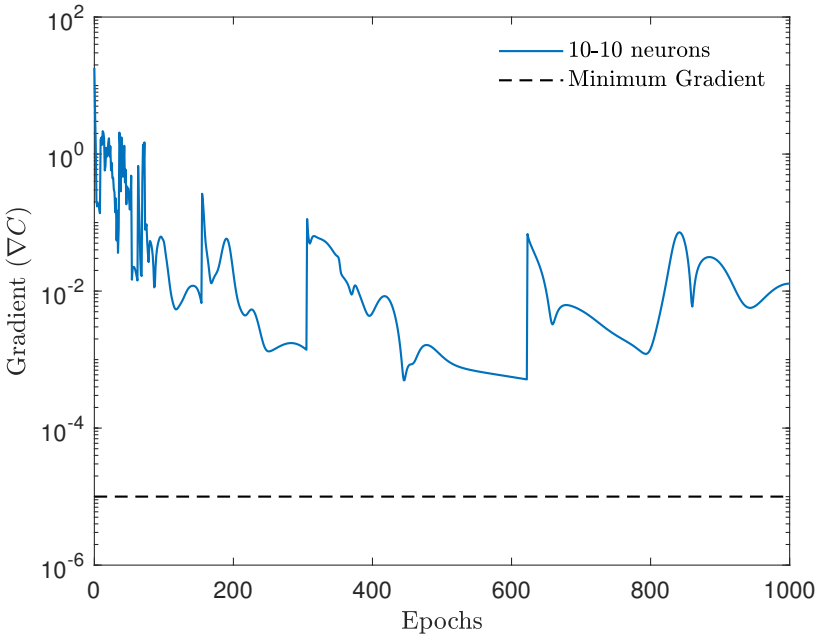

Figure 4.14: Gradient each epoch during training the inverse planar LP case

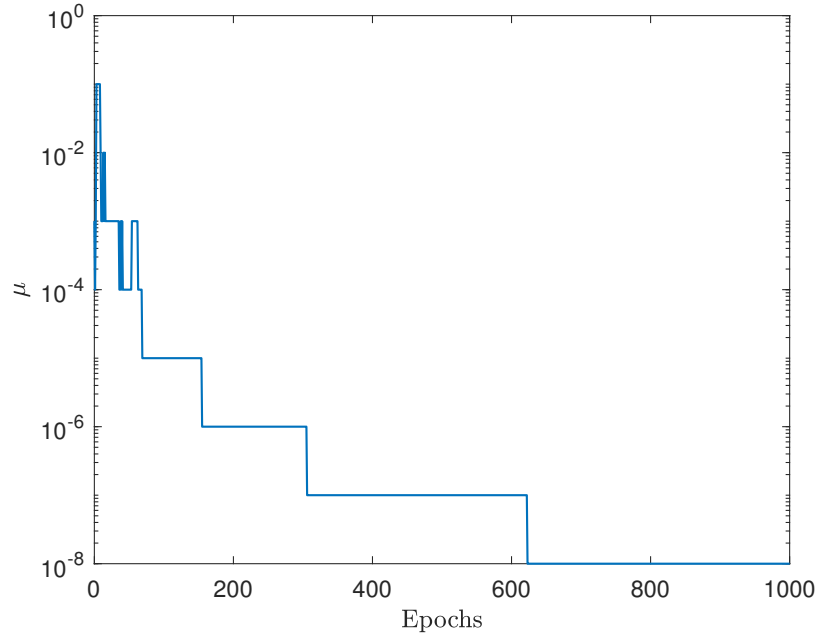

Figure 4.15: μ parameter at each epoch during training the inverse planar LP case

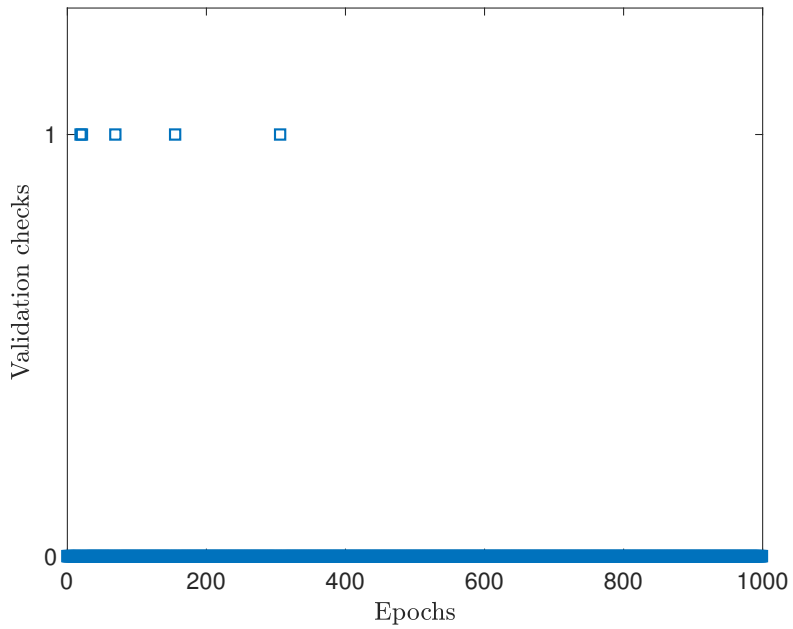

Figure 4.16: Validation checkss at each epoch during training the forward planar LP case

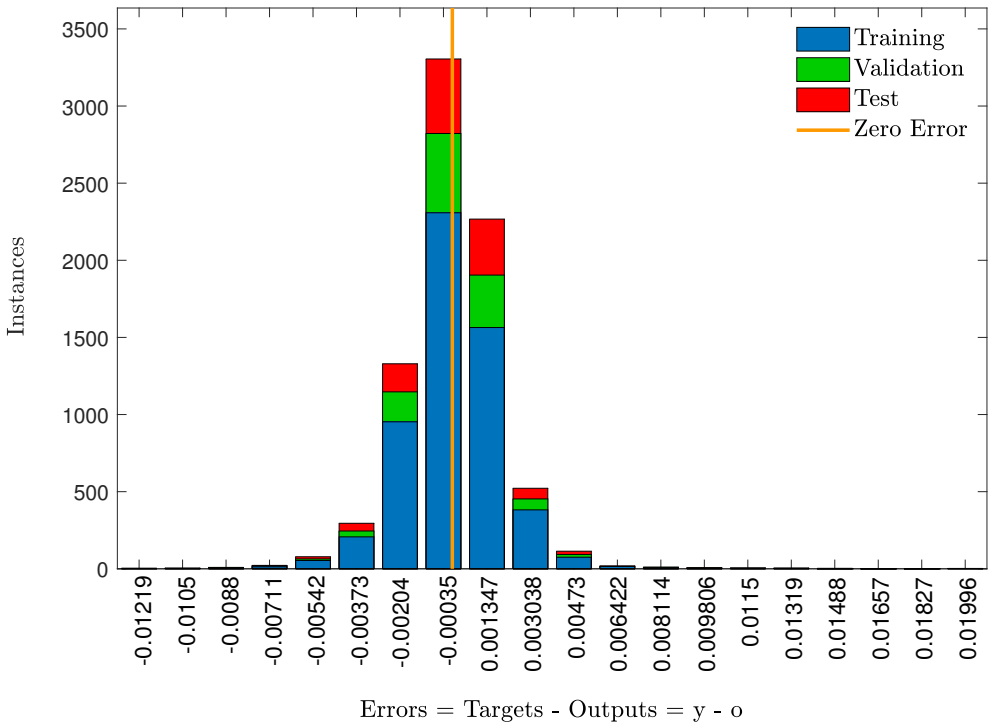

Figure 4.17: Histogram illustration of the error between target and output for every subset trained for the inverse planar LP case

Post-training results anticipate good performance. Training stopped due to reaching the maximum number of epochs. The cost function does not reach the goal, but still small enough to obtain good results, this can be observed better in Figure 4.17. The maximum errors made on the scaled training data are $[-0.01219, +0.01996]$ (edges of the histogram bar). Since data is scaled linearly, it can be expected that maximum difference between target and output will be of around 2%. In Figure 4.14, the gradient does not reach the minimum established and it could still decrease if more epochs are added, converging closer to a local minimum improving the performance, but results are already good and it would be more time-consuming.

To simulate the network and study the results, 10 different I-V curves will be tested (calculated with user-chosen random values of T_e, r_p and n_∞). The network output results are represented in Table 4.6, containing the target values and network output as well as the realtive error in percentage as given by Eq. 4.7 for each simulation. Note that y denotes target value and o output given by the network. The subindex 1 is for electron temperature and 2 for plasma density.

	r_p	T_e			n_e		
	(mm)	y_1 (eV)	o_1 (eV)	Error (%)	y_2 (m^{-3})	o_2 (m^{-3})	Error (%)
Sim 1	5.0	8	8.004	0.05	$5.0 \cdot 10^{15}$	$5.007 \cdot 10^{15}$	0.13
Sim 2		5	5.995	0.10	$1.0 \cdot 10^{16}$	$9.89 \cdot 10^{15}$	1.08
Sim 3		2	2.006	0.32	$1.0 \cdot 10^{17}$	$9.93 \cdot 10^{16}$	0.75
Sim 4		7	7.005	0.07	$2.5 \cdot 10^{15}$	$2.65 \cdot 10^{16}$	6.16
Sim 5		9	9.007	0.08	$8.0 \cdot 10^{15}$	$7.96 \cdot 10^{15}$	0.45
Sim 6		4	4.010	0.24	$6.0 \cdot 10^{16}$	$5.99 \cdot 10^{16}$	0.18
Sim 7	2.0	5	4.995	0.10	$8.0 \cdot 10^{15}$	$7.97 \cdot 10^{15}$	0.38
Sim 8	7.0	2	2.006	0.30	$3.2 \cdot 10^{16}$	$3.19 \cdot 10^{16}$	0.12
Sim 9	3.5	7	6.987	0.19	$1.0 \cdot 10^{15}$	$1.00 \cdot 10^{15}$	0.11
Sim 10	1.0	10	9.995	0.05	$1.0 \cdot 10^{17}$	$1.01 \cdot 10^{15}$	0.62

Table 4.6: Comparison of target values and network outputs for the inverse planar LP problem

Polynomial approach

Before establishing the method of the inverse mapping of Section 4.3.2, another approach was studied and the performance obtained was quite good. It is based on least square fitting.

The least squares method is applied to the I-V curve only for $\Delta V < 0$, obtaining a polynomial function of 5^{th} order which approximates the current according to,

$$\hat{I}_t = p_0 + p_1\Delta V + p_2\Delta V^2 + p_3\Delta V^3 + p_4\Delta V^4 + p_5\Delta V^5 + p_6\Delta V^6 \tag{4.11}$$

With this approach, the goodness of the fit needs to be evaluated too. To evaluate the deviation of the fit respect the curve, the statistic analyzed is the sum square error (SSE), computed as,

$$SSE = \sum_{m=1}^{l_{\Delta V}} \left(I_{tm} - \hat{I_{tm}} \right)^2 \tag{4.12}$$

where I_{tm} is the target current and $\hat{I_{p_m}}$ is the current at the same ΔV but approximated with the polynomial function of Eq. 4.11. The morphological parameters will be composed of the polynomial constants and the probe radius. The data set is built as in Table 4.7.

$$\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{P}$$

p_0	p_0	$\dots$	p_0	p_0	$\dots$	p_0	$\dots$	p_0
p_1	p_1	$\dots$	p_1	p_1	$\dots$	p_1	$\dots$	p_1
p_2	p_2	$\dots$	p_2	p_2	$\dots$	p_2	$\dots$	p_2
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
p_5	p_5	$\dots$	p_5	p_5	$\dots$	p_5	$\dots$	p_5
r_{p_1}	r_{p_1}	$\dots$	r_{p_1}	r_{p_2}	$\dots$	r_{p_2}	$\dots$	$r_{p,l_{r_p}}$

$T_{e,1}$	$T_{e,2}$	$\dots$	$T_{e,l_{T_e}}$	$T_{e,1}$	$\dots$	$T_{e,l_{T_e}}$	$\dots$	$T_{e,l_{T_e}}$
$n_{e,1}$	$n_{e,1}$	$\dots$	$n_{e,1}$	$n_{e,1}$	$\dots$	$n_{e,1}$	$\dots$	$n_{e,l_{n_e}}$

Table 4.7: Training data structure (inputs with the corresponding targets) for the inverse planar problem with the polynomial approach

Where P is calculated and has the same magnitude as in Eq. 4.8. In order to assure that the error carried from the fitting will not influence the results, the mean sum square error of all the subsets is calculated,

$$MSSE = \frac{1}{P} \sum_{p=1}^{P} SSE = 2.6260e - 08 \tag{4.13}$$

The approximation of the curve with a 5^{th} order polynomial results in a really low error as calculated in Eq. 4.13, strengthening the polynomial approach. The training performance is represented in Figures 4.18 and 4.19, which allow to have a quick overall performance in terms of error committed. The simulation input and network output will have the form,

1. Input: $\boldsymbol{a}^{(0)} = [p_0, p_1, p_2, p_3, p_4, p_5, r_p]^T$
2. Output: $\boldsymbol{o} = [T_e, n_\infty]^T$

Figure 4.18 contains the mean square error of the cost function using polynomials as morphological parameters. At the end of the training, the mse is almost 1e-09, a better performance than when using the current as input. Indeed, in Figure 4.19, the maximum error made in all training subsets

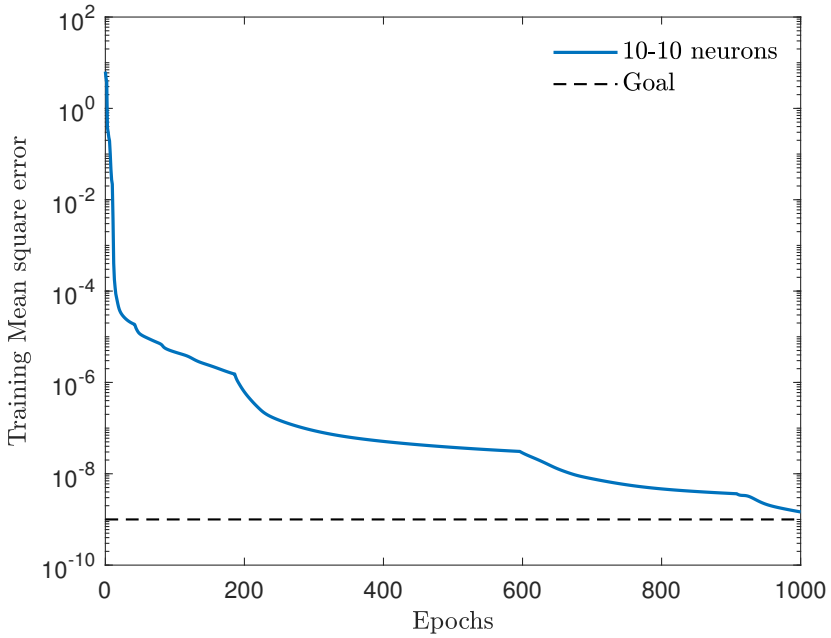

Figure 4.18: Mean square error of the training data for the planar LP case with the polynomial approach

is of 0.061%. Excellent performance in the simulations should be expected. The simulation results are shown in Table 4.8.

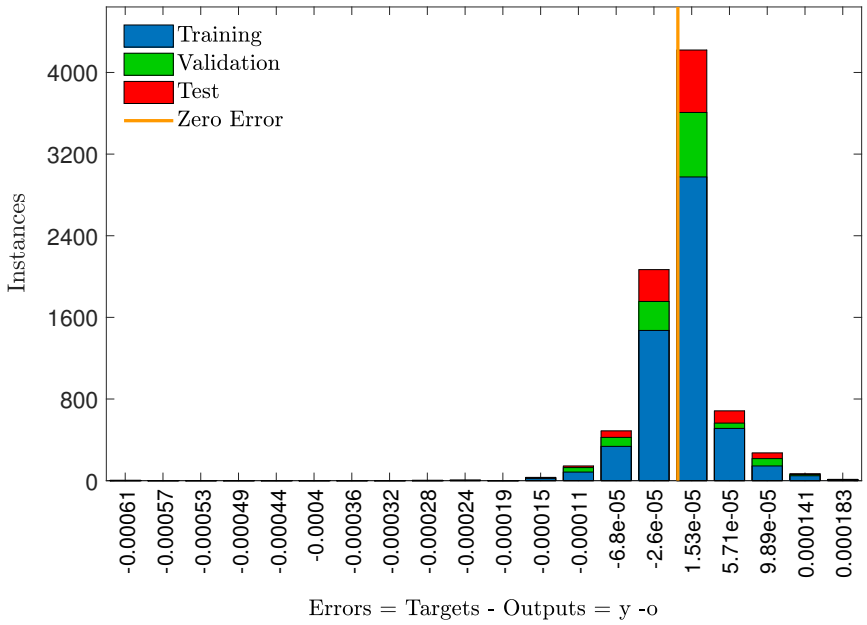

Figure 4.19: Error committed in the data set for the planar LP case with the polynomial approach

	r_p	T_e			n_e		
	(mm)	y_1 (eV)	o_1 (eV)	Error (%)	y_2 (m^{-3})	o_2 (m^{-3})	Error (%)
Sim 1	5.0	8	7.999	0.012	$5.0 \cdot 10^{15}$	$4.999 \cdot 10^{15}$	0.020
Sim 2		5	4.999	0.020	$1.0 \cdot 10^{16}$	$9.996 \cdot 10^{15}$	0.040
Sim 3		2	2.001	0.050	$1.0 \cdot 10^{17}$	$9.998 \cdot 10^{16}$	0.020
Sim 4		7	6.999	0.014	$2.5 \cdot 10^{15}$	$2.5002 \cdot 10^{15}$	0.008
Sim 5		9	9.001	0.011	$8.0 \cdot 10^{15}$	$7.995 \cdot 10^{15}$	0.063
Sim 6		4	3.9999	0.002	$6.0 \cdot 10^{16}$	$5.999 \cdot 10^{16}$	0.017
Sim 7	2.0	5	5.0003	0.006	$8.0 \cdot 10^{15}$	$7.996 \cdot 10^{15}$	0.050
Sim 8	7.0	2	2.0003	0.015	$3.2 \cdot 10^{16}$	$3.200 \cdot 10^{16}$	0.000
Sim 9	3.5	7	6.999	0.014	$1.0 \cdot 10^{15}$	$1.002 \cdot 10^{15}$	0.200
Sim 10	1.0	10	9.999	0.010	$1.0 \cdot 10^{17}$	$1.003 \cdot 10^{17}$	0.300

Table 4.8: Comparison of target values and network outputs for the inverse planar LP problem with the polynomial approach

The results obtained with the polynomial approach are much better than the discrete one. The error made with polynomials never goes above 0.063%, while the biggest error in discrete is found to be 6.16%. Here, the importance of finding a good set of morphological parameters is shown. Although the discrete approach has more inputs, the network finds easier to obtain the relation input/output from the polynomial coefficients. They represent a simpler and more effective way to represent the I-V curve for this case of planar LP.

4.4 OML simulation

Same procedure as for the planar LP problem is followed, with just small variations: When calculating the plasma parameters, the probe geometry will be fixed, $r_p = 0.127mm$ and $L_p = 2mm$. These values correspond to the probe used in the laboratory to obtain the experimental results that will be compared afterwards.

T_i will be kept constant with a value of $1eV$, since its value does not affect the results considerably. This statement is studied and explained in Section 4.5. Moreover, the user-defined training parameters for this case are:

1. Network structure: Two hidden layer with 10 neurons each, which seemed to have excellent results in the planar simulation.
2. Training algorithm: Levenberg-Marquardt with mean square error performance function.
3. Transfer function: Hyperbolic tangent for hidden layer and linear for output layer.
4. Maximum epochs: 1000
5. Goal: 1e-07
6. Minimum gradient: 1e-06

4.4.1 Forward simulation

The forward problem is, $I_t = f\,(T_e, r_p, n_\infty, \Delta V)$. Each subset contains 4 input variables (T_e, r_p, n_∞ and ΔV) and the associated total current, I_t. The number of subsets (P) will be the same as for Eq. 4.5 and the training data is build the same way as illustrated in Table (4.1). Only changes are found in some plasma parameter ranges: new range of T_e will be $[1-12]\,eV$ and range of r_p is $\left[8\cdot10^{-4} - 8\cdot10^{-5}\right]\,m$. The training performance is analyzed with the cost function and histogram error figures, to have an estimation of what error should be expected.

The mean square error of the training is represented in Figure 4.20. In this case, the performance decreases rapidly in the first iterations, and then starts converging more slowly. This phenomena does not affect in any way the network performance. The reason is the initialization of the weights and biases. They are initialized with random values, then the initial point within the multidimensional cost function will play a role on how fast it converges. If the point can find a steeper slope to the next point it will converge faster than if the point is initialized in a flatter surface. For this reason, it is a good practice, when initializing weights and biases randomly, to set the maximum number of epochs up to a value with certain margin, in case the initial point converges slower.

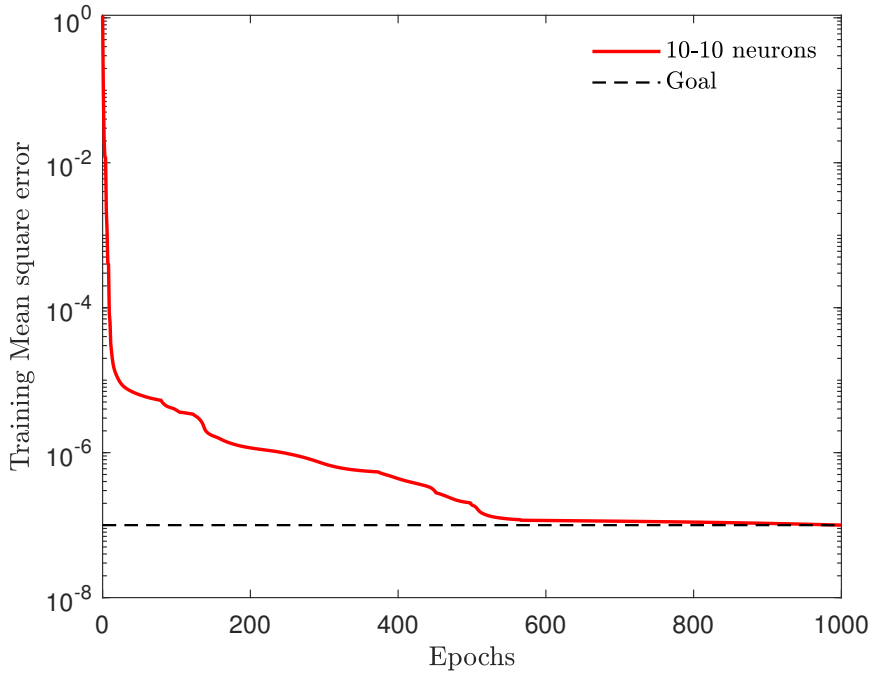

Figure 4.20: Cost function convergence during training of the forward OML simulation

Figure 4.21 anticipates small error between actual I-V curve and ANN output, with a maximum error in percentage of $\approx 0.8\%$. The network will be simulated for 4 different cases shown in Table 4.9.

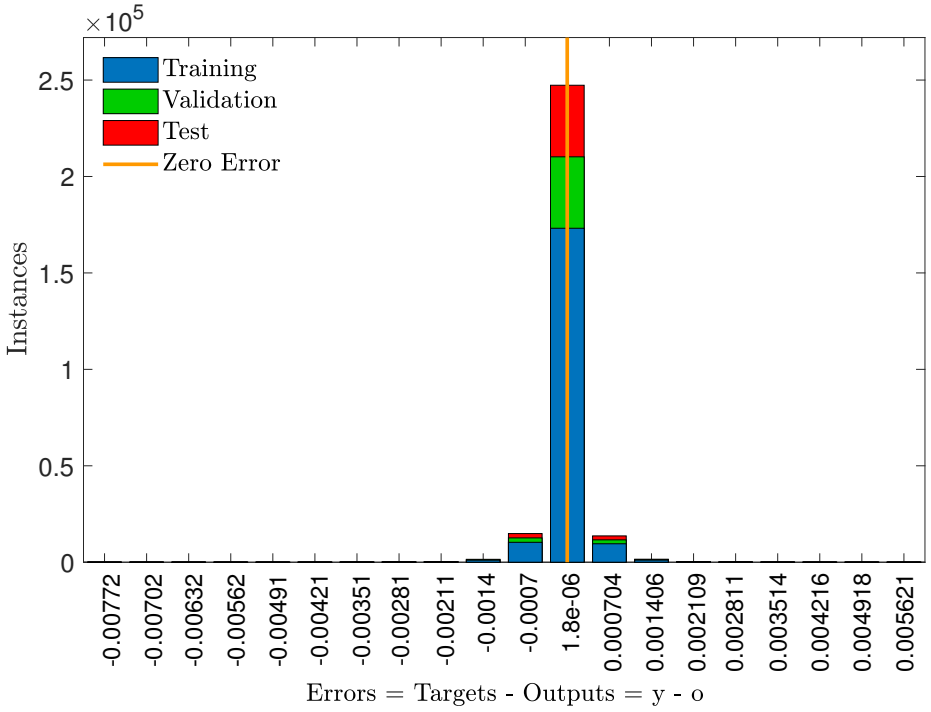

Figure 4.21: Error of each the training subsets after training of the forward OML simulation

	Simulation 1	Simulation 2	Simulation 3	Simulation 4
T_e (eV)	10	7	3	1.5
n_∞ (m^{-3})	10^{16}	$5 \cdot 10^{16}$	$2 \cdot 10^{15}$	10^{17}
r_p (mm)	0.5	0.25	0.7	0.09

Table 4.9: Values used for each simulation of the forward OML problem

Results on the forward simulation are shown in Figures 4.22 and 4.23. In Figure 4.22, the difference between theoretical and neural network output is difficult to differentiate, since the network is able to reproduce the curve almost perfectly. This error is shown in Figure 4.23, the maximum error does not even reach 1% in percentage error, reinforcing the ability of the neural network to solve the forward problem. Respect to the Planar LP problem, the peak of the floating potential is not found in this simulations. However, the larger error are found in the ion current collection zone where the absolute values are much smaller and percentage error does not show a proper comparison.

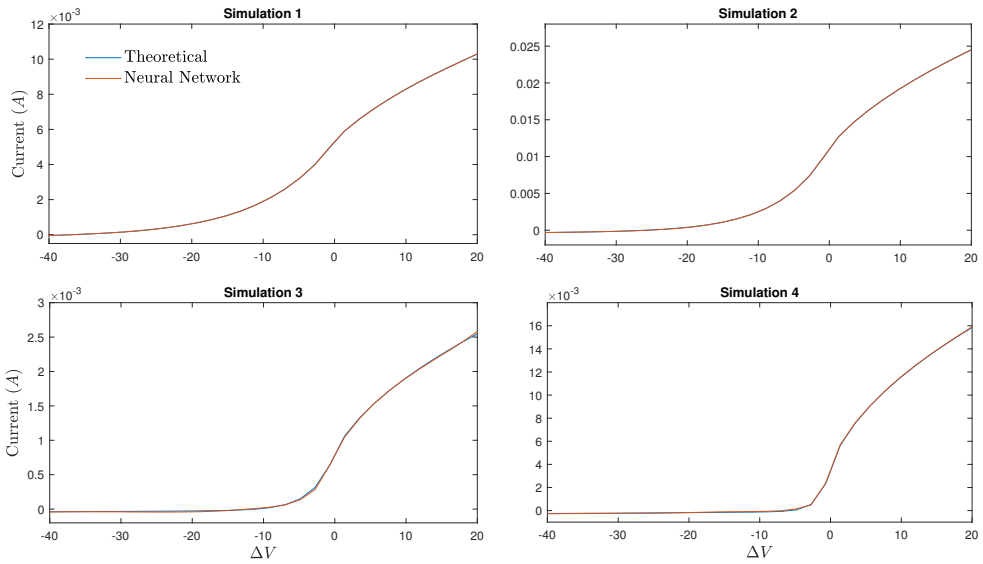

Figure 4.22: Theoretical vs Neural network results of the forward simulation of OML theory

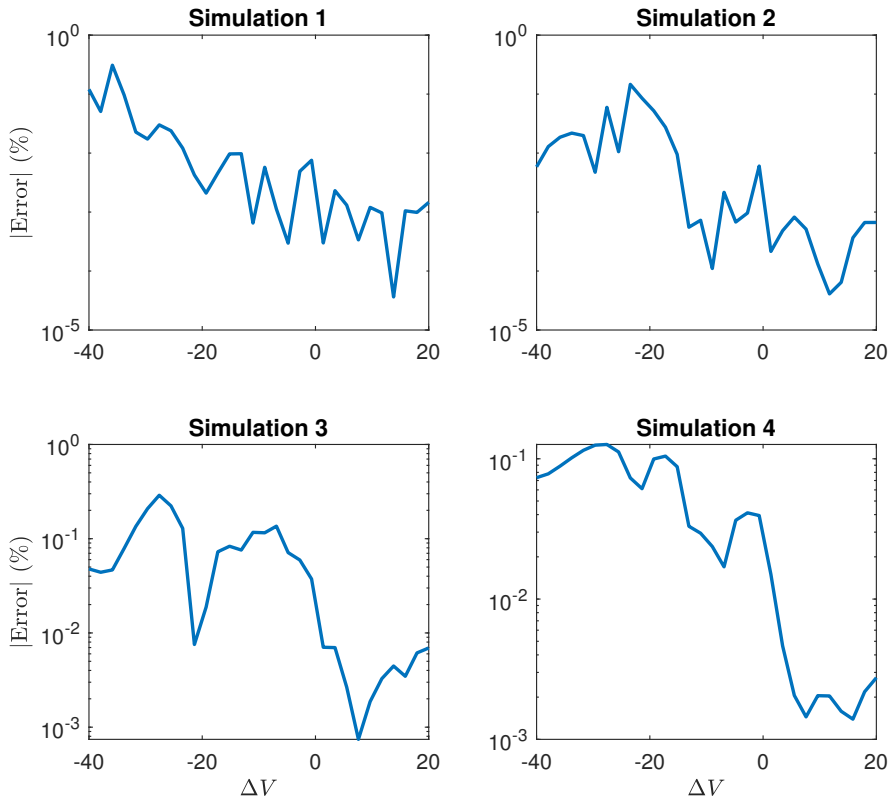

Figure 4.23: Relative error between theoretical and ANN result

4.4.2 Inverse Mapping

For the inverse simulation of the OML model, the plasma parameters are extracted from the I-V curve characteristics, the radius will not be included in the training data, since the problem is characterized for the probe of the UC3M laboratory. The number of subsets, P, and the structure of the training data are shown in Eq. 4.14 and Table 4.10 respectively, where $l_{T_e} = 30$ and $l_{n_\infty} = 30$.

$$P = l_{T_e} \times l_{n_\infty} \tag{4.14}$$

$\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{P}$

$I_p(\Delta V_1)$	$I_p(\Delta V_1)$	$\dots$	$I_p(\Delta V_1)$	$I_p(\Delta V_1)$	$I_p(\Delta V_1)$	$\dots$	$I_p(\Delta V_1)$
$I_p(\Delta V_2)$	$I_p(\Delta V_2)$	$\dots$	$I_p(\Delta V_2)$	$I_p(\Delta V_2)$	$I_p(\Delta V_2)$	$\dots$	$I_p(\Delta V_2)$
$I_p(\Delta V_3)$	$I_p(\Delta V_3)$	$\dots$	$I_p(\Delta V_3)$	$I_p(\Delta V_3)$	$I_p(\Delta V_3)$	$\dots$	$I_p(\Delta V_3)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$I_p(\Delta V_{end})$	$I_p(\Delta V_{end})$	$\dots$	$I_p(\Delta V_{end})$	$I_p(\Delta V_{end})$	$I_p(\Delta V_{end})$	$\dots$	$I_p(\Delta V_{end})$

$T_{e,1}$	$T_{e,2}$	$\dots$	$T_{e,l_{T_e}}$	$T_{e,1}$	$T_{e,2}$	$\dots$	$T_{e,l_{T_e}}$
$n_{e,1}$	$n_{e,1}$	$\dots$	$n_{e,1}$	$n_{e,2}$	$n_{e,2}$	$\dots$	$n_{e,l_{n_e}}$

Table 4.10: Training data structure (inputs with the corresponding targets) for the inverse oml problem

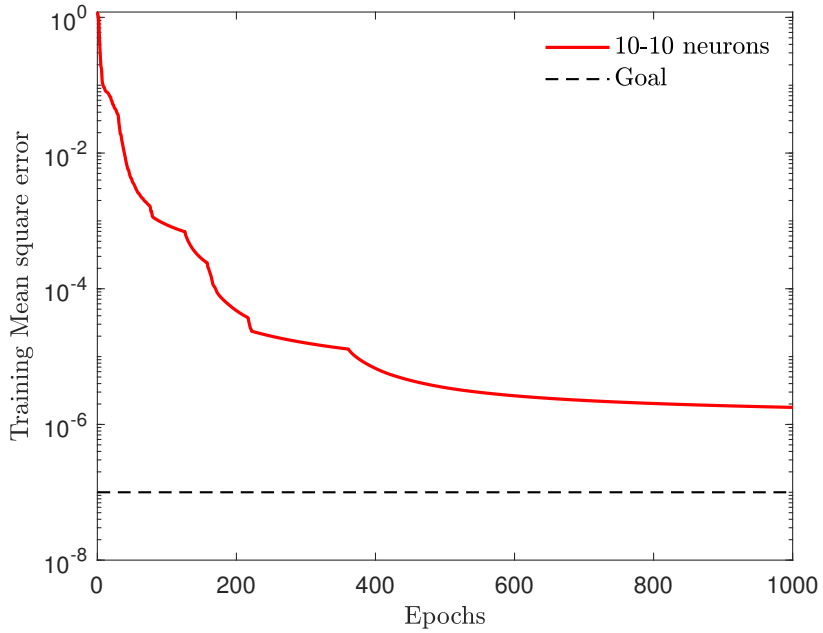

Figure 4.24: Cost function convergence during training of the inverse simulation of OML theory

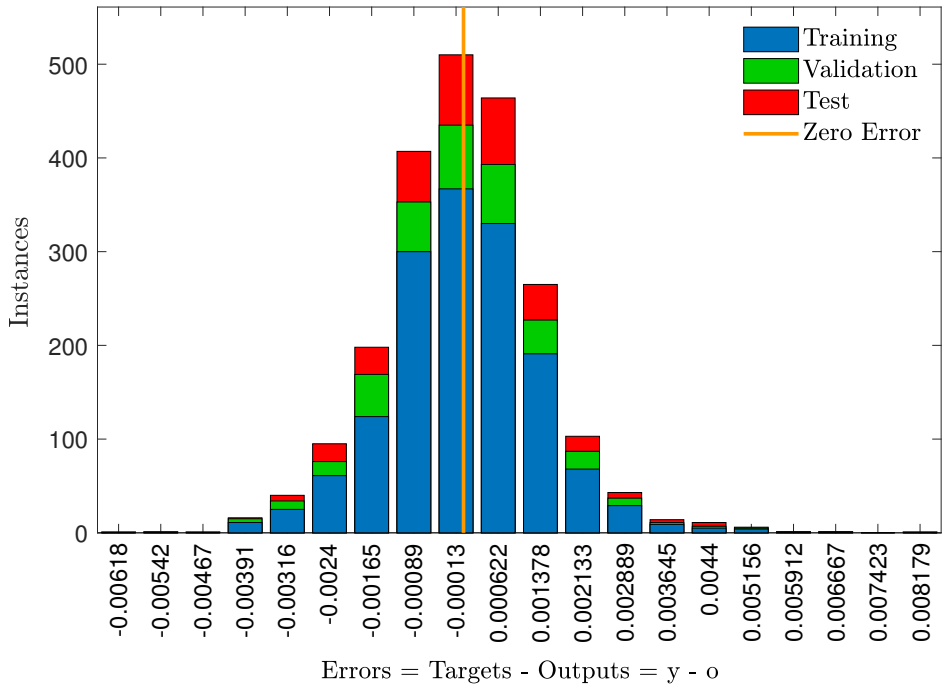

Figure 4.25: Error of each the training subsets after training of the inverse simulation of OML theory

Although most of the errors are not in the central bar of the histogram plot and are more distributed along the all the bars, the maximum and minimum errors are of the order of 10^{-3}, expecting accurate results. The same information is extracted from the mean square error, which at the end of the training is about 10^{-6}. Afterwards, the network is simulated with the input shown in Eq. 4.15, and network output will be given as Eq. 4.16,

1. Input: The network is fed directly with the values of the current associated to the given value of ΔV and calculated with the target values of T_e and n_∞

$$\boldsymbol{a}^{(0)} = [I_t(\Delta V_1), I_t(\Delta V_2), I_t(\Delta V_3), \ldots, I_t(\Delta V_{end})]^T \tag{4.15}$$

2. Output: The network output is composed of the two plasma parameters,

$$\boldsymbol{y} = [T_e, n_\infty]^T \tag{4.16}$$

	T_e			n_∞		
	y_1 (eV)	o_1 (eV)	Error (%)	y_2 (m^{-3})	o_2 (m^{-3})	Error (%)
Sim 1	9	9.013	0.143	$7.0 \cdot 10^{16}$	$6.982 \cdot 10^{16}$	0.253
Sim 2	2	1.988	0.594	$2.0 \cdot 10^{16}$	$1.996 \cdot 10^{16}$	0.185
Sim 3	5	4.998	0.031	$4.0 \cdot 10^{15}$	$3.996 \cdot 10^{15}$	0.103
Sim 4	1	0.992	0.859	$5.0 \cdot 10^{17}$	$4.995 \cdot 10^{17}$	0.103
Sim 5	8	8.003	0.038	$1.0 \cdot 10^{15}$	$9.974 \cdot 10^{14}$	0.263
Sim 6	11	10.995	0.042	$8.0 \cdot 10^{17}$	$8.060 \cdot 10^{17}$	0.748
Sim 7	7	6.998	0.022	$6.0 \cdot 10^{15}$	$6.011 \cdot 10^{15}$	0.185
Sim 8	12	11.99	0.059	$7.0 \cdot 10^{16}$	$6.956 \cdot 10^{16}$	0.633
Sim 9	6	6.001	0.016	$1.0 \cdot 10^{16}$	$1.001 \cdot 10^{16}$	0.083
Sim 10	3	3.009	0.309	$9.0 \cdot 10^{15}$	$9.019 \cdot 10^{17}$	0.507

Table 4.11: Comparison of target values and network outputs for the inverse OML problem

From Table 4.11, the accuracy of the results is found to be very good as expected from the training performance. The discrete approach works perfectly for OML theory also. The polynomial approach was not found to be handy since the error made in the fitting was important and results deviated from the target ones This is mainly because of the form of the curve. Planar theory for $\Delta V < 0$ has a similar form of a parabola, but the OML theory has a more complicated one, increasing the error of the fitting.

4.5 Ion temperature effect on the network output

Ion temperature has been assumed constant and equal to $1eV$ during both simulations, planar and OML theory. This conclusion comes from the fact that changes on ion temperature do not result in significant changes in the I-V curve. Consequently, the plasma parameters remain almost

unchanged by this fact. In order to study the variation of the current with respect to the ion temperature, for $\Delta V < 0$ since it is when ion collection occurs, the derivative is calculated. From Eqs. 2.18, 2.19, 2.21, and 2.22,

$$\frac{\partial I_t}{\partial T_i} = -\frac{I_{the}\sqrt{\frac{T_e T_i}{\mu_i}}\left[\sqrt{\pi}\ \text{erfcx}\left(\sqrt{\frac{-T_e\Delta V}{T_i}}\right) + 2e^{T_e\Delta V/T_i}\sqrt{\frac{-T_e\Delta V}{T_i}}\right]}{2\sqrt{\pi}T_i} \tag{4.17}$$

The derivative of the current can be plotted against the ion temperature for different ΔV, to observe the magnitude of the ratio of change. The geometry of probe remains unaltered, and electron temperature and plasma density will remain constant with magnitudes of $9eV$ and $10^{17}\ m^{-3}$.

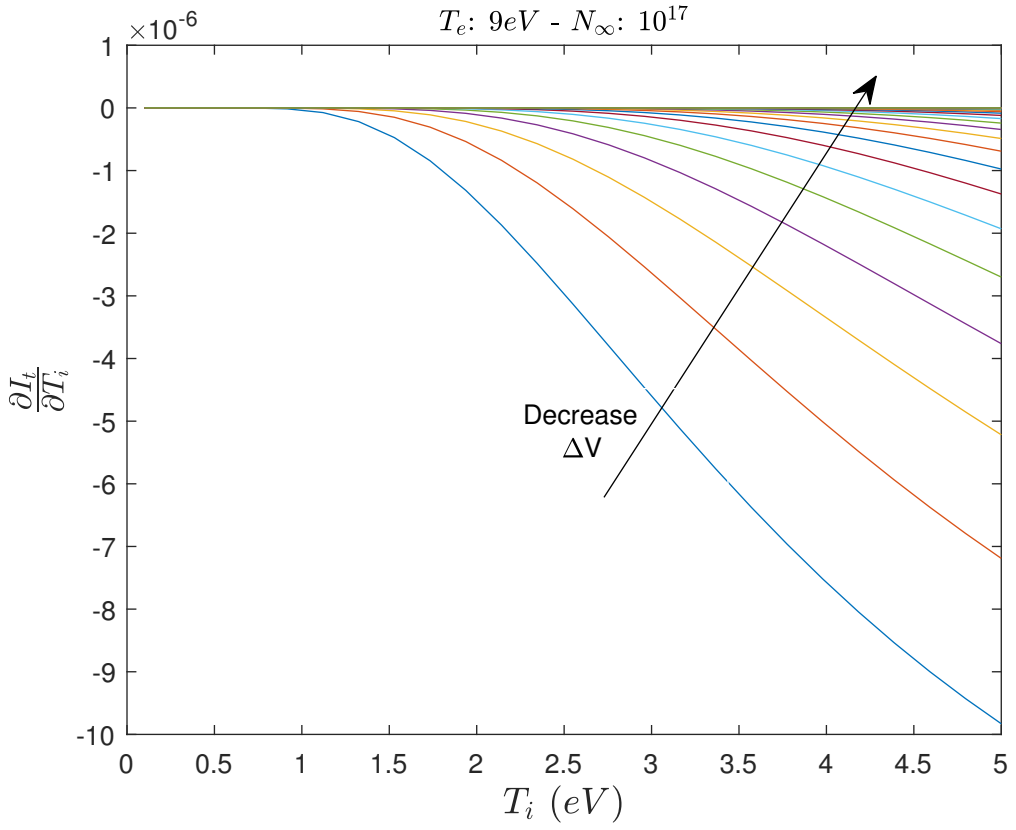

Figure 4.26: Derivative of the total current respect to ion temperature for $\Delta V < 0$

The function represent the sensitivity of the total current to changes in the ion temperature. The order of magnitude of this rate of change is $10^{-6} - 10^{-5}$. The variation in current can expressed as,

$$\Delta I_t = \frac{\partial I_t}{\partial T_i}\Delta T_i \tag{4.18}$$

	T_e	n_∞
Target	9	10^{17}

	Output (eV)	Error (%)	Output (m^{-3})	Error (%)
$T_i = 1eV$	9.004	0.044	$1.003 \cdot 10^{17}$	0.299
$T_i = 0.1eV$	9.011	0.122	$1.004 \cdot 10^{17}$	0.398
$T_i = 2eV$	8.989	0.122	$1.003 \cdot 10^{17}$	0.299
$T_i = 4eV$	8.975	0.278	$1.001 \cdot 10^{17}$	0.100
$T_i = 5eV$	8.966	0.378	$1.001 \cdot 10^{17}$	0.100

Table 4.12: Output values of plasma parameters when varying ion temperature from the actual value

where $\Delta T_i = T_{i,used} - T_{i,actual}$, is the error between the ion temperature used in the calculations and the real (unknown) ion temperature. Comparing the order of magnitude of this increment of current with the order of magnitude, which is 10^{-3}, of the total current at this ΔV, it can be stated that, although the estimation of ion temperature is not correct, the change that will produce this error in the total current values is small enough so that it will not influence the output given by the network.

A numerical example is studied. A network was trained with I-V curves assuming that $T_i = 1eV$. Then, the same network was tested for different I-V curves, each of them calculated with different values of T_i. The results and differences in T_e and n_∞ are shown.

From the results in Table 4.12, it can be stated that the lack of information about the ion temperature does not affect considerably the results. Moreover, it can be noticed that changes in T_i mainly affects the electron temperature, not plasma density. From here, information about the inside of the network functioning can be extracted. When training for this range of ΔV, plasma density is not influenced by the change of T_i, meaning that the network gives less importance to the input data corresponding to ion current collection in order to calculate plasma density. On the other hand, T_e is slightly more influenced by input data corresponding to $\Delta V < 0$.

Moreover, the little influence that T_i has over the output, results in difficulties to obtain it as an output parameter. The network has a hard time differentiating from one value of T_i to another, since changes in the input by variations of ion temperature are almost negligible compared to variations in current by a change in T_e or n_∞. This is the reason why T_i was kept constant during the whole evaluation.

4.6 Experimental simulation

From the laboratory, experimental data was obtained. The Langmuir probe was introduced in the plasma chamber and the bias was varied to construct the I-V curve. Each data file contains values of total current when varying the bias. Moreover, a rough estimation of the electron temperature and plasma density was given along the current law to have something to compare to. The value

of plasma parameters were computed according to Lobbia in [28].

This method of calculating plasma parameters is an approximate evaluation that might results in considerably error, but it will provide an arguable estimation. Plasma potential, V_s, will be calculated using the theoretical method, which consist of identifying the potential at the inflection point of the curve [7].

Figure 4.27 shows every I-V curve given by the lab measurements with the corresponding T_e and n_∞ calculated with the previous least squares method. To obtain different measurements, the probe was located at different positions, given by the parameter $r^\star$.

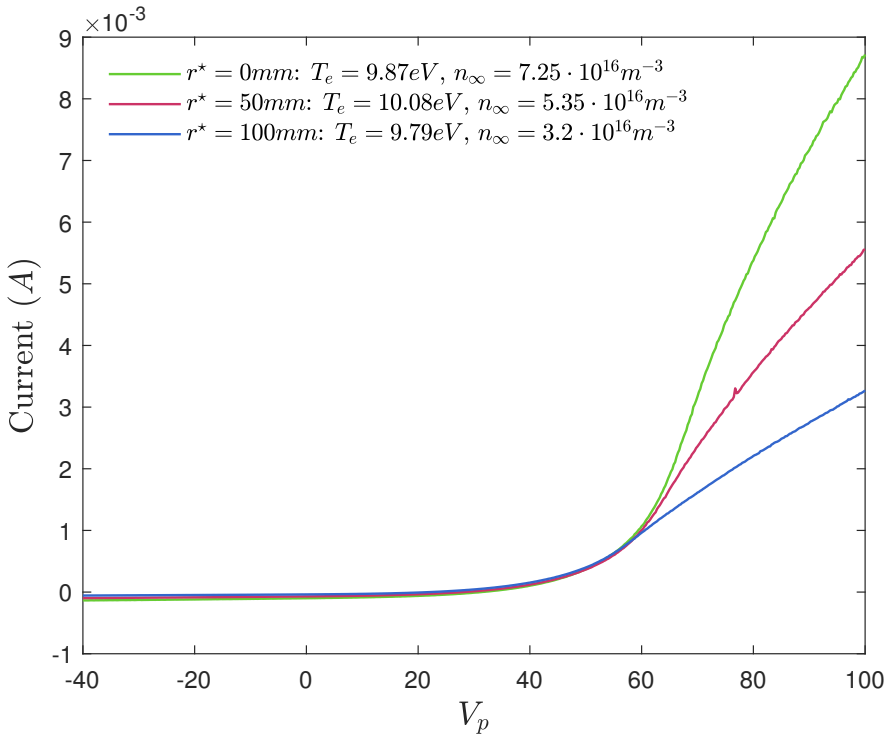

Figure 4.27: Lab measurements at different positions with their associated electron temperature and plasma density estimated

The network was trained exactly the same as the OML inverse simulation, with the exception of the range of ΔV, which now will be $[-80, -10]\, V$. There two reasons behind this choice:

1. Data close to the plasma potential is not useful, since the magnetic field produced modifies the electron current more than ion current.
2. When electron current is to large, it will disturb the plasma.

Experimental data gives the current for a given bias. The maximum value of the data derivative will give V_s. Plasma potential is subtracted to the bias, obtaining ΔV. To simulate the network with the experimental data, the values of current used as input are those given by the same values of ΔV used for the network training. Once those values of current are found, the network is ready to be simulated. The results are shown in Table 4.13.

	V_s (V)	T_e (eV)			n_∞ (m^{-3})		
		Lobbia	o_1	Error (%)	Lobbia	o_2	Error (%)
$r^\star = 0mm$	$75V$	9.87	8.26	16.3	$7.25 \cdot 10^{16}$	$7.29 \cdot 10^{16}$	0.55
$r^\star = 50mm$	$76.5V$	10.08	8.94	12.7	$5.35 \cdot 10^{16}$	$4.80 \cdot 10^{16}$	11.6
$r^\star = 100mm$	$73V$	9.79	10.79	10.2	$3.20 \cdot 10^{16}$	$3.33 \cdot 10^{16}$	3.90

Table 4.13: Results of the network when simulating experimental data

From the results obtained, it is difficult to extract conclusions about their accuracy. The least square method yields an approximate value, but not the actual one, so they should not be trusted as the target ones. However, it can be said that results are good with a acceptable margin of error. The values of plasma density are similar, meaning that n_∞ should be around that value, as proved by two different methods. Figure 4.28 shows the experimental current measurements compared with the OML curve built with the T_e and n_∞ output values of the network.

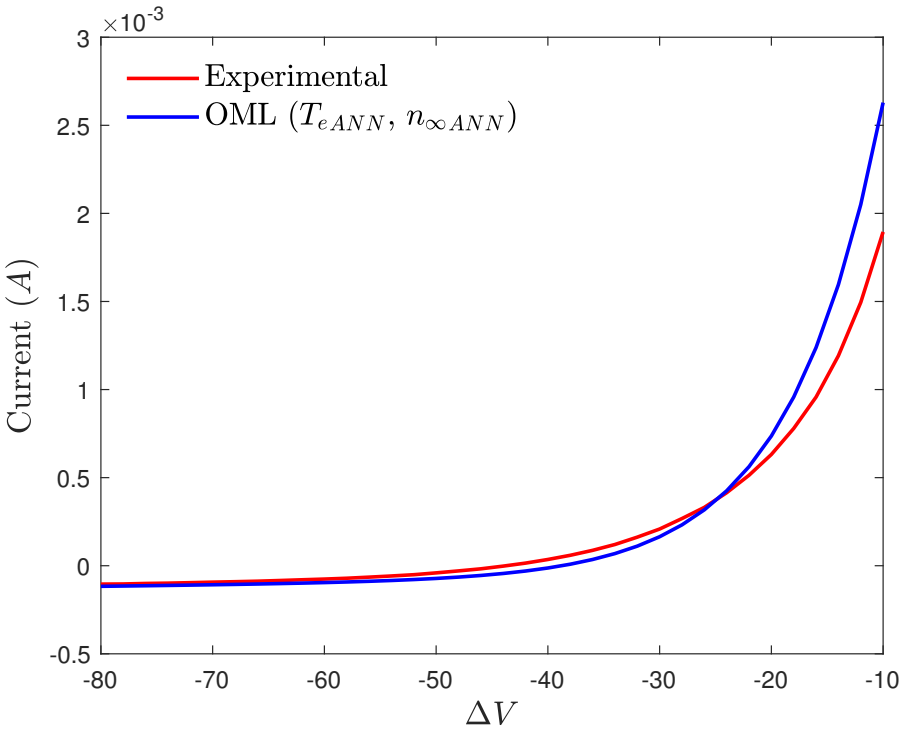

Figure 4.28: Experimental current measurements comparison with OML calculated with the network output values for the probe postion $r^\star = 0mm$

From Figure 4.28, it can be observed clearly the error between lab measurements and the output of the network. The ANN was trained with data obtained directly with the theoretical equations and tries to give values of T_e and n_∞ associated to the curve given (experimental data). However, it fails to give parameters which fit the experimental data. From this plot, it can be stated that results obtained are an approximation of around what value plasma parameters should be, but nothing accurate. To confirm the veracity of the network. The OML curve of Figure 4.28 is fed

to the network and the output should be the same value that was given to build it, $T_e = 8.26eV$ and $n_\infty = 7.29 \cdot 10^{16}$.

T_e (eV)			n_∞ (m^{-3})		
y_1	o_1	Error (%)	y_2	o_2	Error (%)
8.26	8.26247	0.0299	$7.29 \cdot 10^{16}$	$7.28950 \cdot 10^{16}$	0.0068

Table 4.14: Verification of network functioning. Network fed with OML curve obtained with output values of experimental data

Now that the network functioning has been verified, the imperfect fit between the neural network output and the experimental curve was studied. To compare the output results and analyze possible sources of error, the results for the probe position $r^\star = 0mm$ are analyzed. The first source of error is the estimation of V_s. It was done using the inflection point method, and there is not assuredness that is the correct value. In fact, in Figure 4.29 it is demonstrated how changes in plasma potential affect greatly the output values of plasma parameters calculated by the network. They are plotted against the variation of V_s respect the plasma potential calculated with the inflection method, $V_s = 75V$.

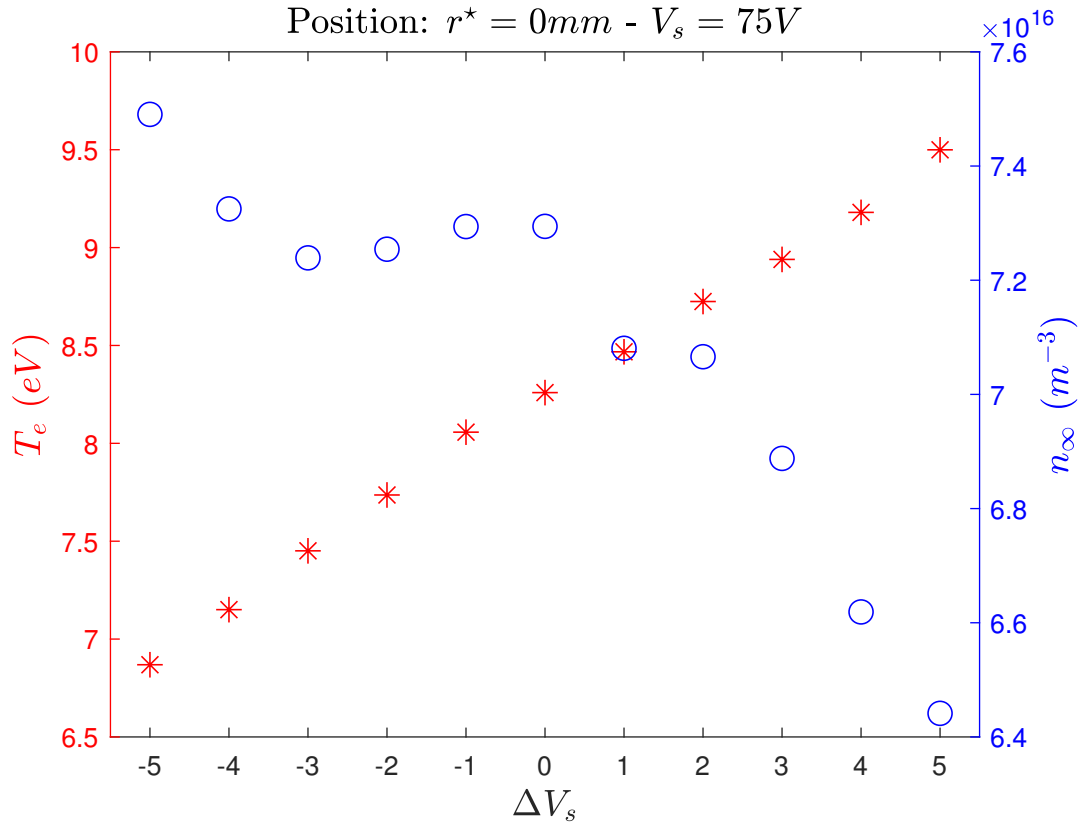

Figure 4.29: Plasma parameters given by the network when changing the plasma potential a value ΔV_s respect plasma potential calculated with the inflection point method.

If plasma potential is miscalculated by an error of $-5V$ or $5V$, the error in T_e would be of 17% and 15% respectively, and the error in n_∞, 2.7% and 12% respectively. From this study, obtaining an accurate fit requires plasma potential to be estimated precisely. The idea of using the neural network to calculate plasma potential would not work, because of the basic principle that the input needed by the network is the current associated to a given ΔV, which in turn it is unknown because of plasma potential. Therefore, a guess of V_s is needed.

Another reason because of the imperfect fitting lies on the validity of the model used, OML theory. Since the network is trained with that theory, it will learn and improve its performance from the relations established in it. The validity and limits of the OML theory will play an important role in the prediction of the results. This theory is an upper limit on the ion current collected by the probe, that is to say, the maximum current that is able to collect for that fixed radius. It is not the real case, since some ions can be reflected by specific potential distributions around the the probe [1]. In Figure 4.29, this fact can be noticed. Although, the value of plasma parameters are not the exact ones, the ion current it is slightly observed to be greater (in absolute value) for the OML theory. This theory will overestimate the ion current, which obviously will result in different output values. Therefore, the goal to obtain the best fit given by the network respect to experimental data, would be to train the network with the most accurate theory and being able to predict plasma potential precisely. The next step in network training is to implement the Orbital-Motion Theory (OM) described by Laframbroise in [25]. This theory relaxed the two first assumptions of the Planar LP theory, keeping the others. The suppression of this two assumption will make the theory harder, but more accurate. However, it has been demonstrated that ANN can be trained by any kind of theory. If a database with this theory is calculated by any means, ANN can be implemented and used to solve the inverse problem.

Chapter 5

Conclusion and Future work

Artificial Neural Networks has been demonstrated to be a powerful tool for the analysis of data. The power of neural networks lies in their simplicity and the great variety of problems it can solve. Once the network is trained, it becomes once of the fastest method to compute the output because of its simple computations. Neural networks present numerous advantages. They are able of handling any kind of data and are able to establish relations between input and output, even if the user does not know them beforehand.

Artificial Neural Network has been successfully implemented in Planar theory and OML theory, both forward and inverse problem. A framework based on theoretical background has been created and analyzed for future research.

The ANN was learned from scratch and important conclusions about it have been learned. ANN is as powerful as you desired it to be. One of the most important aspect of ANN is the data pre-processing. Network without data pre-processing work very poorly. The data was scaled between the activation function limits, improving performance and time, with the drawback of being more tedious. It is one of the most critical steps for the correct functioning.

Other key aspect of ANNs is their structure. The architecture is found to play a important role on the accuracy of the final results. It is determined that using more hidden layers is more beneficial than using more neurons, since they add dimensionality to the problem, improving the network ability to extract information about input/output relations.

This input/output relation plays an influential role. The input and output given to the network (morphological parameters) must reflect the relation desired for the network to understand it. The morphological parameters were completely different for inverse and forward problem. For forward problem, the complete set of plasma parameters is needed along with the voltage. For the inverse problem, two approaches studied resulted with good results: discrete and polynomial approach.

Polynomial approach provided better results for the Planar theory, but failed to give any good results in the OML theory. On the other hand, the discrete approach served as a more general and universal method and was implemented successfully in both Planar and OML theory with excellent results predicting the plasma parameters.

Therefore, it is concluded that in the theoretical frame of plasma diagnostics, T_e and n_∞ can be calculated by any theory with an incredible small margin of error. The other plasma parameter used in the equations, T_i was found to be negligible in the influence of the I-V curve. Diagnosis using neural networks performed badly in output parameters that affect very little the overall magnitude of the inputs. T_i was set as constant during the whole evaluation without any significant variation in the results.

Although, the theoretical problem was solved perfectly, the importance of this method is its viability of being implemented for experimental data. It can be said that the neural network produced nearly as good results as the classical method of Lobbia [28]. Nevertheless, the disadvantage of using ANN for plasma diagnostic was found to rely in the estimation of plasma potential. V_s was calculated using a theoretical approach proved by [7]. In the experimental measurements, I_e is affected by the magnetic field, concluding that the estimation of V_S is probably wrong and resulted in a negative influence in the output of the network.

Here is where the major disadvantage of the neural network is found. To work properly, it should be simulated with data as similar as the one with which is trained. Training the network with data obtained with theoretical equations and simulating it with experimental data might not be the best approach, because of the discrepancies that exist between theory and practice. But here lies the main problem about plasma diagnostics, it is difficult to obtain a theory that approximates reality. Then, the plasma parameters obtained can be stated to be within a reasonable margin of error, compared to the conventional processing. Error is analyzed and improvements can be suggested. However, ANN as proposed by this work can be used once the theory is ready to obtain plasma parameters from any proposed theory.

In order to achieve a better performance with the current theories different alternatives are proposed. Adding the plasma potential as an output could be an important feature for the later implementation in the experimental data. Obtaining a different set of morphological parameters with stronger I/O relations. Study the effect of a ΔV mesh in the results and which values of ΔV will give the curreant values that will result in the best performance of the network. Also, an increase in the database range is proposed, improving the robustness of the network, but a more powerful processor would be needed.

It was concluded that the theories studied are not close to reality and a better theory is needed. The next step is to implement the Laframboise theory, which is expected to work perfectly as long as a god database can be produced. This theory will resemble results more similar to reality improving accuracy in the plasma parameters.

One of the most promising ideas to solve the diagnostic problem is to train with experimental data. If the error comes from the difference between theory and experiments. Training with experimental data would allow the network to learn from it and adapt its behavior to extract relation from lab measurements directly. The big disadvantage of this method is that plasma parameters needs to be known for the network to be trained. If they can be know with certain accuracy, this is the best alternative to implement ANN to plasma diagnostics.

A remoter solution is to create a network that can learn by itself, this is known as unsupervised learning. This method needs tons and tons of data, but it could be able to learn independent of which theory is applied. It could be used as an open source code where everyone can include

databases, creating a robust neural network for plasma diagnostics application.

Artificial Neural Networks could be useful in cases where the essential requirement is speed and huge amount of data needs to be processed. To obtain sufficient results, data should be pre-processed accordingly and the accuracy on certain parameters shuold be improved, such as V_s. If ANN are desired to be used in plasma diagnostics applications involving experimental measurements, it should exist an standardization of the data, so that pre-processing has a common basis. However, the main idea extracted from this work is that no matter which theory is used or which experiment, Artificial Neural Network works.

Bibliography

[1] M. R. Akdim, *Modelling of complex plasmas*, PhD thesis, Utrecht University, Netherlands, 2003.

[2] S. S. D. Arunodhayan, C. Henri, Sushmitha, C. Paartha, Melwin, Timothy, and Nonoshka, *Neuron the memory unit of the brain*, IOSR Journal of Computer Engineering, 17 (2015), pp. 48–61.

[3] P. J. Braspenning and F. Thuijsman, *Artificial neural networks: an introduction to ANN theory and practice*, vol. 931, Springer Science & Business Media, 1995.

[4] E. T. Center, *The fluorescent lamp*, 2013.

[5] F. F. Chen, *Introduction to plasma physics and controlled fusion volume 1: Plasma physics*, vol. 38, Plenum Press, 1985.

[6] ——, *Langmuir probe diagnostics*, IEEE-ICOPS Meeting, Jeju, Korea, 2 (2003).

[7] X. Chen and G. Sánchez-Arriaga, *Current-voltage and floating-potential characteristics of cylindrical emissive probes from a full-kinetic model based on the orbital motion theory*, Journal of Physics: Conference Series, 958 (2018), p. 012001.

[8] T. C. Corke, M. L. Post, and D. M. Orlov, *Single-dielectric barrier discharge plasma enhanced aerodynamics: concepts, optimization, and applications*, Journal of propulsion and power, 24 (2008), pp. 935–945.

[9] H. Demuth and M. Beale, *Neural network toolbox*, vol. 2000, Mathworks Inc, 1992.

[10] D. Frank-Kamenetskii, *Plasma: the fourth state of matter*, Springer Science & Business Media, 2012.

[11] R. J. Goldston and P. H. Rutherford, *Introduction to plasma physics*, CRC Press, 1995.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] D. Graupe, *Principles of artificial neural networks*, vol. 7, World Scientific, 2013.

[14] M. T. Hagan, H. B. Demuth, M. H. Beale, et al., *Neural network design*, vol. 20, Pws Pub. Boston, 1996.

[15] M. T. Hagan and M. B. Menhaj, *Training feedforward networks with the marquardt algorithm*, IEEE transactions on Neural Networks, 5 (1994), pp. 989–993.

[16] S. Hamzavi, A. Hilgers, J.-P. Lebreton, and P.-O. Dovner, *Langmuir probe plasma measurements reduction using a feed-forward neural network: Preliminary results*, in Environment Modeling for Space-Based Applications, vol. 392, 1996, p. 207.

[17] S. S. Haykin, *Neural networks and learning machines*, vol. 3, Pearson Upper Saddle River, NJ, USA:, 2009.

[18] S. He, N. Sepehri, and R. Unbehauen, *Modifying weights layer-by-layer with levenberg-marquardt backpropagation algorithm*, Intelligent Automation & Soft Computing, 7 (2001), pp. 233–247.

[19] N. Hershkowitz, *How langmuir probes work*, Plasma diagnostics, 1 (1989), pp. 113–183.

[20] R. Hood, B. Scheiner, S. Baalrud, M. Hopkins, E. Barnat, B. Yee, R. Merlino, and F. Skiff, *Ion flow and sheath structure near positively biased electrodes*, Physics of Plasmas, 23 (2016), p. 113503.

[21] I. Hutchinson, *Principles of plasma diagnostics*, Springer Science & Business Media, 2012.

[22] E. Illenberger and J. Momigny, *Some general remarks on positive and negative ions, photoionization, and electron attachment*, in Gaseous Molecular Ions, Springer, 1992, pp. 1–7.

[23] R. Jahn G. and E. Choueiri Y., *Electric propulsion*, Encyclopedia of Physical Science and Technology, 2 (2002).

[24] M. Klindworth, *Fundamentals and applications of Langmuir probe diagnostics in complex plasmas*, PhD thesis, Christian-Albrechts Universität Kiel, 2005.

[25] J. G. Laframboise, *Theory of spherical and cylindrical langmuir probes in a collisionless, maxwellian plasma at rest*, tech. rep., TORONTO UNIV DOWNSVIEW (ONTARIO) INST FOR AEROSPACE STUDIES, 1966.

[26] I. Langmuir, *Oscillations in ionized gases*, Proceedings of the National Academy of Sciences, 14 (1928), pp. 627–637.

[27] M. A. Lieberman and A. J. Lichtenberg, *Principles of plasma discharges and materials processing*, John Wiley & Sons, 2005.

[28] R. B. Lobbia and B. E. Beal, *Recommended practice for use of langmuir probes in electric propulsion testing*, Journal of Propulsion and Power, (2017), pp. 1–16.

[29] N. N. T. MathWorks, *User's guide*, Mathworks Inc, (1998).

[30] S. Mazouffre, *Electric propulsion for satellites and spacecraft: established technologies and novel approaches*, Plasma Sources Science and Technology, 25 (2016).

[31] R. L. Merlino, *Understanding langmuir probe current-voltage characteristics*, American Journal of Physics, 75 (2007), pp. 1078–1085.

[32] T. M. Mitchell, *Machine learning*, WCB/McGraw-Hill, 1997.

[33] H. Mott-Smith Jr, *Phys. rev.*, Phys. Rev., 28 (1926), p. 27.

[34] M. A. Nielsen, *Neural networks and deep learning*, Determination Press, 2015.

[35] A. Piel, *Plasma physics: an introduction to laboratory, space, and fusion plasmas*, Springer, 2017.

[36] F. Rafaj, *Analysis of Langmuir probe data by artificial neural network*, PhD thesis, Univerzita Karlova, Matematicko-fyzikální fakulta, 2016.

[37] M. Rafiq, G. Bugmann, and D. Easterbrook, *Neural network design for engineering applications*, Computers & Structures, 79 (2001), pp. 1541–1552.

[38] K. Riemann, *Plasma and sheath*, Plasma Sources Science and Technology, 18 (2009), p. 014006.

[39] S. Sapna, A. Tamilarasi, M. P. Kumar, et al., *Backpropagation learning algorithm based on levenberg marquardt algorithm*, Comp Sci Inform Technol (CS and IT), 2 (2012), pp. 393–398.

[40] I. Shafi, J. Ahmad, S. I. Shah, and F. M. Kashif, *Impact of varying neurons and hidden layers in neural network architecture for a time frequency application*, in Multitopic Conference, 2006. INMIC'06. IEEE, IEEE, 2006, pp. 188–193.

[41] R. M. Shiffrin and R. C. Atkinson, *Storage and retrieval processes in long-term memory.*, Psychological Review, 76 (1969), p. 179.

[42] J. J. Wang, K. S. Choi, L. H. Feng, T. N. Jukes, and R. D. Whalley, *Recent developments in dbd plasma flow control*, Progress in Aerospace Sciences, 62 (2013), pp. 52–78.

[43] B. M. Wilamowski and H. Yu, *Improved computation for levenberg–marquardt training*, IEEE transactions on neural networks, 21 (2010), pp. 930–937.

[44] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, Springer, 2017.