# Fast ion detection in collective Thomson scattering spectra with neural networks

Pliatskas Stylianidis, C.

*Award date:*
2019

# Fast Ion Detection in Collective Thomson Scattering spectra with Neural Networks

Christos Pliatskas Stylianidis [*1]

[1]Technical University of Eindhoven

December 6, 2018


Department of Applied Physics
Science and Technology of Nuclear Fusion Group

| | |
|---|---|
| TU/e supervisor: | Prof.Niek Lopes Cardozo |
| Daily supervisor: | MSc.Ivana Abramovic |
| External supervisor: | Dr.Robert Wolf |

[*]c.pliatskas.stylianidis@student.tue.nl

**Abstract**

In this research, we investigate the feasibility of fast ion detection in Collective Thomson Scattering (CTS) spectra with neural networks. Towards this aim, we developed a code to simulate the spectral form factor of the CTS signals. We used that code to create a database of synthetic spectra. In our investigation we explored two scattering geometry configurations; geometry A simulates the applied geometry in the Wendelstein 7 X stellarator while geometry B is a configuration defined in this project, inspired by fast ion investigations in literature. This database was used for the training and performance evaluation of the neural networks. First, we investigated the fast ion inference with regression networks which yielded promising results but not sufficient accuracy. We also applied classification networks which achieved very high performance scores for geometry B. In conclusion, neural networks can provide a reliable fast inversion tool for the detection of fast ions.

## *Acknowledgements*

First of all, I would like to thank my supervisors in this thesis project, Ivana Abramovic and Niek Lopes Cardozo for their guidance and their advice. They taught me plenty of lessons regarding Thomson scattering, data analysis and the principles that should rule scientific work. Also, I would like to thank Jonathan van den Berg and Andrea Pavone for sharing with me their expertise on neural networks and machine learning in general. Furthermore, I would like to thank the Max Planck institute in Greifswald for hosting me during a part of the project and giving me to opportunity to experience the latest operation campaign. Last but not least, I want to thank my family and friends for their constant support throughout the development of this thesis project as well as during the entirety of my M.Sc. studies in Eindhoven.

# Contents

# Part I
# Introduction

## 1 Introduction

### 1.1 Background

One of the major issues that modern society has to face is the increasing energy demand. This increase can be attributed to the global population growth as well as the advancement of technology which has become an inseparable part of daily life. Currently the main energy source are fossil fuels. Combustion of fossil fuels is a chemical process which provides large amounts of energy. However, the abundance of fossil fuels is limited which means that they cannot serve as a permanent solution for the increasing energy demand.

Therefore, other means of energy production are actively investigated, aiming to serve as a lasting resort. One of these options is nuclear fusion, the nuclear reaction that fuels the stars. In fusion, energy is produced when nuclei of light elements combine in order to create heavier and more stable nuclei. Since it is a nuclear process, the energy released per reaction is around six orders of magnitude higher compared to the energy produced through chemical reactions like fossil fuel combustion. Additionally, the fuel is virtually inexhaustible, making fusion a strong candidate for a future energy source.

Closely related to the energy crisis is the protection of the natural environment. The impact of the industrial human activity on the environment has been catastrophic, with the consequences becoming more evident with each passing day. As a result, the favored energy source should not only provide a long-term solution but also be environmentally friendly. Thankfully, nuclear fusion meets that criterion as it does not produce any high level nuclear waste. The waste from the operation of a fusion plant will have the same level of radioactivity as the ones from today's industries and hospitals. Waste management for this type of materials is already developed and well established.

Nuclear fusion can be achieved when the reacting nuclei come close to each other. However both of the nuclei are positively charged so there is a strong repelling Coulomb force between them (Figure:1). The reactants need to overcome this barrier and get close enough to allow the quantum tunneling effect to take place. For that purpose, they are heated to extremely high temperatures in the order of hundreds of millions of Kelvins creating a hot plasma. Even under these conditions, a contesting process with a much higher probability than nuclear fusion is Coulomb scattering. Therefore to achieve net fusion energy gain the plasma needs to be confined. A plasma confinement mechanism, which also prevents the melting of the machine's inner wall, employs magnetic fields. Due to the gyromotion of the charged particles, particle and energy transport perpendicular to the magnetic field lines is inhibited. Plasma confinement for the motion parallel to the field lines is achieved with a toroidal configuration. Additionally a poloidal field is required to provide plasma stability which in combination with the toroidal field leads to a helical magnetic field.

The are two main designs of magnetic confinement devices: the tokamak and the stellarator (Figure:2). Both of these devices are toroidal and the main difference between them lies in the way that the helical magnetic field configuration is created. In the case of tokamak, an axisymmetric toroidal field is created by a set of coils that envelope the plasma torus. The poloidal field is generated by inducing a toroidal plasma current. This inductive current leads to pulsed operation

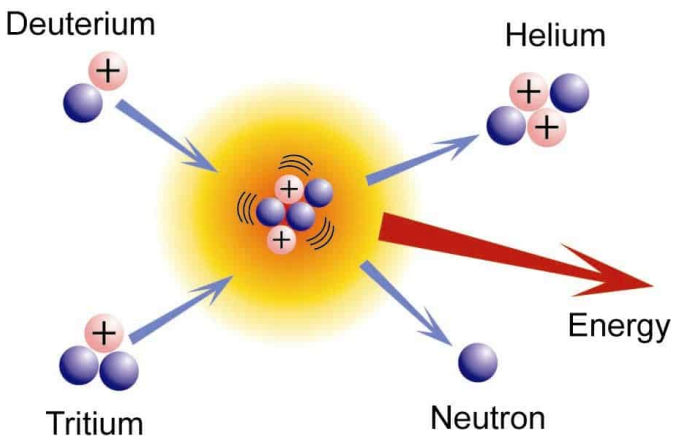



Figure 1: Nuclear fusion is a reaction where light nuclei (*e.g.* hydrogen isotopes like deuterium and tritium) combine to produce heavier nuclei, in this case Helium. An additional result of the reaction is the release of energy. Image source:[1]

of the tokamak which is a major disadvantage for a fusion power plant. Additionally, the presence of a current can drive instabilities in the plasma. On the other hand, the stellarator design uses external non-planar external coils to create the three dimensional magnetic configuration. Although the design, production and assembly of the device is more difficult, it has the advantage that it can operate in steady state and there are no current driven instabilities as in the tokamak case.

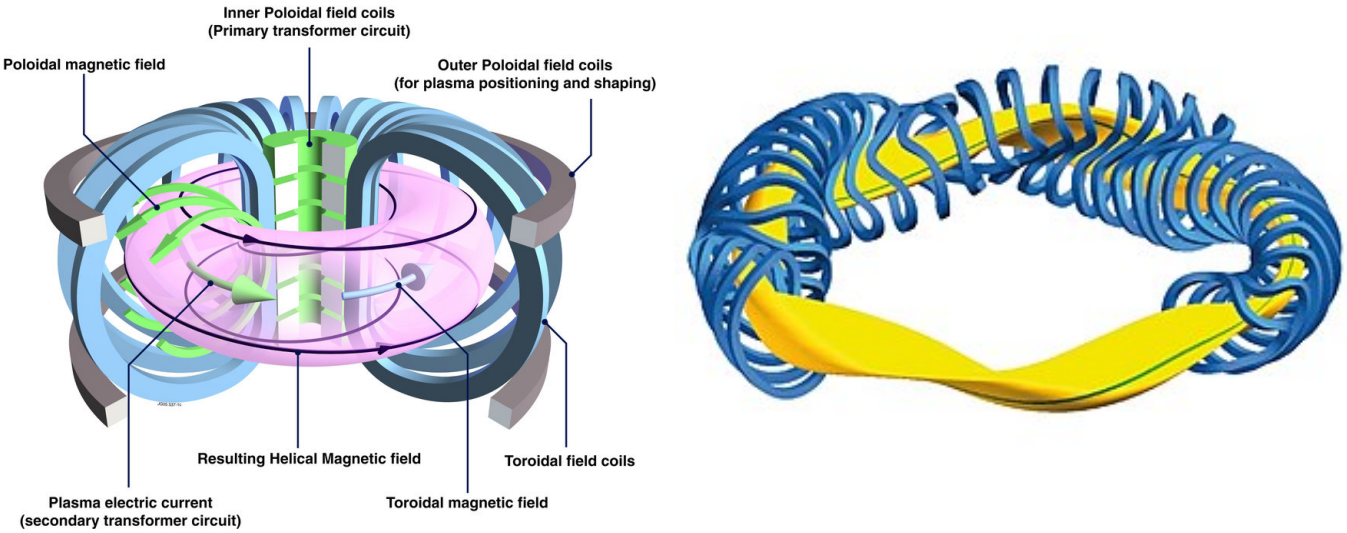



Figure 2: The two main designs for magnetic confinement machines are the stellarator(right) [2] and the tokamak (left) [3].

Any fusion device, regardless if it is a tokamak or a stellarator, is accompanied by a set of diagnostic tools. These are used to provide information about the plasma parameters which are necessary for the control of the plasma. Future fusion power plants will require uninterrupted and safe operation in order to be economically viable. For both of these reasons high quality diagnostic tools and methods are necessary. Furthermore simplicity ,which can be translated into a minimization of the diagnostics required for operation, is important for economic viability.

One of the diagnostics that is currently applied in large fusion experiments is Collective Thomson Scattering (CTS). CTS refers to the scattering of electromagnetic radiation off plasma fluctuations in the electron density, current density, the electric and magnetic field. From those four parameters the electron density is in most cases the dominant one. Fluctuations of the electron density can be attributed to the movements of ions, as they drag electrons along their path. The scattered radiation of CTS can provide information on the fluctuations and consequently on the ion motion. The parameters characterizing the ion motion are: the ion temperature, the fast ion velocity distribution and the drift velocity. Clearly, CTS can be used to extract a lot of information about the plasma parameters. Additionally, it can be operated in the microwave range so the requirements on material properties for the plasma facing components and the transmission line are less strict and the receiver can be placed far from the plasma [4]. These are some of the properties that constitute CTS a strong candidate for the diagnostics necessary in future power plants.

Based on their energy the ions of the plasma can be distinguished in two populations: the bulk ions and the high energetic or fast ions. Both of these groups contribute to the CTS scattered spectrum. The scattered radiation is Doppler shifted compared to the incident beam. Therefore, analysis of this Doppler shift can provide information about the velocity of the ions. The velocity distribution of the bulk ions is characterized by the ion temperature and is the one that dominates the spectrum. The fast ions are the main source of heating for the bulk ions through collisions in reactor scenarios. The sustainability of fusion reactions relies on this mechanism since continuous external heating will not allow a net power gain. Additionally, the fast ion population can carry significant current and efficient non-inductive current drive is one of the fusion research goals. Lastly, it has been demonstrated that the fast ions can influence the occurrence of plasma instabilities (Sawtooth instabilities, Fishbone instabilities, interaction with Toroidal Magnetic Field Ripples *etc.*)[5].

## 1.2 Research question

From the preceding discussion, it is clear that information on the fast ion population is valuable and CTS can be used to determine their velocity distribution. However, the detection of the fast ion component is challenging. There are three sources of fast ions: fusion reactions, neutral beam injection (NBI) and ion cyclotron resonance heating (ICRH). These sources give rise to velocity distributions which in principle are not Maxwellian. Analytical expressions of the scattered spectrum are available for some distribution cases like the Maxwellian, the loss-cone and the slowing down distribution. The derivation of scattered spectrum's analytical expressions for more complex distributions is a difficult task [6].

Although CTS is a valuable diagnostic tool, it has the drawback of a low signal-to-noise ratio which makes the detection of the weak fast ion signal harder. In many fusion experiments, CTS operates in the microwave frequency range since it provides more flexibility in terms of experimental geometry [7]. In these cases, the main source of background noise is Electron Cyclotron Emission (ECE) in the plasma. The power of the CTS signal can be up to 100 times lower compared to the background signal [8]. The situation is even more difficult if the beam used for CTS measurements is the same as the one used for plasma heating with Electron Cyclotron Resonance Heating (ECRH). This is the case for the stellarator Wendelstein 7-X in Greifswald, Germany where part of this thesis was conducted.

The first step towards the measurement of fast ions on W 7-X is an increase of the signal-to-noise ratio. The CTS signal is dependent on plasma parameters as well as scattering parameters

which together span a large parameter space (7-8 parameters). In order to investigate this space and find a configuration for which fast ion detection is possible, a fast inversion algorithm is needed. This could be achieved through neural networks which have proven successful in fast inversion measurements of the bulk ion temperature with CTS [4]. Before their application, the neural networks require a database of CTS spectra for their training. For this requirement, a forward model which simulates the form factor of a CTS spectrum was developed and a database of synthetic spectra was constructed. In short, the aim of this thesis project could be summarized in the following research question:

**Is it feasible to use neural networks for fast ion detection in CTS spectra? If yes, can neural network techniques for data analysis optimize the signal-to-noise ratio of the diagnostic?**

Before answering the research question, it is helpful to break it down into sub-questions which would be easier to answer. These sub-questions can be the following:

1. How will the fast ion spectrum be simulated? Which velocity distributions will be examined?
2. Which parameters affect the CTS spectrum and in particular its fast ion component? What is their expected range of values?
3. Which of these parameters should be modified to create a database of synthetic spectra?
4. Which parameters should be used as an input for the training of the neural network?
5. Which parameter could serve as an indication of fast ion presence in the CTS spectrum?
6. Under which circumstances (if any) can the neural network provide accurate predictions about the presence of fast ions in the CTS spectrum?

## 1.3 Methodology

Before addressing the research question, it was deemed useful to provide the reader with some necessary information regarding the theory of CTS and some of the main concepts of Artificial Neural Networks (ANN). An introduction to CTS and ANNs' theory is presented in Sections 2 and 3 respectively. This thesis is structured according to the steps of the methodology followed to answer the research question and its sub-questions. In Section:4, a method to calculate the spectral density function for an arbitrary distribution through a polynomial approximation is presented. Additionally, the slowing-down model and its analytical expressions for the CTS spectrum components are introduced. Once the model was completed and benchmarked, the investigated parameters and their expected range were defined. Some initial results regarding the form of the spectra in the explored parameter spaces and their dependency on the modelling parameters are also presented. Based on this information, a few databases of synthetic spectra were generated.

The next step was the determination of the neural network's characteristics: model, architecture, design parameters, input and output variables. Two approaches were pursued in that aspect: classification and regression. For each of these approaches several networks were constructed and trained on the databases of the synthetic spectra. The characteristics of the investigated networks and their performance on the training sets of the database are analyzed in Section:5 of this thesis.

The same section also address the ability of the networks to generalize their conclusions and provide accurate predictions. Therefore, preliminary results of their performance on synthetic spectra that were not included in their training process are also presented.

The main result along with supplementary results from the modelling process, the classification and regression networks and their analysis are presented in Section:6. A summary of the research and discussion of its main result is presented in Section:7. The same section also addresses the limitations of this work and recommendations for follow-up research. Some options for future work and extension of this project's application field are given in Section:8. Lastly, an overall evaluation of the project followed by concluding remarks can be found in Section:9.

# 2 Collective Thomson Scattering Theory

Thomson scattering refers to the scattering of electromagnetic radiation by free charged particles. From a physics point of view, the electromagnetic fields of the incident radiation accelerate the ions and electrons of the plasma. As a result, the accelerated particles emit radiation, defined as scattered radiation, which is proportional to this acceleration [9]. Since the ions have a much higher mass than the electrons, their acceleration is much smaller and as a result their contribution to the scattered radiation is in most cases negligible. The phases of the emitted radiation fields depend on the phase of the incident radiation at the position of the free charges. If the incident wavelength is large compared to the inter-particle distance and the charges have a uniform distribution, then the emitted fields cancel out. Density fluctuations of the plasma can lead to constructive contribution of the scattered fields [8] [9].

At sufficiently high densities, the emitted field by each particle is affected by the other charges. This is the case when the incident frequency is in the range of the plasma frequency $\omega_{pe}$.

$$\omega_{pe} = \sqrt{\frac{n_e q_e}{m_e \epsilon_0}} \tag{1}$$

According to its definition, plasma frequency is solely dependent on the electron density $n_e$. The rest of the symbols are: the elementary charge $q_e$, the electron mass $m_e$ and the vacuum dielectric constant $\epsilon_0$.

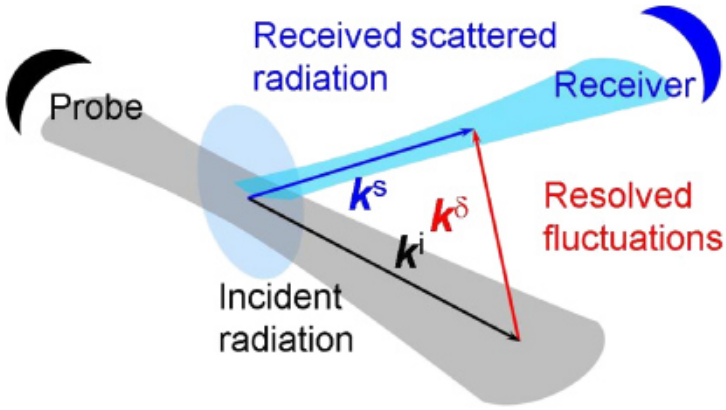



Figure 3: The incident radiation is scattered off the plasma fluctuations and detected by the receiver. The analysis of the scattered radiation can provide an insight for the fluctuations along the vector difference of incident and scattered beams. Image source:[10]

When the incident radiation frequency is in this range, then it is easier to consider the scattered radiation as a result of fluctuations in the dielectric properties of the plasma. One of these properties, usually the most dominant one, is the electron density [8].

Independently of the description, a Fourier component of the fluctuation can be defined as:

$$(\mathbf{k}, \omega) = (\mathbf{k}_s - \mathbf{k}_i, \omega_s - \omega_i) \tag{2}$$

,where $\mathbf{k}$ is the wave vector, $\omega$ is the frequency and the subscripts i and s refer to incident and scattered respectively. The incident radiation is scattered off the plasma fluctuations and detected by the receiver (Figure:3). The fluctuations along the vector difference of incident and scattered beams can be resolved through the analysis of the scattered radiation.

For diagnostic purposes, the incident wave has too high frequency for the ions to react so scattering is determined by fluctuations in the electron density, velocity distributions and the local fields. As it was mentioned before, the scattering from the ion population is negligible. However their presence affects the electron distribution and by extent the scattered radiation. The polarization of the plasma accompanied by the redistribution of charge to provide shielding from external electric fields is a dielectric phenomenon called Debye shielding. A measure of this effect is the Debye length, which indicates the radius for the electrostatic effect in a medium.

$$\lambda_{De} = \sqrt{\frac{k_B T_e \epsilon_0}{q_e^2 n_e}} \tag{3}$$

In the above expression, $T_e$ refers to the electron temperature and $k_B$ is the Boltzmann constant. As a result of Debye shielding the movement of ions induces changes in the electron distribution which in turn influences the scattered radiation. In a simple picture, the scattered field will be Doppler shifted compared to the incident radiation and therefore provide information about the ions' velocities. Consequently, the velocities of the ions in the direction of $\mathbf{k}$ can be inferred by the magnitude and form of the scattered radiation spectrum [8] [9].

## 2.1 Salpeter parameter

The effect of the ion dynamics in the electron density calculations was first presented in a theoretical paper in 1960 [11]. A parameter was introduced which can indicate the dominance of the ion dynamics in the electron density fluctuations. This parameter, the Salpeter parameter, compares the scale of the fluctuations with the Debye length (Equation(3)).

$$\alpha = \frac{1}{k\lambda_{De}} \tag{4}$$

When the Salpeter parameter is greater than unity the electrons behave collectively and the electron density fluctuations are dominated by the ion dynamics. This type of scattering is defined as collective scattering. For $\alpha < 1$ the spectrum of the scattered light reflects the thermal motion of the electrons [8].

## 2.2 Scattered spectrum composition

This section addresses the current form of CTS as it is expressed through the transfer equation. This equation relates the received spectral power density with the incident radiation and the plasma properties. The transfer equation is the following [8]:

$$\frac{\partial P_s}{\partial \omega} = P_i O_b \lambda_i \lambda_s r_e^2 n_e G \quad \frac{S(\mathbf{k}, \omega)}{2\pi} \tag{5}$$

where:
$\partial P_s/\partial\omega$: Spectral power density of the accepted power at the receiver
$P_i$: Power of the incident beam
$O_b$: Beam overlap. A measure of the overlap between launcher (incident radiation) and receiver (scattered radiation) beam patterns (light blue region in Figure:3)
$\lambda_i$: Incident wavelength in vacuum
$\lambda_s$: Scattered wavelength in vacuum

$r_e$: Classical electron radius
$n_e$: Electron density
G: Geometrical or Dielectric form factor. This factor describes the coupling of the incident to the scattered wave [12].
$S(\mathbf{k}, \omega)$: Spectral density function

This thesis focuses on the last term, the Spectral density function which defines the form of the spectrum. This term contains the contribution of the different particle populations of the plasma and can be expressed as follows:

$$S(\mathbf{k}, \omega) = \quad S_e(\mathbf{k}, \omega) + \sum_i S_i(\mathbf{k}, \omega) \tag{6}$$

where $S_e$ is the thermal fluctuation spectrum of the electrons by themselves and $S_i$ are the contribution of the electron fluctuations driven by the motion of the different ion species. As it was mentioned earlier, the ion populations can be separated in two categories, the thermalized bulk ions and the highly energetic fast ions. If the plasma contains impurities, they also contribute to the scattered spectrum, extending the summation of $S_i(\mathbf{k}, \omega)$ for each impurity species.

It is possible to account for fluctuations in plasma dielectric properties other than the electron density. In that case, each fluctuating property will have a corresponding geometrical factor and a spectral density function. The extended transfer equation would include a summation over the different dielectric property fluctuations in G and $S(\mathbf{k}, \omega)$ [8].

## 2.3 Signal-to-noise ratio

One of the challenges that CTS has to face is the relatively high levels of background noise. In some fusion experiments CTS operates in the millimetre wavelength range. This choice provides more flexibility for the operating geometry while remaining in the collective regime [7]. However, Electron Cyclotron Emission (ECE) has a strong presence in this spectral range and is the main contributor of the background signal in a CTS measurement. The background noise level can be up to three orders of magnitude higher than the signal[8].

In order to distinguish the signal from the background, the probing gyrotron is alternating between on and off states. The signal is derived by subtracting the background measurement, made in the gyrotron-off period, from the measured sum of the signal with background, made in the gyrotron-on period. More information about this techniques can be found in reference [5]. Another approach to determine the ECE background is to estimate its contribution by using channels where no CTS signal is expected allowing a comparison between the estimated and measured background ECE. This method was demonstrated to have smaller discrepancies between the measured and estimated ECE background signals compared to the probe modulation technique of the CTS signal [13].

The post-detection signal is defined as the ratio of the signal divided by the standard deviation of the estimate of the signal[8]:

$$\begin{aligned} \frac{S}{N} &= \frac{E\{P_s\}}{V\{P_s\}} \\ &= \frac{E\{P_s + P_b\} - E\{P_s\}}{V\{P_s + P_b\} - V\{P_s\}} \\ &= \frac{P_s}{2P_b}\sqrt{\eta W T} \end{aligned} \tag{7}$$

,where:
$E\{\}$: Estimate of the signal
$V\{\}$: Standard deviation of the signal
$P_s$: Power of the signal
$P_b$: Power of the background
$\eta$: Useful fraction of the chopping period of gyrotron's modulation
$W$: Bandwidth of the receiver's channel
$T$: Total integration time

The minimum desirable value for the signal to noise ratio is 10. For low levels of ECE, the signal to noise ratio can be improved by a factor of 10 with the increase of the pulse duration [7]. If the CTS probing frequency is chosen to be higher than the expected electron cyclotron emission, then potential absorption can be avoided and the background noise will be lower. However,the fast ion component will still be hard to identify since its signal is weak compared to the contributions from the bulk ions and the electrons [14] [15].

In this thesis, it is investigated whether the use of neural networks can improve the signal-to-noise ratio. This improvement could be accomplished either by reducing the standard deviation of the estimates or by accurate distinction between noise and signals. In addition, the neural networks can perform very well in pattern recognition problems [16] [17], so they could prove accurate in identifying the fast ion feature without necessarily improving the signal to noise ratio of the diagnostic.

## 2.4 CTS simulation models

In order to analyze the CTS signals and extract information about the plasma properties there have been developed codes in the form of forward models which can simulate these spectra [13] [6]. These codes use two main treatments for the plasma in their model development. The first and more complete approach is a fully electromagnetic treatment of the plasma [13]. This approach is more mathematically involved and usually is combined with kinetic transport codes. The second approach operates in the electrostatic approximation [18], which means that the assumed electric field is not coupled with the magnetic field. This approximation is useful in cases where the magnetic field is well-defined and not changing quickly. This approximation is used for simplification when the time consuming electromagnetic treatment is not necessary. One of this cases are the CTS codes developed for W-7 X [18].

Furthermore, these codes use assumptions for the velocity distributions of the plasma. In most cases, the bulk ions and electrons are assumed to have a thermal isotropic Maxwellian distribution characterized by their temperature. The fast ions of the plasma have non-thermal distributions which depend on their source of origin. In cases where the fast ions are the result of NBI or fusion reactions, then they have either isotropic or anisotropic slowing-down distributions. This is also the case that was examined in this research project as it will be analyzed in Section:4.2.

Thirdly, the response of the plasma populations depends on whether they are treated as magnetized or unmagnetized. This assumption refers to the ability of the particles to complete a cyclotron orbit within the scattering volume [19]. Usually, the fast and bulk ions are treated as unmagnetized and magnetized respectively. There are cases where the electron population received either the unmagnetized [19] or the magnetized [20] treatment.

In literature, CTS simulation codes calculate the expected scattered power density (Equation (5))[13] [10] which makes the comparison with experimental data from scattering experiments

easier. Alternatively, there are investigations focused on the spectral density function (Equation (6)) [19] [6] [20]. The latter is also the case of the work that is presented in this thesis report. In order to compare with experimental data this type of codes need to be combined with ray-tracing algorithms and also take into account the beam polarizations.

Lastly, there are two approaches that are followed in the investigation of fast ions in CTS signals. The first approach, is a mapping from the velocity space to the frequency space. In this case the forward model simulates the expected CTS spectrum based on the species velocity distribution and the scattering parameters. The second approach, is a mapping from the frequency domain to the velocity space. This approach infers the projection of the fast ion velocity distribution in the direction of the fluctuations from the scattered spectrum [13] [10]. This approach is more complicated but it can provide very useful insights. Particularly, if more than one fluctuation directions are resolved, then reconstruction of the fast ion 2-D velocity distribution can be achieved. Essentially this approach is a tomography of the fast ion velocity space [21].

# 3 Artificial Neural Networks (ANN)

An artificial neural network (ANN) is a type of machine learning algorithm inspired by the biological neural systems. It is a computing system that consists of a large number of artificial neurons - mathematical functions - that simulate the process of biological neurons [4]. These numerous neurons behave as a simplified version of their biological counterparts, work in parallel and transmit signals to each other. The most important characteristic of this system is its capability to learn [17].

This learning process is referred to as the training of the neural network and can have many forms. The most common method is called supervised learning and refers to the training of the neural network through sets of input patterns and matching targets. Successful training will result in a neural network that has the ability to generalize and associate data. It should be able to provide correct results for cases similar to the ones which it was trained on but has not encountered during the training process. This generalization results in high fault tolerance against noisy input data [17].

One of the disadvantages of the neural networks is that it is hard to gain insight into the operations it performs and the knowledge it contains. The only way to transfer knowledge to the network is through the learning process. Therefore, analysis of a neural network's performance can be more challenging compared to conventional algorithms.

The neural network can be seen as a set of neurons which are connected to each other and transmit signals through these connections. The strength of these connections is implemented through weights which can either enhance or inhibit the signal transmission. Similarly to the biological model, the neurons accumulate the weighted inputs, compare the input signal to a threshold and adapt their activation status accordingly. If they are activated, then they provide an output (not necessarily linearly dependent on the input). This process can become clearer by looking at the structure of the neuron (Figure:4).

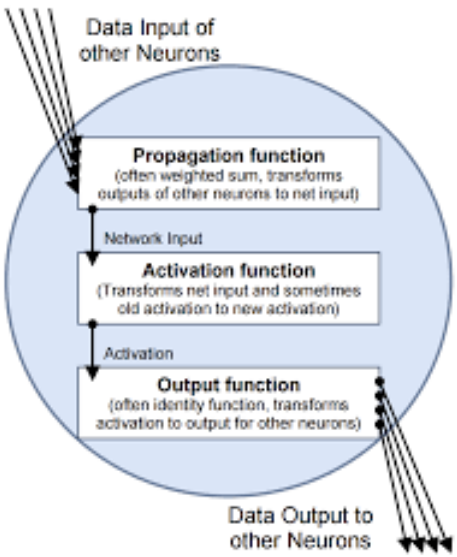



Figure 4: The accumulation and process of the input signal is performed by the propagation function. Then the activation function is applied on this input and the status of the neuron is decided. Based on this status the output function provides the output of the neuron. Image source: [17]

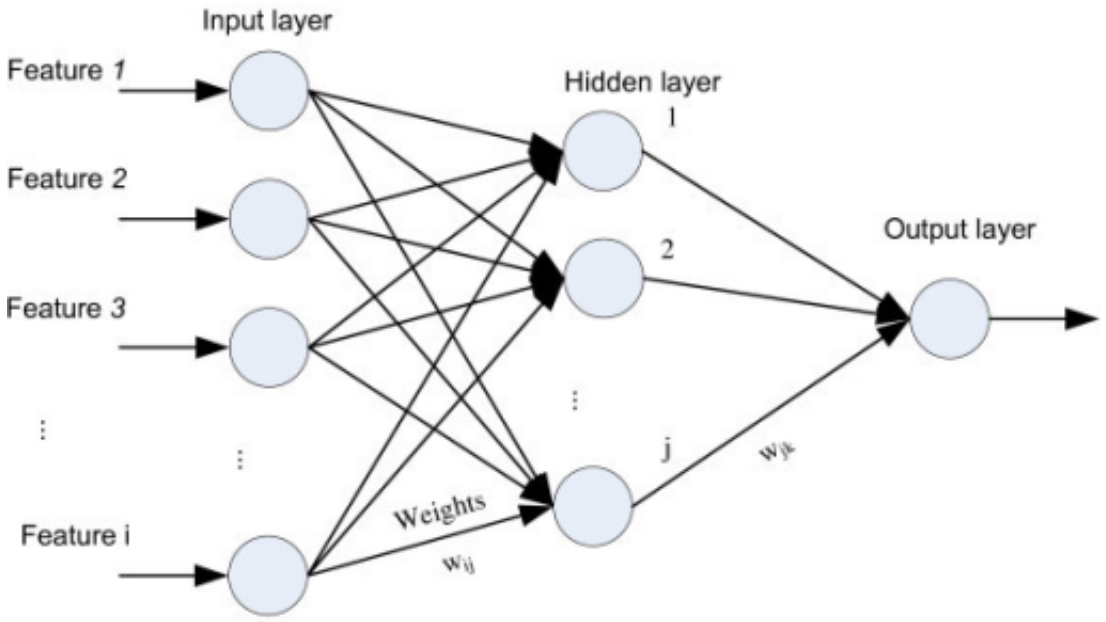



Figure 5: A feedforward network is organized in layers, with distinguishable input and output layers and any number of hidden layers in between. Image source:[22]

## 3.1 Feedforward networks and backpropagation

Based on their structure and their type of connections different type of networks can be defined. One of those types, which is relatively popular, is the *feedforward network*. A feedforward network organizes its neurons in clearly separated layers. It consists of an input and output layer along with a number of layers which are not visible from the outside, called *hidden layers*. The network's name is derived from the fact that connections are only allowed with neurons in the following layers [17].

In this context it is useful to address the training process of the feedforward network. The application of neural networks can be seen as a way of mapping the input features to the output targets. In supervised learning, during its training the network is provided with the input patterns and their corresponding correct results (targets). The process is the following: the input patterns $p_i$ are fed into the network and are propagated through the network by the activation of neurons. The result of the propagation through the output layer is the output of the network (Figure:5). This output is compared with the expected targets and provides an error vector. Based on this error estimation, corrections for the network are calculated and applied. The process is repeated until the error is sufficiently small or a certain number of iterations is reached.

By introducing a bias neuron to facilitate the activation thresholds of neurons it can be shown that the training of the network is realized by proper selection of the connections' weights [17]. The corrections that are introduced to the network's weights are dependent on the error vector of the output. Obviously, the aim of the training process is to minimize the discrepancy between the network's outputs and the expected values, *i.e.* the error function. The search for the error function's minimum is performed by optimizing the gradient descent. From a mathematical point of view, the gradient represents the function's rate of change. A gradient descent optimization algorithm refers to moving against the direction of the gradient. Therefore, the optimization moves towards the lowest values of the function with steps that are comparable to the size of the gradient.

In the above discussion it was assumed that the error function depends only on the weights of the network. In the gradient descent optimization the changes applied to the weights are proportional to the derivative of the error function with respect to the weight that is adjusted. By proper selection of the activation function and by treating the propagation function as simply the weighted sum of the input signals, the backpropagation rule is derived. An exceptional and detailed derivation of the rule can be found in [17]. This rule states that the correction applied to each network weight is proportional to the output of the preceding layer's neuron and the difference of the succeeding neuron's output from the target, if the latter is in the output layer. If the succeeding neuron is not in the output layer then the difference is substituted by the weighted sum of differences of the neurons from the subsequent layers. In terms of an equation the rule is expressed as follows:

$$\Delta w_{k,h} = \eta o_k \delta_h \tag{8}$$

$$\delta_h = \begin{cases} f'_{act}(net_h)(t_h - y_h) & (h \quad outside) \\ f'_{act}(net_h) \sum_{l \in L} (\delta_l w_{h,l}) & (h \quad inside) \end{cases} \tag{9}$$

In the above expression: $\Delta w$ is the weight adjustment, the subscript $k$ denotes the neuron of the preceding layer, $h$ refers to the neuron of the subsequent layer, $o$ is the output of the neuron, $y$ is the output of the network, $t$ is the target or desired outcome, $f_{act}$ is the activation function, $net$ is the weighted sum of the inputs of the neuron. Lastly, $\eta$ refers to the learning rate which is a parameter that determines the speed and accuracy of the learning procedure. As the term suggests, in backpropagation the weights that are processed first are the ones in the last weight layer and the procedure moves backwards from layer to layer while considering each preceding change in weights [17].

## 3.2 Multilayer perceptron

A commonly used feedforward network is the perceptron, originally proposed by Rosenblatt as a single binary classifier [23] but further development made it into a useful neural network with multiple applications. The perceptron contains a retina, a layer that is used only for data acquisition and is connected with the network's input layer with fixed weighted connections. The input layer is followed by at least one trainable weight layer and subsequently by the output layer. One neuron layer is completely linked with the following layer. If there are more than one trainable weight layers, then the perceptron is called multilayer perceptron (MLP). The layers between the input and output layer with the trainable weight connections are what were previously defined as hidden layers [17].

The addition of the hidden layers to the original perceptron concept, which consisted of two layers and one trainable weight layer, made the MLP capable to handle non-linear problems [24] [17]. However the MLP has also disadvantages, which according to [24], are the following:

1. MLPs require tuning a number of hyper-parameters such as the number of layers, neurons and training iterations.

2. MLP is sensitive to feature scaling which is why it is recommended to normalize the input patterns before providing them to the network.

3. It is possible that the error function has more than one local minimum so the achieved accuracy of the network is dependent on the weight initialization.

Throughout this section there have been references to the training set of patterns and their corresponding targets. As it was discussed in section 3.1, supervised learning means that the network receives an input pattern and through multiple iterations tries to reach the desired target which is also provided. The finite number of pairs of input patterns $p_i$ and their corresponding targets $t_i$ constitute the training set. The network can be trained with one pattern at a time which is referred as online training [17]. Alternatively, the network can be trained with sets of pairs $\{p_i, t_i\}$, called *batches*, which is referred as offline training. In the later case, a training *epoch* is concluded when the network is applied to all of the batches that comprise the training set.

## 3.3 Classification and Regression

There are two main categories of problems concerning pattern recognition that neural networks can solve: *classification* and *regression* problems [25].

In classification problems, the network has to assign the input pattern to one of the provided discrete classes or categories [16]. There are cases, defined as binary classification, where the network acts as a Boolean operator. In general there is no limitation to the number of classes that can serve as the output of the network. The latter category of problems is called multi-class classification. There are also some problems when the output can belong to more than one category, these are called multi-label cases [24] [25]. In every classification problem, the network calculates the probability that the provided input belongs to the output classes and (in most cases) assigns the pattern to the class with the highest probability.

In regression problems, the output of the network is a parameter that can receive real values. Networks that perform as regressors, attempt to estimate a value that represents the input feature instead of assigning the input sample to a class [25]. With that definition in mind, classification can be treated as a regression problem with the estimated value being the assignment probability of each class [16]. In the case of multilayer perceptrons, when they act as regressors, the output layer can be treated as lacking an activation function, since the calculated value serves also as the network's output [24].

## 3.4 Performance metrics

The subject of this section are the metrics that can be used to evaluate the performance of neural networks when they act as classifiers and regressors. Since the two types of networks give different types of outputs, the performance metrics are defined to reflect that difference. It is important to note that the performance of the network can be evaluated when it is applied to data that was not included in the training set. These sets of data, often called test sets, are used to determine the ability of the network to generalize its conclusions. There are some cases, when the performance of the network on the training set can provide useful insights but it is a practice generally avoided [25].

As it was mentioned in Section:3.3, in regression problems the networks estimate the value of a parameter and compare it with its true value. Based on this property a straightforward way to evaluate the performance of the network is to look into the relative error of the predicted value of the parameter with respect to its true value:

$$Err_{rel} = \frac{|y_{pred} - y_{true}|}{y_{true}} \tag{10}$$

Another widely used metric that is used in regression problems is the $R^2$ coefficient of determination. It compares the standard deviation of the samples with the discrepancy between the predicted and true values. The best possible score is unity while a model that does not correspond to the input features has a value of zero for this metric. Of course this coefficient can also get negative values because the model can be arbitrarily worse [24]. For a number of samples N, it is defined as follows:

$$R^2 = 1 - \frac{1}{N}\frac{\sum_{i=0}^{N-1}(y_{pred,i} - y_{true,i})^2}{\sum_{i=0}^{N-1}(y_{pred,i} - y_{mean})^2} \tag{11}$$

A commonly used metric, derived from the $R^2$ parameter, is the Fraction of Variance Unexplained (FVU) which corresponds to the calculated fraction in the $R^2$ definition. As a result FVU is defined as: $FVU = 1 - R^2$ [4].

For classification problems, in order to evaluate the performance of the network we need to examine whether the input patterns were assigned to the correct class. In this context, the most straightforward method for performance evaluation is the confusion matrix. The columns of this matrix refer to the predicted conditions while the rows refer to the true conditions or vice versa. Ideally the matrix should have elements only on its diagonal so that predicted and true values would coincide for every sample of the test set [26].

Even though the confusion matrix is a great tool for visualization it is not convenient for comparing the performance of different networks. The equivalent of the $R^2$ parameter of regression for classification is the mean accuracy which is defined as follows:

$$Accuracy = \frac{1}{N}\sum_{i=0}^{N-1} I(y_{pred,i} = y_{true,i}) \tag{12}$$

,where $I(y_{pred,i} = y_{true,i})$ is the indicator function that defines whether an element belongs to a specific set.

Although the mean accuracy score is a useful metric for the overall performance of the classifier it can be misleading [25] [27]. For binary classification, some widely used metrics that aim to provide a more clear picture are the following (Figure:6):

1. Precision is the fraction of the true positives over the total number of cases that were identified as positive. $Precision = \frac{t_p}{t_p + f_p}$
2. Recall is the fraction of the true positives over the sum of true positives and false negatives. $Recall = \frac{t_p}{t_p + f_n}$ [28]
3. F1-score is a metric that combines the previous two metrics and is defined as: $F1 - score = \frac{Recall \quad Precision}{Recall + Precision}$

In the above notation $t_p$ and $f_p$ correspond to true and false positives each while $t_n$ and $f_n$ represent true and false negatives respectively.

Lastly, a binary classification metric similar to the $R^2$ coefficient is Matthew's Correlation Coefficient (MCC). The coefficient values can range from -1 to +1 with the first corresponding to complete disagreement between prediction and correct values and the latter to a perfect agreement. If the coefficient has a value of zero then the network behaves as a random estimator. MCC can be calculated straight from the confusion matrix [30] and is defined with the following expression where the notation is the same as above:

$$MCC = \frac{t_p * t_n - f_p * f_n}{\sqrt{(t_p + f_p)(t_p + f_n)(t_n + f_n)(t_n + f_p)}} \tag{13}$$

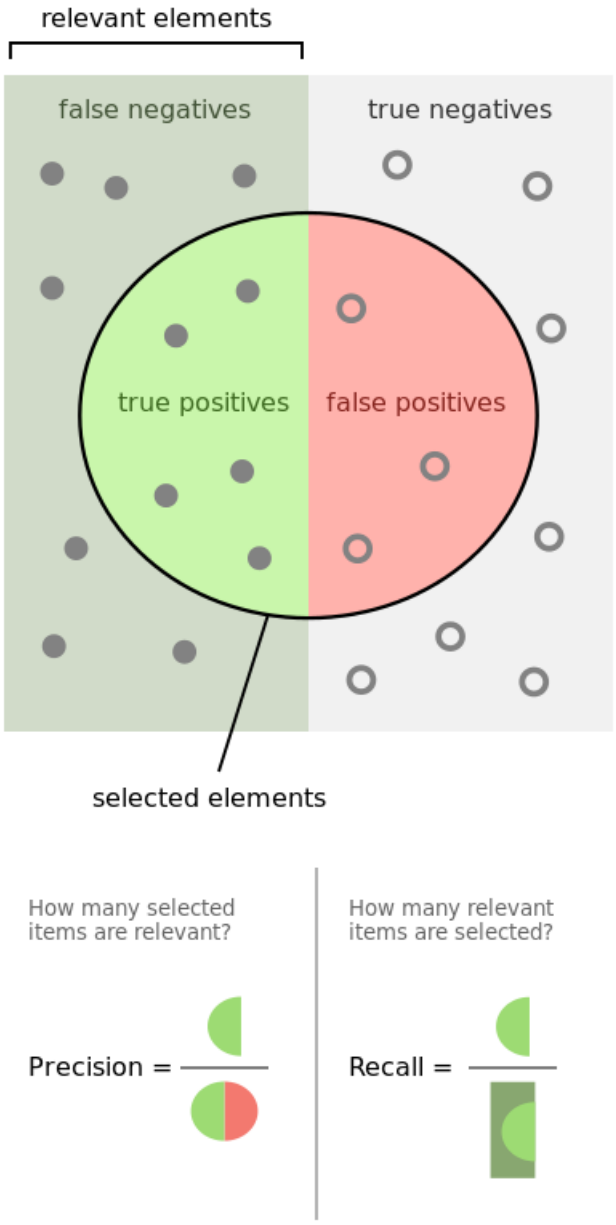



Figure 6: Visualization of Precision and Recall metrics. Precision is the probability that a selected sample is relevant to the category while Recall shows the probability that a relevant sample will be picked up by the network. Image source:[29]

MCC is regarded as a better representation of the network's performance compared to the metrics like Precision and Recall because it is independent of the classes' relative size [27].

# Part II
# Code Development

## 4 Forward Model

This chapter is devoted to the description of the forward model code we developed during this project for the simulation of the CTS spectra. In our research, two approaches were followed: the first is an approximation of an arbitrary velocity distribution for the fast ions through a series of orthogonal polynomials while the second assumes a slowing-down distribution for the fast ion population and Maxwellian distributions for the electrons and bulk ions. Both of these approaches are developed in the electrostatic approximation (so there is no coupling between the electric and magnetic field). Our model calculates only the spectral density function ($S(\mathbf{k}, \omega)$) of the spectrum. This means that the polarization coupling of the scattering beams, the beam overlap and ray tracing considerations are not included in our code. All of these factors need to be included for comparison with experimental CTS spectra. Assumptions regarding the magnetized or unmagnetized nature of the plasma populations are addressed in the analysis of each approach.

The polynomial approximation is addressed in Section:4.1, while the slowing-down case and its related equations are discussed in Sections 4.2 and 4.3. Section:4.4 addresses the input parameters of our model and mainly the two configurations that were investigated in this project: configuration A, which was designed to simulate the situation in the W 7-X experiment and configuration B: which refers to a scattering geometry we defined in this research inspired by literature papers on fast ion investigations. In the same section, a spectrum for each configuration is presented and the dependency of the simulated spectra on our model parameters is discussed. Lastly, Section:4.5 briefly presents the choices that we made for the development of a database using the slowing-down model for each configuration. These databases were later used for the training of the neural networks.

The benchmark tests for each approach were spectra from literature references. More specifically, the code for the polynomial approximation was validated with respect to the results in [6], while the slowing-down model was compared with the spectra presented in [19] [20]. Our forward model was developed in Python and the source codes are documented and available in Appendix section:12 of this thesis.

### 4.1 Spectral density calculation for arbitrary functions

As it was mentioned in Section:2 there are multiple sources of fast ions. Analytic expressions for their contribution to the spectral density function is not always available. It is also possible that their velocity distribution can only be expressed numerically. An approach to overcome this difficulty while treating the plasma as fully magnetized is to approximate the velocity distribution of the species that needs to be simulated with a series of orthogonal polynomials [6]. This approximation is expressed with Equation (14), where $\alpha_{mn}$ is a coefficient calculated by Equations (15-16), $L_m$ are the Laguerre polynomials and $H_n$ are the Hermite polynomials. The variables $x$ and $y$ can either refer to the velocity components parallel and perpendicular to the magnetic field or the energy and pitch angle of the particle [13].

$$f(x, y) = \sum_m \sum_n \alpha_{mn} L_m(y) H_n(x) e^{-y} e^{-x^2/2} \tag{14}$$

The velocity distribution that is approximated is encapsulated by the set of coefficients $\alpha_{mn}$. The rest of the model is mostly dependent on the plasma parameters and the simulated experimental choices.

$$\alpha_{mn} = \int_0^\infty g_n(y) L_m(y) e^{-y/2} \tag{15}$$

$$g_n(y) = \frac{1}{2^n n! \sqrt{\pi}} \int_{-\infty}^{\infty} f(x,y) H_n(x) e^{-x^2/2} \tag{16}$$

The first step is to calculate the susceptibility of the different populations and by extent the dielectric function of the plasma. These terms are each calculated with the Equations (17) and (18).

$$\begin{aligned} H_j(\mathbf{k},\omega) = \frac{2\pi^{3/2}\omega_{pj}^2}{k^2\sigma_\perp^2} \sum_m \sum_n \sigma_\perp^2 \sigma_{||} a_{mn} \\ \times \sum_l \{ \frac{\sigma_\perp^2}{\sigma_{||}^2} \mathcal{L}(l,m) [\frac{n!}{2(n/2)!} + \frac{\zeta_l}{2} Z(\zeta_l, n) - \frac{\sqrt{2}n}{2} Z(\zeta_l, n-1)] \\ + [(\zeta_0 - \zeta_l) Z(\zeta_l, n) (\frac{1}{2}\mathcal{L}(l,m) - \mathcal{L}'(l,m))] \} \end{aligned} \tag{17}$$

$$\epsilon_L = 1 + H_e(\mathbf{k},\omega) + H_i(\mathbf{k},\omega) + G_a(\mathbf{k},\omega) \tag{18}$$

In order to calculate the susceptibility function, the integrals defined with the Equations (19-21) have to be calculated by making use of the parameters presented in Table:1.

Table 1: Normalized variables and parameters required to calculate the integrals given by Equations (19- 21). In the above notation, $\sigma = \sqrt{\frac{2T}{m}}$ for the parallel($||$) and perpendicular direction ($\perp$).

| | |
|---|---|
| $t \equiv \frac{\sqrt{2}v_{||}}{\sigma_{||}}$ | $y \equiv \frac{2v_\perp^2}{\sigma_\perp^2}$ |
| $\beta_j \equiv \frac{k_\perp \sigma_{\perp j}}{\sqrt{2}\Omega_j}$ | $\zeta_l \equiv \frac{\omega - l\Omega_j}{k_{||}\sigma_{||} j}$ |

$$\mathcal{L}(l,m) = \int_0^\infty dy e^{-y/2} J_l^2(\beta\sqrt{y}) L_m(y) \tag{19}$$

$$\mathcal{L}'(l,m) = \int_0^\infty dy e^{-y/2} J_l^2(\beta\sqrt{y}) L'_m(y) \tag{20}$$

$$Z(\zeta_l, n) = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} dt \frac{e^{-t^2} H_n(\sqrt{2}t)}{t - \zeta_l} \tag{21}$$

Once the susceptibility is calculated, then the contributions of the different populations to the spectral density function are given by Equations (22) and (23) for the electrons and the ion species respectively.

$$S_e(\mathbf{k},\omega) = \pi^2|1 - \frac{H_e}{\epsilon_L}|^2 \sum_l \sum_m \sum_n \sigma_{e\perp}^2 \sigma_{||e}(a_m n)_e \frac{H_n(\sqrt{2}\zeta_l)e^{-\zeta_l^2}}{k_{||}\sigma_{||e}} \mathcal{L}(l,m) \tag{22}$$

$$S_j(\mathbf{k},\omega) = \pi^2 Z_{eff}|\frac{H_e}{\epsilon_L}|^2 \sum_j \sum_l \sum_m \sum_n \sigma_{j\perp}^2 \sigma_{||j}(a_m n)_j \frac{H_n(\sqrt{2}\zeta_l)e^{-\zeta_l^2}}{k_{||}\sigma_{||j}} \mathcal{L}(l,m) \tag{23}$$

According to this approximation, to simulate the spectral density function for a Maxwellian distribution, only the coefficient $\alpha_{00}$ is necessary since all the higher order coefficients will be zero [6]. This was also a benchmark test that was used to validate the code developed during this thesis project. Unfortunately, these attempts were not fruitful and the approach was abandoned in favor of the slowing down model (Section: 4.2) due to lack of time. Additionally, the mathematical formulation of the model is quite involved and too complicated to allow a straightforward detection of the error source.

## 4.2 Slowing down distribution

The forward model that we developed in this project includes three plasma populations: the electrons, the bulk ions and the fast ions. The first two were assumed to have a Maxwellian distribution. The fast ions were assumed to have a slowing-down velocity distribution given by Equation (24).

The alpha particles created in a fusion plasma are expected to have such a distribution, where they gradually lose their birth energy $E_a = 3.5$ MeV through collisions. The distribution was originally calculated for the case of NBI injected ions in a plasma of Maxwellian electron and bulk ion populations [31]. The expression for the slowing-down velocity distribution is:

$$f_a(\mathbf{v}) = \begin{cases} 0 & , v > v_a \\ \\ \frac{F_0}{v^3+v_c^3} & , v < v_a \end{cases} \tag{24}$$

,where $F_0 = \frac{3}{4*\pi \ln(1+v_a^3/v_c^3)}$, $v_a = \sqrt{\frac{2E_a}{m_a}}$ and $v_c \simeq 0.09 v_e$. From this expression, the one-dimensional form of the distribution along the direction of $\mathbf{k}$ (Equation (25)) can be derived [20].

$$f_a(u) = -\frac{2\pi F_0}{3v_c}(0.5 \ln(\frac{(y+v_c)^2}{y^2 - v_c y + v_c^2}) - \sqrt{3}\arctan(\frac{2y - v_c}{\sqrt{3}v_c}))|_{y=u}^{y=\sqrt{u^2+v_a^2}} \tag{25}$$

## 4.3 Spectral density functions

The first step towards the calculation of the spectral density function is the calculation of the shielding electron response as expressed through the species' susceptibilities and the dielectric function. The ions (both bulk and fast) are considered unmagnetized in the sense that they do not complete a cyclotron motion within the scattering volume [19] while the electron population is treated as magnetized. In this case the susceptibilities of the species are given by Equations (26-28) [20] [19]. The integral calculation of the last term is included in the appendix of this thesis (Appendix section:10).

$$H_e(\mathbf{k},\omega) = \alpha^2 \sum_{l=-\infty}^{l=+\infty} \exp(-k_\perp^2\rho_e^2) I_l(k_\perp^2\rho_e^2)(1 + \frac{\omega}{k_{||}v_e} Z(\frac{\omega - l\Omega_e}{k_{||}v_e})) \tag{26}$$

$$H_i(\mathbf{k},\omega) = \alpha^2 \frac{Z_i^2 n_i T_e}{n_e T_i} \sum_{l=-\infty}^{l=+\infty} \exp\left(-k_\perp^2 \rho_i^2\right) I_l(k_\perp^2 \rho_i^2)\left(1 + \frac{\omega}{k_{||} v_i} Z\left(\frac{\omega - l\Omega_i}{k_{||} v_i}\right)\right) \tag{27}$$

$$G_a(\mathbf{k},\omega) = \left(\frac{\omega_{pa}}{k}\right)^2 \int d^3v \mathbf{k} \frac{\partial f_a(\mathbf{v})}{\partial \mathbf{v}} \frac{1}{\omega - \mathbf{kv} + i\delta} \tag{28}$$

In the notation above, $T$ refers to the species' temperature, $n$ is the species' density, $\alpha$ is the Salpeter parameter (Equation (4)), $v_e$ denotes the electron thermal velocity, $v_i$ is the thermal bulk ion velocity, $\rho$ is the gyroradius, $I_l$ are the modified Bessel functions of the first kind, $\Omega$ is the cyclotron frequency, $Z_{i,a}$ is the species electric charge, $Z(\mathbf{k},\omega)$ is the plasma dispersion function (implemented through PlasmaPy [32]), $\perp$ and $||$ refer to directions perpendicular and parallel to the magnetic field respectively.

Under these assumptions, the analytical expression for the contribution of each population to the spectral density function can be calculated: the electron contribution (Equation (29)), the bulk ion contribution (Equation (30)) and the fast ion contribution (Equation (31)) [20] [19].

$$S_e(\mathbf{k},\omega) = |1 - H_e/\epsilon_L|^2 \frac{2\sqrt{\pi}}{|k_{||}| v_e} \times \sum_{l=-\infty}^{l=+\infty} \exp\left(-k_\perp^2 \rho_e^2\right) I_l(k_\perp^2 \rho_e^2) \exp\left(-\frac{(\omega - l\Omega_e)^2}{k_{||}^2 v_e^2}\right) \tag{29}$$

$$S_i(\mathbf{k},\omega) = |H_e/\epsilon_L|^2 \frac{2\sqrt{\pi} Z_i^2 n_i}{|k_{||}| n_e v_i} \times \sum_{l=-\infty}^{l=+\infty} \exp\left(-k_\perp^2 \rho_i^2\right) I_l(k_\perp^2 \rho_i^2) \exp\left(-\frac{(\omega - l\Omega_i)^2}{k_{||}^2 v_i^2}\right) \tag{30}$$

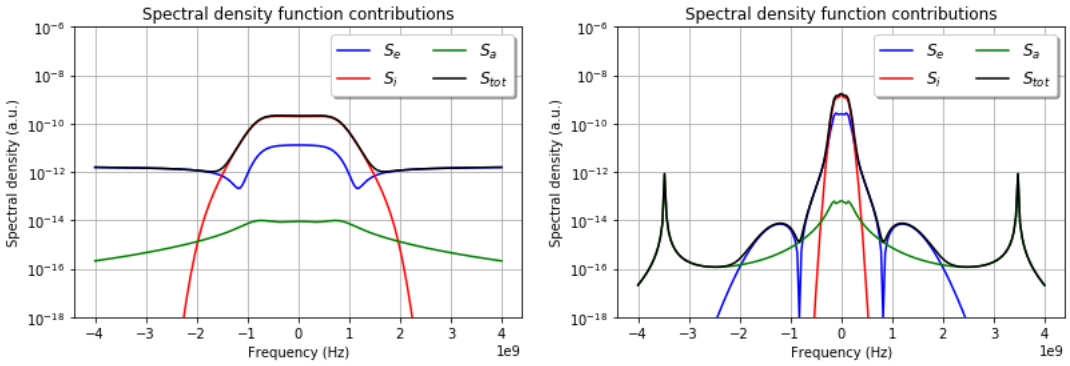
$$S_a(\mathbf{k},\omega) = |H_e/\epsilon_L|^2 \frac{2\pi Z_a^2 n_a}{n_e |\mathbf{k}|} f_a(|\omega|/|\mathbf{k}|) \tag{31}$$

Obviously, the total spectral density function in the CTS spectrum is equal to the sum of the individual contributions.

$$S_{tot}(\mathbf{k},\omega) = S_e(\mathbf{k},\omega) + S_i(\mathbf{k},\omega) + S_a(\mathbf{k},\omega) \tag{32}$$

## 4.4 Simulation results

The forward model that we developed in this project simulated the CTS spectral density function for a Deuterium plasma. According to the previous section, the populations that contribute to the simulated spectrum are: the electrons and bulk ions which have a Maxwellian distribution and the fast ions produced by NBI which have a slowing-down distribution. The probing frequency was set to 140 GHz and the observable bandwidth is 136 to 144 GHz. The simulated spectra were benchmarked through comparison with the spectra presented in references [19] and [20]. Initially the model simulated a spectrum in the Wendelstein 7-X configuration in terms of magnetic field, scattering geometry and plasma parameters (geometry configuration A). As it can be seen in Figure:7a, the fast ion contribution cannot be distinguished easily due to its overlap with the electron contribution. For this purpose a different scattering geometry was defined, inspired by the work on fast ions presented in [19] and [20] (geometry configuration B). In this new geometry, the fast ion contribution is a lot stronger in the region outside the probing frequency. Additionally, there is a clear resonance in the spectrum which corresponds to the lower hybrid frequency of the plasma [9] [19]. The form of the spectrum in configuration B (Figure:7b) could prove more suitable for fast ion detection due to the strong presence of that population.




(a) Geometry configuration A (b) Geometry configuration B

Figure 7: The total spectral density function (black) and the contributions by the plasma populations: electrons (blue), bulk ions (red) and the fast ion population (green). The synthetic spectrum on the left is calculated for the W 7-X configuration, where the fast ion presence is weak. The spectrum on the right demonstrates a stronger contribution by the fast ions which is the reason for its investigation.

The main difference between the two configurations lies in the angle $\theta$ between the incident $k_i$ and the scattered $k_s$ beam as it can be seen in Table:2.

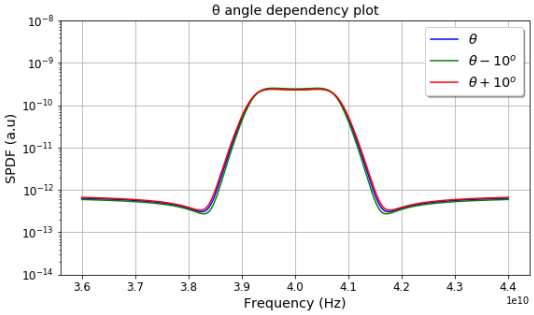
Table 2: In order to investigate the dependency of the simulated spectra on the input parameters, one of them was altered each time while the rest maintained the values presented in this table. The main difference between the two geometries is the scattering angle. Although different values of the magnetic field were used, the influence of this parameter on the spectrum is not significant.

| Geometry configuration A (W 7-X) | Geometry configuration B |
|---|---|
| B = 2.2T | B = 3.5 T |
| $T_e = 5.00$ keV | $T_e = 5.00$ keV |
| $T_i = 5.00$ keV | $T_i = 5.00$ keV |
| $E_a = 45$ keV | $E_a = 45$ keV |
| $n_i = 5.00 \times 10^{19} m^{-3}$ | $n_i = 5.00 \times 10^{19} m^{-3}$ |
| $n_a = 1.5 \times 10^{15} m^{-3}$ | $n_a = 1.5 \times 10^{15} m^{-3}$ |
| $\phi = 110^o$ | $\phi = 105^o$ |
| $\theta = 150^o$ | $\theta = 20^o$ |

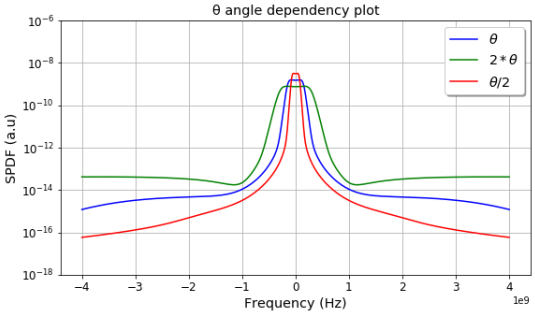
The simulation code uses as input parameters the following parameters:

1. Electron and bulk ion temperature: $T_e$ and $T_i$
2. Bulk and fast ion densities: $n_i$ and $n_a$
3. Fast ion energy: $E_a$
4. The magnetic field B.
5. The angle $\theta$ and the scattering angle $\phi$ between $\mathbf{k}$ and the magnetic field.

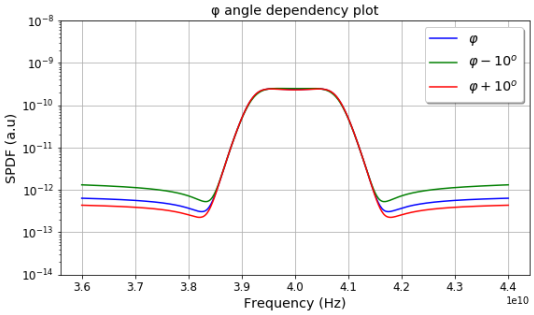
The density of the electron population is calculated through the quasi-neutrality condition of the plasma: $n_e = Z_i n_i + Z_a n_a$. An investigation of the dependency of the simulated spectrum on the input parameters was carried out in both geometry configurations (Figure:8 and Figure:9). For this investigation, one variable was altered each time while the rest were assigned the values presented in Table:2.



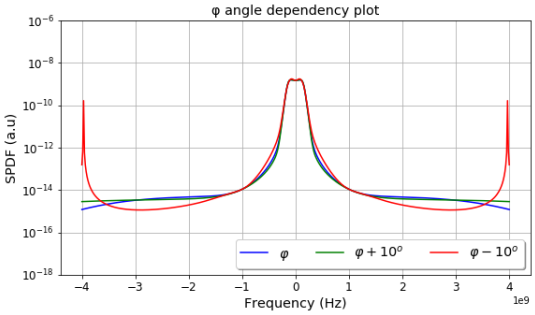

(a) Dependency on $\theta$ angle in geometry A.



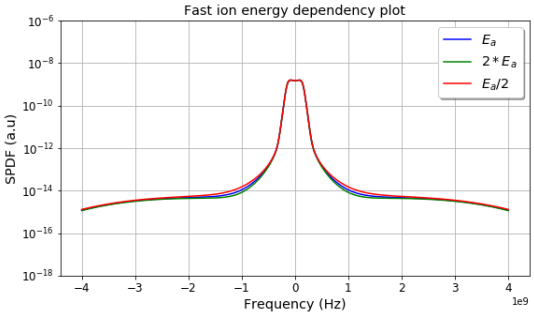

(b) Dependency on $\theta$ angle in geometry B.



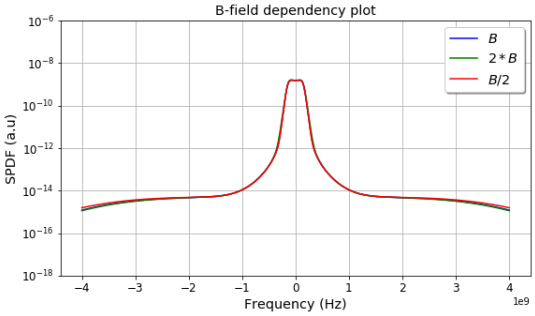

(c) Dependency on $\phi$ angle in geometry A.




(d) Dependency on $\phi$ angle in geometry B.

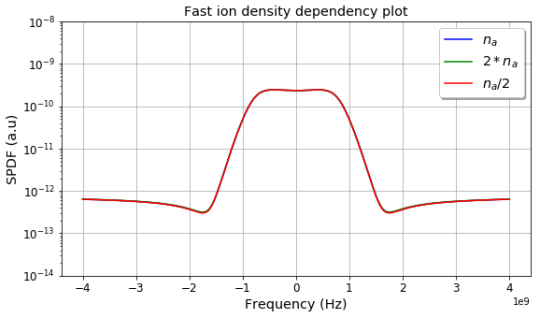



(e) Dependency on fast ion energy in geometry A.

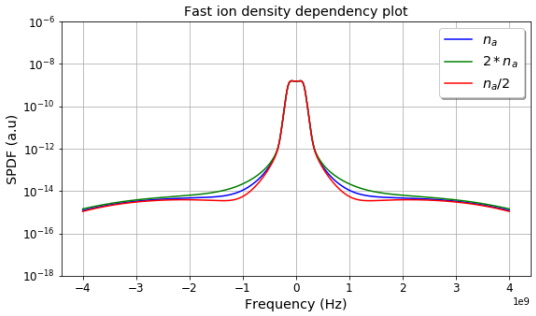



(f) Dependency on B-field in geometry A.

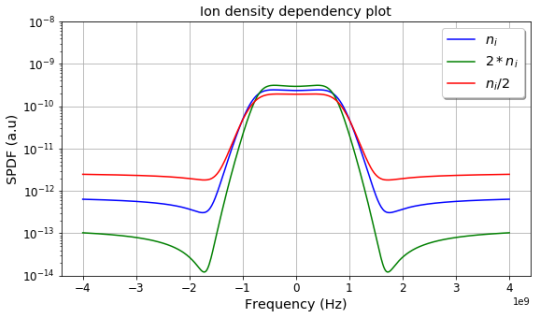
Figure 8: Dependency plots of the model for the two scattering angles in configurations A (left) and B (right). The dependency of the model on the fast ion energy (bottom left) and the magnetic field (bottom right) is barely noticeable. On the contrary, the scattering angle $\theta$ has a big influence on the simulated spectrum in geometry B (top left).



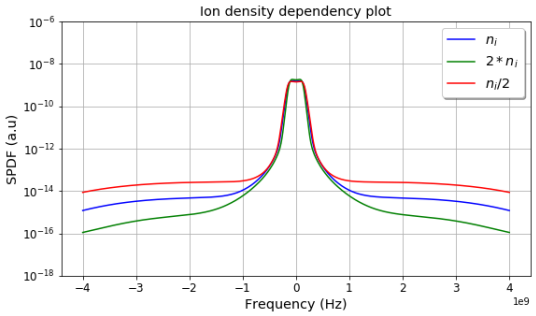

(a) Dependency on fast ion density in geometry A.



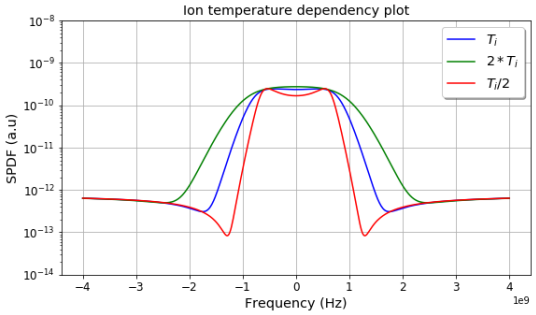

(b) Dependency on fast ion density in geometry B.



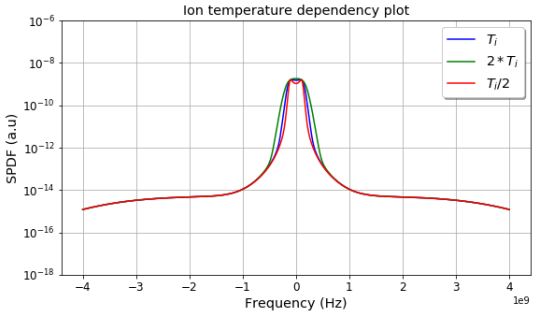

(c) Dependency on bulk ion density in geometry A.



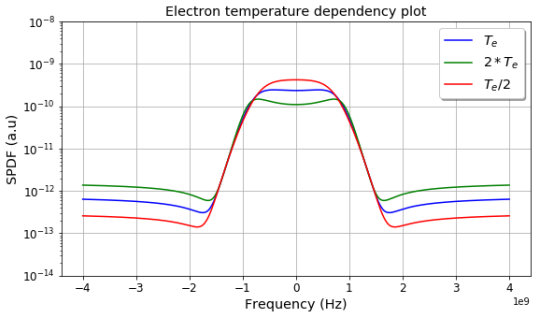

(d) Dependency on bulk ion density in geometry B.



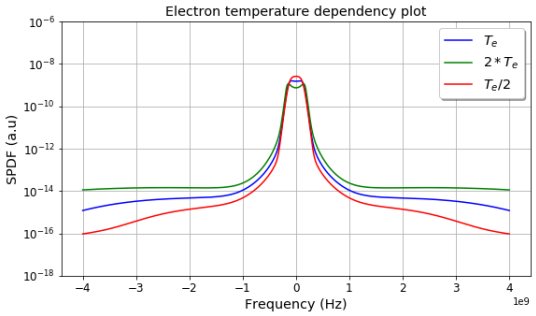

(e) Dependency on bulk ion temperature in geometry A.




(f) Dependency on bulk ion temperature in geometry B.




(g) Dependency on electron temperature in geometry A.




(h) Dependency on electron temperature in geometry B.

Figure 9: Dependency plots of the model for the plasma parameters in geometry A (left) and geometry B (right). The small effect of the fast ion density in the W-7 X configuration (top left) reinforces the motivation to look for a geometry where the fast ion presence can be detected more easily.

The dependency plots behave similarly for both configurations for all the investigated parameters. From Figure:9, it is evident that the influence of fast ion density is not distinguishable for the geometry configuration A (W 7-X inspired). The dependency plots of the spectra on the bulk populations densities and temperatures are in agreement with previous research [18] [14]. Additionally, the fast ion energy does not have a big impact on the form of the spectra. On the other hand, there is a strong dependence on the scattering geometry which makes the latter a very important knob in the search for a configuration where fast ion detection is possible.

## 4.5 Database development

The forward model introduced in the above sections was used to produce databases of synthetic CTS spectra for the training set of the neural networks. The parameter spaces that were explored with these databases are summarized in Tables 3 4. Although the influence of the fast ion energy on the simulated spectrum may be weak, it was included in the set of modelling parameters because it is characteristic of the fast ions as a plasma population. A database of 25000 spectra was developed for each geometry configuration. Two spectral densities were considered: $0.0625MHz^{-1}$(500 sample frequencies) and $0.025MHz^{-1}$ (200 sample frequencies). The values of the input parameters for each spectrum were randomly drawn from a uniform distribution and then adjusted to their respected parameter range. Since the fast ion density can serve as an indication of fast ion presence, a fraction of the spectra was chosen to have this parameter set to zero. The reason for that choice is that the network should be trained on examples that include or lack fast ion presence. In the databases developed for the regression problem this fraction corresponds to 0.1 while for the classification case the fraction was set to 0.5 in order to have similarly sized classes.

Table 3: This table combined with Table:4 represent the parameter space that was explored with the developed databases.

| Parameter | Parameter range |
|---|---|
| Electron temperature $T_e$ [keV] | [1, 10] |
| Ion temperature $T_i$ [keV] | [1, 10] |
| Fast ion energy $E_a$ [keV] | [40, 50] |
| Bulk ion density $n_i[\times 10^{19} m^{-3}]$ | [1,10] |
| Fast ion density $n_a[\times 10^{15} m^{-3}]$ | [0, 3] |

Table 4: This table depicts the two geometry configurations that were investigated, the one applied at the Wendelstein 7-X campaign (left) and the one that was explored in this thesis (right).

| Geometry configuration A | Geometry configuration B |
|---|---|
| $100^o < \phi < 120^o$ | $95^o < \phi < 115^o$ |
| $140^o < \theta < 160^o$ | $10^o < \theta < 30^o$ |

# 5 Neural Network Implementation

This chapter is devoted to the neural networks that we used for fast ion detection in CTS spectra. Neural networks are well established mathematical models and have been implemented as part of computational libraries and modules. In the cases of their application in research for data analysis, researchers adjust the parameters of the models included in those libraries to fit their purposes. In our case, we use a machine learning library for Python: scikit-learn [24]. This library contains many machine learning algorithms, including neural networks, and tools for data processing and evaluation of these algorithms. For our case, we used the multilayer perceptron models (MLP) and some of the available evaluation metrics for the networks' performance. During the project, we used regressive neural networks to estimate a parameter that is indicative of fast ion presence (Section:5.1). Also we applied classification networks to distinguish between spectra that include or lack fast ions (Section:5.2). For each approach, we present the parameters of the networks (input and output parameters, architecture and model parameters as defined in [24]) and graphs that demonstrate their performance on the training and test sets. These preliminary results are presented here to provide a clearer picture of the situation. The source codes that we used for the training of the networks and their evaluation were based on previous work [4] and they are documented and available at the Appendix section:13 of this thesis.

## 5.1 MLP Regressors

### 5.1.1 Regressor input and output

In regression problems, the network estimates parameters that can take values from a real continuous range. These parameters define the features of its input patterns. MLP regressors implemented with the scikit-learn library have the option to infer multiple parameters. Each calculated parameter corresponds to a node in the network's output layer. Therefore the nature of the problem partly defines the architecture of the regressor. Although the neural network provides estimations for all seven parameters, the parameter that was chosen as indicative of fast ions is the species' density [10]. The MLP regressors received as input the CTS spectra, i.e. the spectral density function calculated in each sampling frequency of the spectrum, and estimated the value of the parameters that were used to produce the spectrum (Tables 3 and 4).

Before their introduction to the input layer the CTS spectra were processed. Since the neural networks are sensitive to feature scaling [17], the spectra were normalized to unit variance and centered by removing the mean, using the standard scaler method of the scikit library [24]. This is a procedure that is performed in most machine learning approaches and is considered as good practice. Furthermore, one of this project's aims is to take advantage of the high fault tolerance of neural networks and examine noisy CTS signals (Section:6.2.2). For this reason, a noise signal of Gaussian origin was introduced to the synthetic spectra. The amplitude of the noise signal was defined as a fraction of the mean of the spectrum's maximum value for all the spectra included in the training set.

### 5.1.2 Regressor architecture

One of the first choices in the neural network design is the definition of the hyper-parameters related to its architecture. This corresponds to the number of hidden layers that should be used and the number of neurons in each hidden layer. Both of these choices are highly dependent on the type of problem that is addressed by the neural network. As a result, the choice of these

parameters is mostly based on experience and intuition. The decision for the networks implemented in this project was based on a set of guidelines that are considered good practice in the general field of machine learning [33], [25] and preceding related research [4]. The developed networks had two or three hidden layers while different configurations for the number of neurons in each hidden layer were investigated. A rule, applied in all the examined cases, was that the number of neurons in each layer was decreasing from the input towards the output layer. The influence of the architecture on the performance of the neural network was investigated as part of this thesis project (Section:6.2.2). The notation that defines the regressor architecture is the following: $LN_1LN_2$ where each $L$ corresponds to a hidden layer and $N_1$, $N_2$ represent the number of neurons in each hidden layer as information propagates from the input to the output layer.

### 5.1.3 Regressor parameters

By looking at the schematic of a neural network (Figure:5), the choices for its architecture are the first ones to be addressed. However, in order to complete the network model, further choices regarding its parameters were made: some of them are related to the neuron characteristics (Figure:4), some are related to the optimization algorithm that is used and some are referring to its training process. The neural networks of this project were implemented with the scikit-learn library so this section addresses the choices regarding the parameters of that model. A complete list of the parameters can be found in the Appendix Section 11. The choices for the network parameters of this project are the following:

1. *activation*: The activation function that was used for the neurons of the hidden layers is the logistic or Fermi function, which maps the output in the range (0,1) [17]:

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{33}$$

   This function is widely used in machine learning due to its convenient derivative that can simplify the gradient optimization algorithms.

2. *solver*: The chosen weight optimization solver was Adam, a stochastic gradient-based optimizer [34]. This solver has demonstrated good performance in terms of training time and validation score for datasets containing thousands of samples [24].

3. *alpha*: The regularization parameter $\alpha$ was set to $5 \times 10^{-4}$ based on suggestions from [33] [25].

4. *batch_size*: The batch size corresponds to the number of samples that propagates through the network at each iteration. It was set to 250 samples. This choice is supported by [4], however different sizes were tested. Larger batches (500 samples) result in worse performance while smaller (100 samples) lead to very slight improvement.

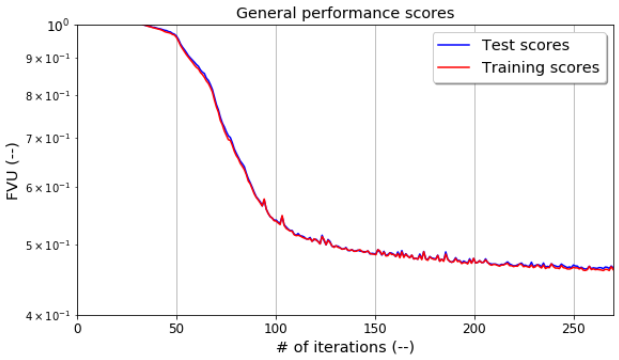
5. *warm_start*: This parameter was set to True so that the calculated weights after each iteration were used as initialization values for the next.

6. *max_iter*: The number of training epochs was implemented in the code explicitly so this parameter was set to one.

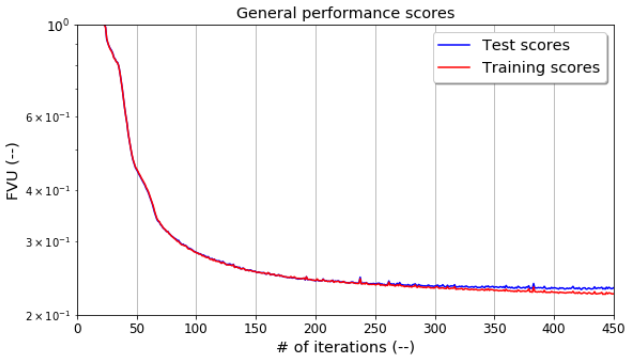
For the rest of the parameters the default values were used, as presented in the respective Appendix section (Table:11).

### 5.1.4 Regressor training performance

The database that was developed in each parameter space, was split into a training set and a test set. The first one corresponds to the 90% of the total database and is used in the weight optimization of the neural network. Each sample of the training set is comprised of the synthetic spectrum and the modelling parameters that were used to simulate that spectrum. These parameters serve as the targets for the network's fitting. After each iteration, the performance of the network is monitored through the calculation of the Fraction of Variance Unexplained, which should decrease as training progresses. Additionally, the ability of the network to generalize its results is evaluated through its application on samples of the test set. The latter corresponds to the remaining 10% of the database and contains samples that the network did not encounter during its training.




(a) Neural network performance on training (red) and test (blue) sets in geometry configuration A.




(b) Neural network performance on training (red) and test (blue) sets in the geometry configuration B.

Figure 10: These graphs are based on the performance of the network L150L100L50 on noiseless spectra calculated in the two parameter spaces. Clearly, the FVU in the configuration B is lower compared to the one in the W-7 X case which means that the overall performance is better. Also the curves in parameter space B are smoother compared to A.

If the neural network is performing well, the FVU for the test set should follow the decrease of the respective parameter for the training set. For noiseless data, this appears to be the case in both parameter spaces (Figure:10). In geometry configuration A, the FVU converges to a value around 0.48 while in geometry configuration B the same metric reaches a value of 0.24. FVU is an indication of the regressor's overall performance and not its particular ability to detect fast ions. Therefore in terms of general performance, the operation in geometry configuration B is preferable.

### 5.1.5 Regressor's performance on test set

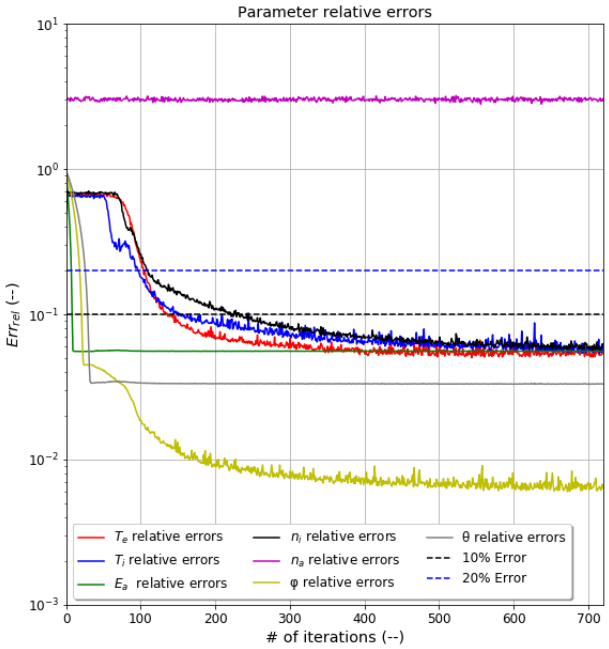
A metric that can provide insight on the accuracy of the regressor estimated values for each parameter is the evolution of the relative error for each parameter $Err_{rel}$ as training progresses. According to Figure:11a, the regressor is not able to infer the fast ion density in geometry configuration A. On the other hand, the relative error of the inferred fast ion density in parameter space B is improving with network training but is still significantly high (Figure:11b). For evaluation purposes, we define as sufficient inference, relative errors of 20% and as very good inference, relative errors of 10%. The fast ion density inference exceeds both these levels. With resperct to the bulk population temperatures, the performance of the network is better in geometry configuration

A. The bulk ion density inference is slightly better in the same configuration but in both cases it is below the acceptance level. The inference of the fast ion energy does not improve with training and is the same for both configurations.

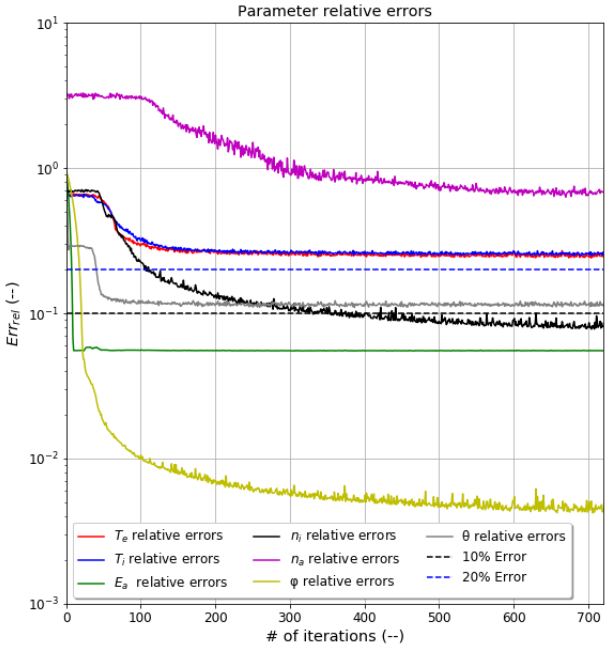
Regarding the angles that define the experimental geometry, the inference of the angle $\theta$ shows little improvement through training for both configurations. However, the calculated relative error for this parameter is lower in the W 7-X configuration. Lastly, the ability of the network to infer the observation angle $\phi$ is very good in both configurations and the corresponding relative error is the lowest among the inferred parameters.

The relative error for the inferred parameters is a very good metric to gain insight on the performance of the network for each estimated parameter. A clearer and more straightforward conclusion can be drawn by plotting the inferred against the true values of the parameters. If the inference of the regressor was perfect then the inferred values in this plot would land on the identity line. In this thesis, we will refer to these plots as prediction plots. The prediction plots for each of the inferred parameters in both parameter spaces are presented in Figures 12 and 13.

The prediction plots confirm the observations based on the study of the relative errors for most of the parameters. There are two features that should be addressed. In parameter space B the network is able to recognize the difference in the spectrum for different fast ion densities. Although the inference of the parameter is not sufficiently accurate, the network could perhaps distinguish between cases where the fast ion density is zero and non-zero. Towards this aim, a database with predefined values for the fast ion density in the same parameter range was developed. As it can be seen in Figure:14, the deviation of the inferred values is in the same order of magnitude for zero and non-zero densities. This constitutes the order of the deviation an unreliable criterion for the fast ion presence.

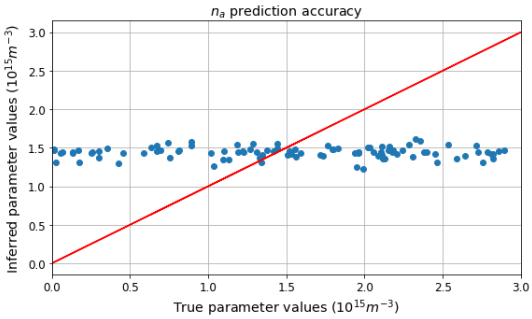



(a) Overview of the relative errors for the estimated parameters in parameter space A.

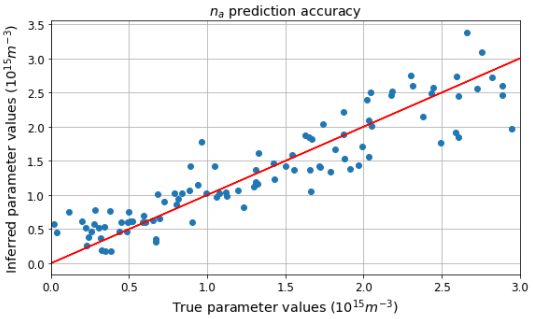



(b) Overview of the relative errors for the estimated parameters in parameter space B.

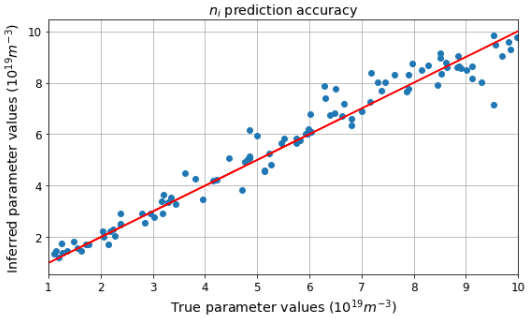
Figure 11: These graphs are based on the performance of the regressor L150L100L50 on noiseless spectra calculated in the two parameter spaces. The blue dash line (20% error) represents inference which could be considered sufficient. The black dash line (10% error) represents inference which is considered very good. The relative errors for the fast ion density (purple) in the parameter space A is very high and does not improve as the training progresses. In the parameter space B, the relative error remains high but the training procedure has a positive influence. On the other hand the relative errors for the rest of the parameters are better in the geometry configuration A.



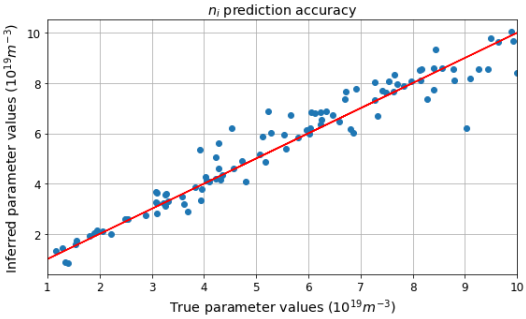

(a) Prediction plots for the fast ion density in geometry A.



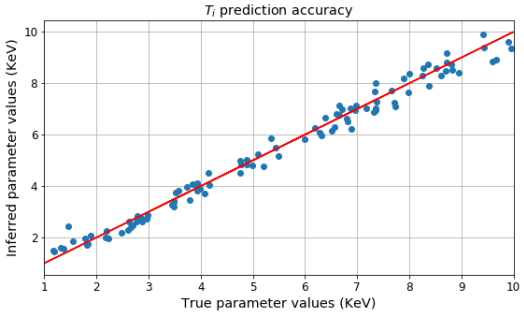

(b) Prediction plots for the fast ion density in geometry B.



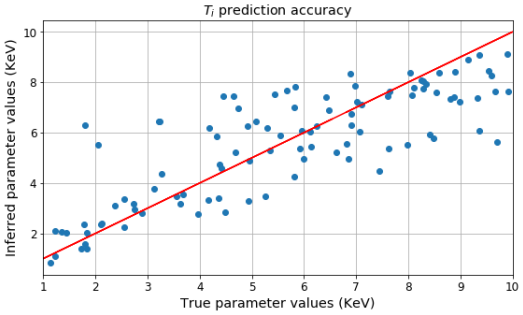

(c) Prediction plots for the bulk ion density in geometry A.



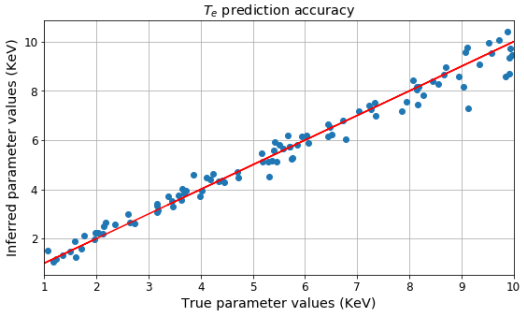

(d) Prediction plots for the bulk ion density in geometry B.



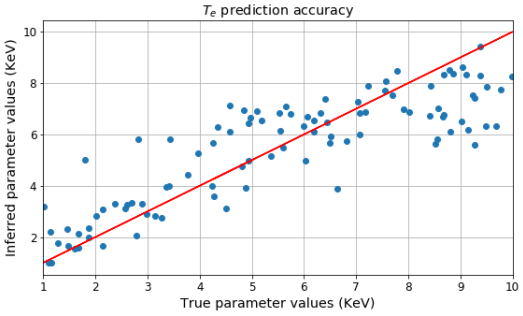

(e) Prediction plots for the bulk ion temperature in geometry A.




(f) Prediction plots for the bulk ion temperature in geometry B.

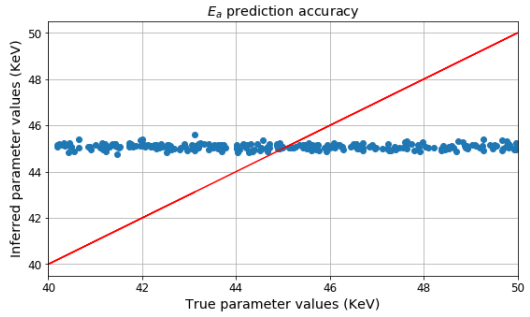



(g) Prediction plots for the bulk electron temperature in geometry A.

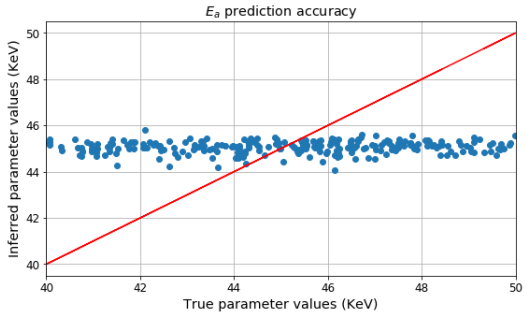



(h) Prediction plots for the electron temperature in geometry B.

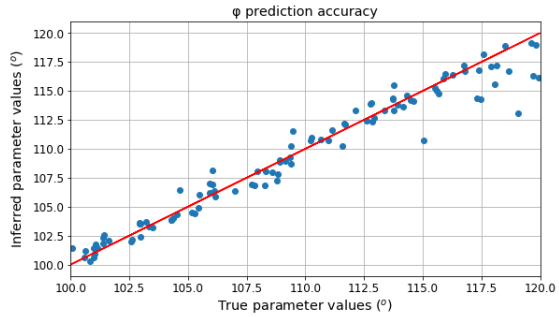
Figure 12: These plots represent the inferred values (blue) for the plasma parameters versus their true values, where the identity line (red) corresponds to complete agreement. In terms of the bulk population properties, the inference appears to work better in geometry configuration A (left). The network cannot infer the fast ion density in the W-7 X configuration (top left) while in geometry B the network can distinguish the influence to that parameter (top right).



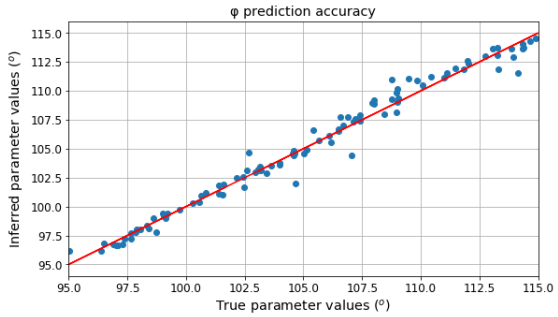

(a) Prediction plots for the fast ion energy $E_a$ in geometry A.



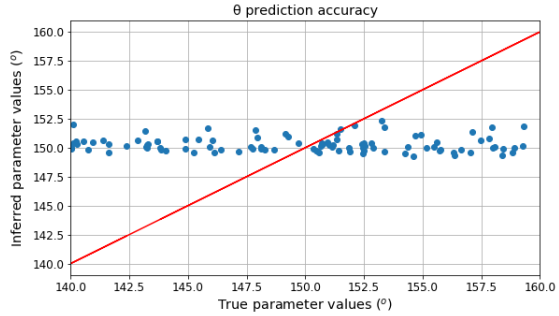

(b) Prediction plots for the fast ion energy $E_a$ in geometry B.



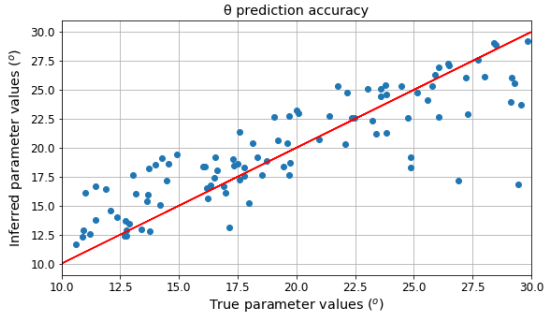

(c) Prediction plots for the angle $\phi$ in geometry A.




(d) Prediction plots for the angle $\phi$ in geometry B.




(e) Prediction plots for the scattering angle $\theta$ in geometry A.




(f) Prediction plots for the scattering angle $\theta$ in geometry B.

Figure 13: These plots represent the inferred values (blue) for the fast ion energy and the geometrical parameters versus their true values, where the identity line (red) corresponds to perfect agreement. The network appears unable to infer the value for the fast ion energy $E_a$ in either parameter space as well as the $\theta$ angle in the geometry configuration A. On the other hand, the inference of the $\phi$ angle is very accurate in both configurations.

The second feature is that the regressor is not able to infer the fast ion energy in both parameter spaces although the parameter's relative error was lower than the acceptance level. Similarly, the inference of angle $\theta$ appeared better in geometry configuration A (Figure:11a). However the parameter's prediction plot shows that the network cannot distinguish the parameter's influence in that geometry. This constitutes proof of the useful insights that can be gained from the prediction plots.

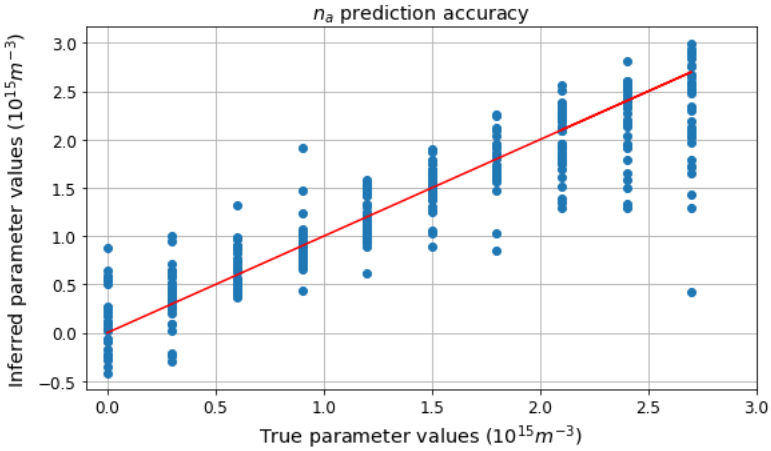



Figure 14: A database where the fast ion density was assigned specific values in the same parameter range was constructed. The deviation of the inferred $n_a$ has the same magnitude for zero and non-zero values.

## 5.2 MLP Classifiers

### 5.2.1 Classifier input and output

In classification problems the neural network has to assign the input sample to one of the classes that have been provided as targets. In the case of binary classification, the output of the network can be a boolean parameter with each value corresponding to one class. In this project, the classifiers are MLP feedforward networks implemented with the scikit-learn library [24]. Analogously to the regressors' case the input of the network is the simulated CTS spectrum. The output classes of the network are distinguished as "Presence" or "Absence" of fast ions. The pre-processing of the spectra is identical to the one followed in the regression case (Section 5.1.1) in terms of normalization and noise addition. According to Section: 4.5, some of the samples had fast ion density set to zero. For classification these are the spectra that were assigned the label "Absence", while the rest received the label "Presence".

### 5.2.2 Classifier architecture

The principles governing the choices for the structure of the neural network are the same for regression and classification problems. Similarly to the regression case, the developed classifiers used two or three hidden layers. The rule regarding the descending order for number of neurons (Section:5.1.2) along the propagation of the data is also applied in the classification case. The influence of these hyper-parameters to the classification accuracy as well as the rest of the classification metrics was investigated during this project (Section:6.2.3). The architecture of the classifiers follows the same notation rules as the regressors (Section:5.1.2)

### 5.2.3 Classifier parameters

Analogously to the MLP regressors, this section is devoted to the hyper-parameters of the classifiers that are not referring to its structure design. The parameters which were used for the classifiers are the same with the ones for the regressors (Section:5.1.3). The only difference lies in the activation function that was used for the neurons of the hidden layers. The classifiers use the

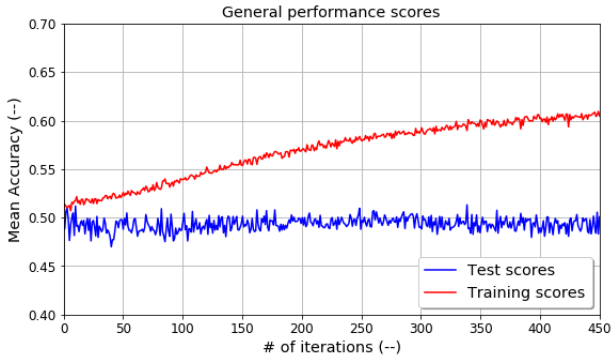
Rectified Linear Unit function (ReLU) [17]:

$$f(x) = max(0, x) \tag{34}$$

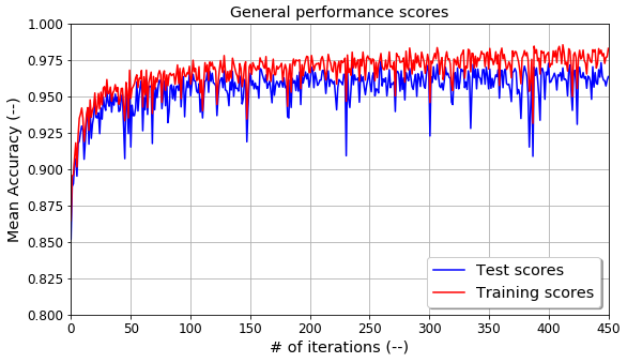
A complete list of the network parameters implemented in the scikit-learn MLP Classifier model can be found in Appendix section:11 (Table:12).

### 5.2.4 Classifier training performance

Similarly to the regression study, the developed database was split into a training set and a test set. The samples of each set consist of the calculated spectrum and the label "Presence" or "Absence". A representation of the overall performance of classifiers is available by examining the evolution of the mean accuracy during the training progress.




(a) Neural network performance on training (red) and test (blue) sets in geometry configuration A.




(b) Neural network performance on training (red) and test (blue) sets in geometry configuration B.

Figure 15: These graphs are based on the performance of the classifier L150L100L50 on noiseless spectra calculated in the two parameter spaces. In parameter space B (right), the performance on the test sets follows the one on the training samples which is not the case for configuration A (left). Also, the mean accuracy is really close to unity in geometry configuration B while for the W-7 X case it is around 0.5 for the test set and 0.6 for the training set.

From Figure:15 it is evident that the overall performance of the classifier is better in geometry configuration B. In that configuration the mean accuracy is close to 0.97 while for geometry A the corresponding value is around 0.5. Additionally, in the latter configuration, the performance of the classifier improves on the training set but not on the test set. In most cases, this is an indication that the network is over-fitting, which means that it cannot capture the generalized trends of the provided data [17] [25].

### 5.2.5 Classifier's performance on test set

In classification problems, mean accuracy is not always the most suitable parameter to evaluate the performance of the network. A set of metrics that can provide a more useful insight into the classifier performance are Recall, Precision and F1-score. The performance of the classifier is by far better in configuration B and only a small percentage of the total samples are mislabeled. At the same time, the samples calculated in configuration A receive equally correct and false labels (Table:5).

Table 5: Classification metrics for the performance of L150L100L50 on noiseless spectra in the two parameter spaces. In parameter space B the network provides better results for every metric.

| Classification metric | Geometry configuration A | Geometry configuration B |
|---|---|---|
| Recall | 0.5 | 0.96 |
| Precision | 0.5 | 0.96 |
| F1-score | 0.48 | 0.96 |
| MCC | 0.000667094 | 0.927309359 |

These metrics are commonly used in classification problems but they are vulnerable to misinterpretation since they are dependent on the size of the classes. A metric that does not have this disadvantage is the Matthew's Correlation Coefficient [27]. This parameter can take values from -1 to 1, with the first representing inverse predictions and the latter corresponding to perfect predictions. The 0 value is equivalent to random predictions. According to Table:5, the classifier performance in geometry configuration A is equally good to assigning labels randomly. This means that each sample has almost 50% chance to receive either label. On the other hand, the performance of the classifier in configuration B is very good, an observation that agrees with the result of the other metrics.

Lastly, the most straightforward visualization of a classifier's performance is the confusion matrix. Although this metric is the most descriptive, it is generally not suitable for comparison between classifiers. In this case however, the difference between the performances in the two parameter spaces is so great that it is also visible in the confusion matrix (Table:6). In configuration A the network has a predisposition to assign the label "Presence" so its performance is even worse than random assignment with a 50% chance.

Table 6: The confusion matrix for the performance of L150L100L50 on noiseless spectra in both geometries. A perfect prediction would result in zero non-diagonal elements of the matrix. In configuration A, the classifier assigns equally correct and incorrect labels. Additionally, it shows a predisposition to assign the label "Presence", so its performance is actually worse than random guesses. In geometry configuration B, only a few (36 and 55) samples were assigned the wrong label.

| | Geometry configuration A | | Geometry configuration B | |
|---|---|---|---|---|
| | Predicted Absence | Predicted presence | Predicted Absence | Predicted presence |
| True Absence | 366 | 881 | 1211 | 36 |
| True Presence | 367 | 886 | 55 | 1198 |

# Part III
# Results

## 6 Result analysis

In this chapter we present the results of our research project and compare them with previous work in related literature. Firstly, we address the main result of our work (Section:6.1) and then the supplementary conclusions of our investigation (Section:6.2). These additional results are divided into modelling results (Subsection:6.2.1), regression results (Subsection:6.2.2), classification results (Subsection:6.2.3) and lastly any results which do not fit in these categories (Subsection:6.2.4). All of these are discussed in detail in their respective subsections. The discussion of the main result is only briefly touched in this chapter since it is the focus of the following chapter (Section:7).

### 6.1 Main result

The aim of this project was to answer the research question that we posed in the beginning. Equivalently, the main conclusion of this research should be an answer to that research question. This project was a feasibility study of fast ion detection in CTS spectra with neural networks.

Our main result is: fast ion detection is possible in CTS spectra with classification neural networks. A prerequisite for this feasibility is that the spectra are calculated in scattering geometry configuration B ( $10^o < \theta < 30^o$ and $95^o < \phi < 115^o$).

The achieved accuracy ($\sim 97\%$) of the networks' label assignment in these circumstances showed that the signal-to-noise ratio (of the fast ion component) can be optimized with data analysis techniques. A catalytic factor for this optimization is the scattering geometry of the CTS measurements. The impact as well as the limitations of this result are discussed in more detail in Section:7

### 6.2 Supplementary results

#### 6.2.1 Modelling results

Firstly, we address the conclusions that were drawn from the developed code for the forward model.

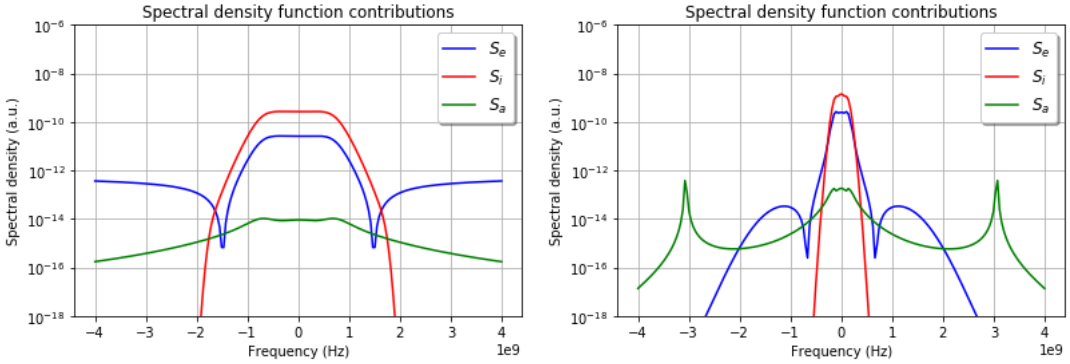
##### Arbitrary distribution polynomial approximation

As it was mentioned in Section:4, the initial approach was to implement the model described in [6]. This model provides an analytical expression for the spectral density function by approximating the species' distribution function with a series of orthogonal polynomials. A model which can be applied on an arbitrary fast ion distribution, given in either analytical or numerical form, has very promising prospects. This kind of model gives the opportunity to explore many different distribution cases and provide generalized results. These advantages lead us to invest a considerable amount of time on the implementation of this approach. Unfortunately, this investment was not fruitful. We were not able to reproduce the conclusions and the spectra that are presented in that paper. The main example, which also constituted the model's benchmark test, was the simulation of a Maxwellian distribution spectrum. According to [6], this case only requires the zero

order approximation ($a_{00}$ term). However, in our code implementation, we observed the following: higher order terms had significant values, odd-numbered order coefficients exhibited a pattern and most importantly the resulting spectrum did not resemble a Maxwellian case even when higher order approximation terms were included. A preliminary mathematical analysis revealed that the distribution approximated with the $a_{00}$ term is anisotropic and does not resemble a Maxwellian distribution. Since the reproduction of the Maxwellian distribution was not successful it was not clear how to proceed. Due to shortage of time and the mathematical complexity of the approach, we decided to pursue a well-established case for fast ions, the slowing-down model instead.

### Slowing-down model spectrum

The slowing-down model, assumes a classical slowing-down distribution for the fast ions and thermal Maxwellian distributions for the bulk populations. A good starting point for the evaluation of the model is the calculated contributions to the spectrum from each of the species. The spectra simulated by our code were benchmarked with the spectra presented in [20] [19]. The form of the spectral functions for the bulk ions and electrons is in agreement with similar synthetic spectra [20] [35].




(a) Geometry configuration A (b) Geometry configuration B

Figure 16: The bulk ion contribution (red) is dominant in the region of the probing frequency for both configurations. On the sides of that region, the electron contribution (blue) is the strongest one in configuration A. In parameter space B, the fast ion component (green) can be dominant in spectral regions away from the probing frequency, especially for scattering geometries were resonances are present.

Although we assumed NBI as the origin of the fast ions in our model, their spectrum does not exhibit the asymmetry that is observed in both simulations and experiments [15] [13] [10]. The lack of asymmetry is a result of the electrostatic approximation that is applied in our model. Simulations of synthetic spectra that use this approximation in related research [19] [20] take advantage of this feature and limit their calculations to frequencies higher than the probing frequency. Furthermore, in our code implementation the calculation of the fast ion susceptibility (Equation (28)) and the velocity distribution (Equation (25)) used the absolute value of $\omega$. This choice affects the bandwidth half that lies below the probing frequency. This decision was made to deal with occurring numerical errors related to the data type of the variables. From a physics point of view, this modelling choice is equivalent to assuming an isotropic distribution for the fast ions. The latter is an assumption that is applied for the contribution of fusion alpha particles in a plasma

which follow a slowing down distribution [35]. Since the purpose of this project is to examine the possibility of fast ion detection in a CTS spectrum, without explicit connection to NBI, we deem that this modelling choice does not affect the validity of our results.

Next we look into the region where each contribution is dominant. The bulk ion contribution is dominant around the probing frequency and the width of the spectral component depends on the bulk ion temperature. This dependency is clear in our research (Figure:9) and in agreement with literature [18] [19] [13]. Away from that region, the CTS signal is significantly lower. In our modelling process we calculated the Salpeter parameter, an indicator for scattering in the collective regime. For some of the spectra calculated in parameter space A, the Salpeter parameter was lower than one. These cases occurred for subspaces of the parameter space which were not excluded. Even when we are operating in the collective regime, the main contribution to the total spectral density away from the region of the probing frequency is from the electrons and not the fast ions, especially in configuration A (Figure:16a). We expect the fast ion signal to be more relevant in the frequency area away from the probe [15] [36]. We attribute the fact that the electrons' contribution is stronger compared to the fast ions to the low levels of fast ion density and the influence of the scattering geometry which we will discuss below. In geometry configuration B, the fast ion component can overcome the electron one, especially for large Doppler shifts (16b). This effect is enhanced by a plasma wave resonance. Further analysis of this feature will follow later in this section.

**Spectrum dependency on modelling parameters**

The next aspect of the model we investigated is the spectrum's dependency on the modelling parameters. As we mentioned above, the bulk ion temperature affects the width of the bulk ion contribution and by extent the width of the spectrum in the area around the probing frequency. This species' density has a small influence on the height of the bulk ion component which determines the spectrum's central peak. In addition, this parameter affects the "wings" of the spectrum, a region where no bulk ion contribution is present. This dependence stems from the method we used to calculate the electron density in our model. Using the quasi-neutrality condition, the electron density for our model was defined as the sum of the ions' densities. Due to the low density of fast ions in comparison with the bulk ion population, the latter is the one that defines the electron density. Since the electron contribution depends on the electron density, the areas where this contribution is the strongest (the wings of the spectrum) are influenced by this parameter and by extent from the bulk ion density. Similarly the electron temperature affects mainly those frequency regions. Regarding the fast ion density, its influence on the spectrum is only visible in scattering geometry B and its effect appears to be in agreement with [13]. The dependencies analyzed here are in agreement with previous results which used the same or similar forward models [19] [18] [14].

In our research the fast ion energy does not appear to have any effect on the simulated spectrum. Simulations of CTS spectra for different NBI energies show that the calculated spectrum is only affected for significant increases of the beam's energy [15]. This effect is a small increase in the spectrum's height (in linear scale). However, this was observed when two instead of one NBI beams were introduced to the plasma so this observation could be the result of increased fast ion density instead of the species' energy [36].

In our research we examined two geometry configurations: configuration A simulated the scattering geometry that is applied in the Wendelstein 7-X stellarator, configuration B is a scattering geometry that was defined in this project and was inspired by the investigations in [19] and [20].

The dependency plots presented in this thesis highlight the strong influence that the scattering geometry has on the simulated spectra. The simulation geometry is expressed in our model through the scattering angle $\theta$ and the resolved angle $\phi$. The effect of these parameters on the calculated spectrum depends significantly on the investigated parameter range. In both geometry configurations, $\phi$ angle affects the wings of the spectrum. More importantly, this angle influences the presence of the resonance in parameter space B. The influence of the scattering angle $\theta$ is only visible in configuration B and it appears to affect both the wings of the spectrum as well as the width of the central region. The importance of these parameters has also been demonstrated in publications which aimed to simulate the spectral density function of a CTS experiment [18] [19] [20]. The conclusions of those references agree with our results but it should be noted that in some cases [20] their investigation was performed in a considerably different parameter range ($\phi < 90^o$ and $\theta < 90^o$).

**Lower Hybrid Resonance**

Closely related to the scattering geometry parameters is the presence of a resonance in the spectra we calculated in a subspace of configuration B. This resonance has been identified as the lower hybrid resonance of the plasma and originates from the fast ion contribution. This resonance strongly enhances the CTS signal in the region and is present when the resolved fluctuation direction is almost perpendicular to the magnetic field ($\phi \simeq 90^o$). In these circumstances, the bulk ion density is the parameter that decides whether the resonance will appear inside the observed scattered bandwidth. Evidence of this resonance have been documented in simulations [19] [20] but have also been predicted and formally described in [9].

Lastly, both the slowing-down model and the one applying the polynomial expansion operate in the electrostatic approximation. The applicability of this approximation is limited by the coupling of the longitudinal and the transverse modes in a magnetized plasma [18]. However, it has been shown [37] that a fully electromagnetic treatment of the plasma supports the existence of the lower hybrid resonance in the same parameter space.

### 6.2.2 Regression results

The development of the model was followed by the construction of a database with synthetic spectra for each parameter space. The next step was the implementation of neural networks through the scikit-learn Python library to detect the fast ions in the calculated spectra. The developed codes for the training and evaluation of the networks were based on previous work which investigated the inference of the bulk ion temperature from CTS spectra [4]. The scripts are documented and available in the Appendix section:13.

At first we investigated the networks' ability to distinguish fast ions in the spectra by inferring through regression the fast ion density which was used for the spectrum calculation.

**Fraction of Variance Unexplained (FVU)**

A general metric for the performance of the regressors is the Fraction of Variance Unexplained (FVU). The aim of the designed networks was to infer the values of the parameters that were used to simulate the spectrum. In terms of the FVU metric, a perfect inference would result in a zero value for FVU. Clearly, operation in geometry configuration B achieved a lower FVU score, and by extent more accurate inference, compared with configuration A (Figure:10). In order to

evaluate our results, we compared the achieved FVU scores with the ones presented in [4]. In that publication, the achieved FVU at the end of training is around 0.5 for the test and training set. Our result for the regressors trained in the W-7 X configuration (geometry A), similar to the parameter space in [4], reaches 0.55. Personal communication with the author of that publication revealed that he was able to improve the scores of his developed networks and reach a value of 0.15 which is even better than our achieved FVU of 0.25 in scattering geometry B.

However when comparing these results we should keep in mind that: first, the spectra were calculated with different codes, mainly the former did not include the fast ion feature in the simulated spectra and second, the FVU parameter is a good general regression metric but it evaluates the performance for all the inferred parameters and not specifically for the parameter which indicates the fast ion presence, which is the aim of this project.

Another performance evaluation that is suggested in literature of machine learning, is to compare the scores of the designed network with a simple regressor (dummy regressor) which makes estimations using simple rules. This comparison is used to establish a baseline and is not intended for application on real problems [24]. For our case, a dummy regressor that always infers the mean value of each parameter has an FVU score of 1.0003 and 1.0009 for configurations A and B respectively. Obviously our networks outperform the baseline significantly in both scattering geometries in terms of the FVU metric.

**Relative errors analysis**

The FVU parameter provides an insight on the overall performance of the regressor but we want to investigate the network's performance with respect to the fast ion feature. In our model, the most closely related parameter to the fast ion presence, is the species' density [15]. We evaluate the ability of the network to infer this parameter by examining the corresponding relative error. We set as a very good inference level, a relative error of 10% and as sufficient a 20% relative error. We should note that these limits are more strict compared to the acceptance levels that are set in related research studies (30%) [15] [36]. By tracking the relative error of the fast ion density during the training of the network (Figures:10a), it is clear that the regressor trained in configuration A does not respond to the fast ion density. In fact the achieved relative error is the same as the one achieved by a dummy regressor that always predicts the mean value of the parameter. In parameter space B on the other hand (Figure:11b), the regressor is reacting to the fast ion density and the parameter's relative error is improving with training (90% at the end of training). But the error levels that are reached at the end of training exceed by a fair margin the acceptance levels that we have defined. The corresponding baseline dummy regressor in this parameter space has a relative error of 3.18 for the fast ion density.

The respective metrics that are achieved for the rest of the modelling parameters are lower than the accepted levels, with the exception of the bulk species temperatures in parameter space B. Aside from the fast ion density, the relative errors that are achieved in configuration A are lower compared to the scattering geometry B. Similarly to the FVU parameter discussion, the achieved relative errors by the baseline dummy regressor are higher: for the species' temperatures and bulk ion density($\sim 65\% - 70\%$) and the resolved angle $\phi$ (5%). The respective parameter for the fast ion energy (5.5%) is in the same range.

The scattering angle $\theta$ is the only parameter for which the baseline regressor performs significantly different in the two parameter spaces. In configuration A our regressor performs in the same levels as the baseline, while in parameter space B, our network outperforms the baseline with almost 20% lower error. In a more general view, the relative errors of the designed regressors

are higher than the ones achieved by other inference methods, such as Bayesian inference [18]. However, the latter is a much slower process compared to the neural networks, intended for offline analysis.

### Prediction plots

The relative errors of the inferred parameters are useful evaluation metrics but they can be misleading. For this reason our research includes prediction plots: plots of the inferred versus the true values of the parameters (Figures: 13 and 12). If the inference is perfect, then the points in these plots will lie on the identity line.

Regarding the fast ion density, we can clearly see that the regressor in parameter space B is responding to the variance of the parameter. The inferred values exhibit significant deviation as we move towards the upper limit of the parameter range. However, the aim of the project is not to retrieve the fast ion density value but distinguish between presence and absence of the ion feature. For that purpose we created a new database where the fast ion density had predetermined values in order to see if the inference accuracy is better for zero than non-zero values. Unfortunately, the deviation of the inferred parameter values is independent of this characteristic (Figure:14). Therefore the use of the inference deviation is not a reliable criterion to establish presence or absence of fast ions.

For the electron temperature, bulk ion temperature and density, the prediction plot confirm the conclusions of the relative error analysis. Furthermore, the prediction plot for the inference of the bulk ion temperature in parameter space A is very similar to the one presented in [4]. The prediction plots for the scattering angle and the fast ion energy prove that conclusions drawn by the relative errors examination can be misleading. In both parameter spaces we saw that the relative error of the fast ion energy was at 7-8%. However the prediction plots clearly show that the regressors are not responding to the variance of the parameter.

The result for the scattering angle is even more interesting: the relative errors lead us to the conclusion that this parameter can be more accurately inferred in the W-7 X configuration. However the respective prediction plots show that the regressor does not respond to the changes of the parameter in scattering geometry A. On the contrary the inference accuracy in parameter space B may be lower but the prediction plot shows that the network responds to the parameter variance.

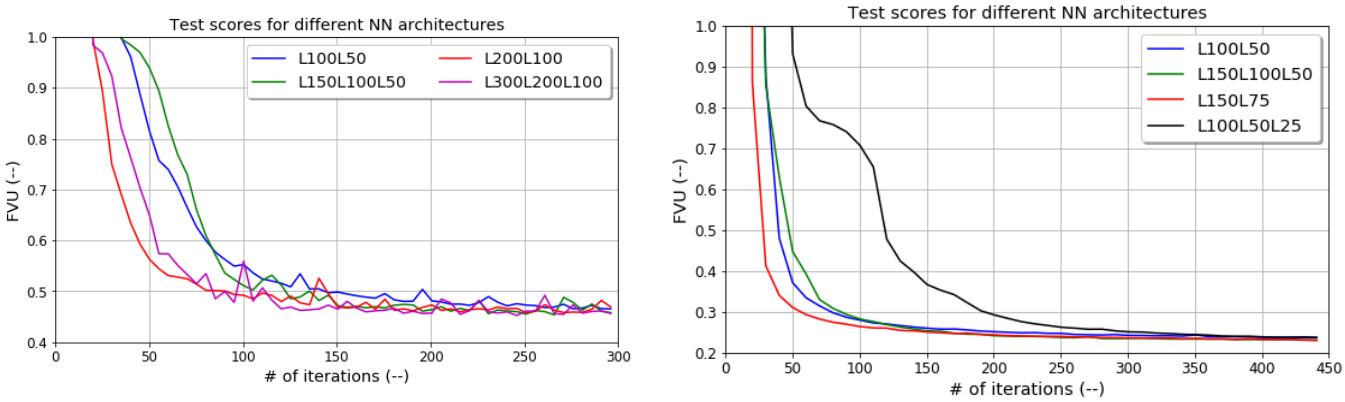
Unfortunately, the bulk ion temperature prediction plot in [4] is the only available reference for this type of plots. Nonetheless, we have demonstrated without a doubt that this type of plots can provide very useful insights and can serve as validation for the conclusions drawn by other metrics such as the relative error investigation.

### Influence of the network's hyper-parameters

In this research project, we investigated the effect of the neural network hyper-parameters on the performance of the regressors. This investigation was based on the evaluation of the FVU parameter and the relative error of the inferred fast ion density. The focus of our investigation was on the hyper-parameters that define the architecture of the neural network. According to [38], the architecture of a network is defined by the input neurons, the hidden neurons (number of hidden layers and number of neurons at each layer), the activation function and the weights connecting the layers. The input neurons were defined by the number of sampling frequencies in

our simulated spectra. The weight initialization was implemented with the default values of the scikit-learn module (Appendix section:11, Table:11).

Regarding the FVU parameter, the regressor architecture does not affect the value that is reached at the end of the training procedure, but it influences how fast the parameter converges to that value (Figure:17). This conclusion holds for both geometry configurations and is in agreement with literature [39]. The evolution of the FVU parameter in configuration A shows a high fluctuating behaviour when a large number of neurons is used in the hidden layers. In scattering geometry B, the use of two hidden layers appears to result in faster convergence of the neural network.




(a) FVU parameter evolution for different regressor architectures in parameter space A.

(b) FVU parameter evolution for different regressor architectures in parameter space B.

Figure 17: The regressor architecture does not affect the performance of the network at the conclusion of the training process. However, this hyper-parameter influences the speed with which the network converges to the final value. Additionally, in the W-7 X configuration, a high number of neurons in the hidden layers leads to stronger fluctuations as FVU decreases (magenta line in left picture).

As it was mentioned in Section: 5.1.5, the regressor trained in configuration A cannot infer the value of the fast ion density. For this reason our investigation on the relative error was focused in configuration B. As it can be seen in Figure:18, with the exception of L100L50L25, the achieved relative error is almost the same independently of the regressor architecture. Similarly to the FVU parameter, the hyper-parameter influences the rate of convergence towards the final value rather than the value itself. Both in terms of FVU and relative error, the network that exhibits the best performance in the new parameter space is L150L75.

A preliminary investigation with networks containing more than three hidden layers was conducted in the two parameter spaces. In the W-7 X configuration, the FVU parameter exhibits a slight increase but the network remains insensitive towards the fast ion density. In parameter space B, the FVU parameter remains in the same levels but the relative error of the fast ion density becomes worse.

The regressor architecture is an important parameter because it defines the complexity as well as the learning capability of the network [38]. The structure of the network is in most cases decided heuristically. However, the search for an algorithm that defines the optimal architecture is an active field of research in machine learning. If the network is too small, then it will not be able to extract the intended features from the data. On the other hand, if the network is too large,

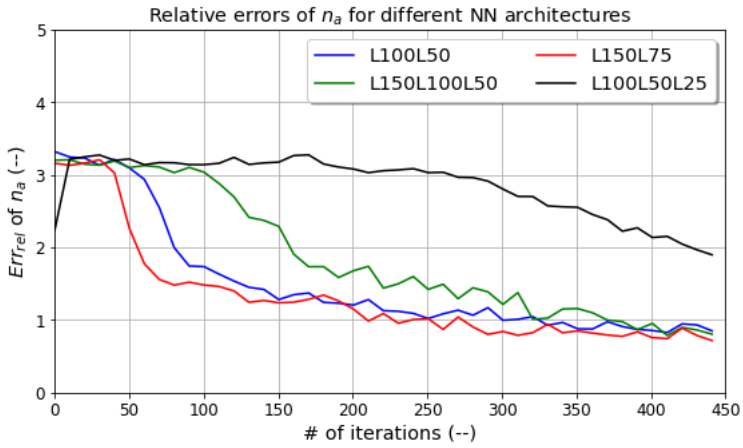



Figure 18: This graph depicts the influence of the regressor's architecture on the fast ion density relative error. For most cases,the chosen architecture only affects the rate of convergence to the final value. A low number of nodes in the last hidden layer (black line) of the network creates the need for more epochs of training to reach the same level of relative errors as the other cases.

then it can lead to over-fitting [39]. In the latter case, the network responds to the random noise in the data instead of the underlying features.

There are many different algorithms that optimize the network structure. The most commonly used are pruning algorithms. These algorithms start by using an oversized structure and as a second step they remove unnecessary neurons and weights. The decision-making regarding the usefulness of the neurons/weights depends on the applied algorithm. Common methods include magnitude-based, sensitivity based and brute-force decision-making. Further information on these algorithms can be found in [39] and its related references. We deem that further analysis on the pruning algorithms and network structure optimizers lies out of the scope of this thesis. Nevertheless, none of the regressors developed in this project exhibit over-fitting for noiseless spectra.

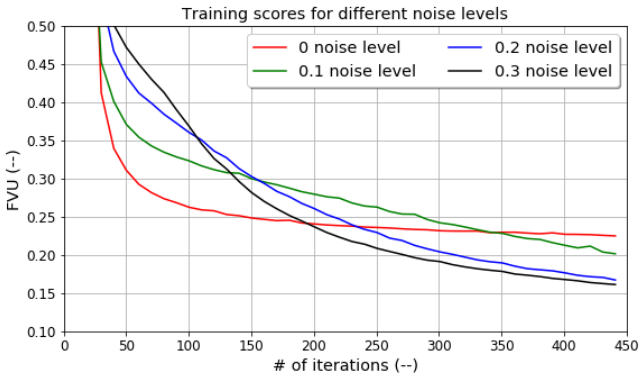
In addition, a preliminary investigation with respect to the optimization solver and the activation function of the hidden layers showed that the Adam solver is by far a better choice for this application. Regarding the activation function, the best performance is reached with the logistic function, followed by the tanh function and lastly the ReLU function. This result agrees with literature suggestions for regression problems [33] [25]. Furthermore, different initializations of the regularization term $\alpha$ affect the rate of convergence towards the final value but not the value itself.

Lastly, the performance of the regressors is independent of the spectral resolution. Both in terms of FVU and relative error, the behaviour of the network is the same when the spectra have 0.0625 $MHz^{-1}$ (500 sample frequencies) and 0.025 $MHz^{-1}$ (200 sample frequencies) spectral densities.

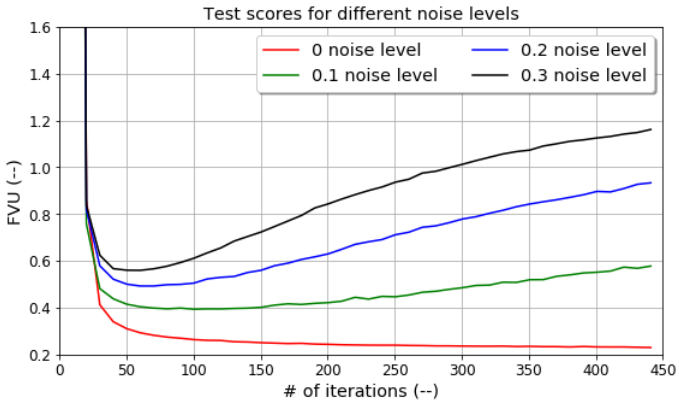
### Noise addition investigation

One of the goals of our research was to take advantage of the fault tolerance that characterizes the neural networks and apply it on the noisy CTS signals. Towards that aim, we investigated the influence of added noise signals of Gaussian origin on the FVU and relative error metrics of the regressor. The amplitude of the noise signals was defined as a fraction of the mean of the maximum values of the spectra. In parameter space B, the addition of noise in the spectra results in worse network performance on the test sets (Figure:19b). On the contrary, the performance

of the regressor on the training sets is improving with the addition of noise (Figure:19a). The discrepancy between the behaviours of the regressor on the two sets is a sign that the network is over-fitting [25] [27]. The situation is similar for geometry configuration A.




(a) FVU parameter for the performance on the training sets for different noise levels in parameter space B.




(b) FVU parameter for the performance on the test sets for different noise levels in parameter space B.

Figure 19: The addition of noise in the synthetic spectra has a negative influence on the performance of the regressor on the test set (right). On the other hand, the achieved FVU parameter on the training set is improving with further noise addition (left). This discrepancy is a clear indication that the network is over-trained and not responding to the input features.

As a comparison, the addition of noise in [4] lead to a decrease of the FVU parameter to 0.7 and 0.64 for 20% noise level and 30% noise respectively, when the achieved score for noiseless spectra was at 0.86.

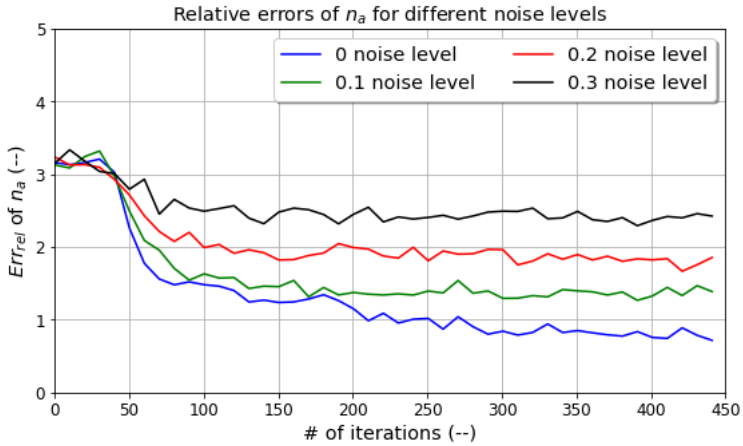



Figure 20: The performance of the regressor L150L75 in terms of the fast ion density relative error for different noise levels is presented in this graph. The addition of noise increases the errors for the inferred fast ion density.

The influence of noise addition on the relative error of the fast ion density is similar to the FVU parameter case. The addition of noise results in higher relative errors of the parameter (Figure:20). Of course this result refers to the spectra developed in parameter space B, since the networks trained on the W-7 X configuration are not sensitive to the fast ion density values. However, it should be noted that the relative error achieved in the new parameter space for noise reference levels of 30% is still lower compared to the one achieved in the W-7 X configuration.

Lastly, we should mention that the reference noise levels we examined are much lower compared to the ones in experimental CTS measurements. In these cases the noise can be up to 10 times higher than the CTS signal. With that in mind, our investigation refers to noise that "escapes" the noise subtraction procedure that is applied in CTS measurements. An uncertainty level that is related with background subtraction, beam normalization and intensity calibration is 10% [15]. According to that, our investigated noise levels are even higher than the ones that are assumed in literature. We observe that the regressor performance is significantly worse even for 10% noise levels. More importantly, the networks are over-fitting even for this noise level. Therefore, the fault tolerance of the regressors is a feature that needs to be improved. The first step towards improving the noise robustness of the networks would be to treat their over-fitting behaviour with a cross-validation algorithm.

### 6.2.3 Classification results

Since regression could not provide us with conclusive indication for the fast ion presence, we decided to use classification networks. For that purpose, we developed a new database where half the spectra contained the fast ion feature while in the rest this contribution was absent. These classes were assigned the labels "Presence" and "Absence" respectively. Similarly to the regression case, classifiers were implemented with the scikit-learn module [24] and were trained in spectra from both scattering geometries. The parameters that were examined are the mean accuracy and the classification metrics that were introduced in Section:3.4.

#### Mean accuracy and classification metrics

The performance of the classifiers was evaluated through their mean accuracy, precision, recall, F-1 score [28] and Matthew's Correlation Coefficient [30]. In fusion research there are many examples for the inference of parameters from CTS spectra. However, we could not find any previous classification approach to filter the spectra (simulated or experimental) with respect to their feature. In that respect, the application of classification on this topic is a novel approach. As a result we had to search for performance references in the general machine learning field. Unfortunately there is no clear limit regarding the acceptance levels for the classification metrics, since they highly depend on the treated problem.

Analogously to the regression case, a baseline can be determined through simple (dummy) classifiers. In our research we used as baseline two dummy classifiers: one which generates predictions uniformly at random (uniform strategy) and another which generates predictions with respect to the training set's class distribution (stratified strategy) [24]. The baseline classifiers perform at the same level for both parameter spaces with mean accuracy 0.49 and 0.5 for each strategy. Regarding the rest of the classification metrics (Precision, Recall and F-1 score) they vary between 0.5 and 0.51 for the two strategies in each parameter space with no clear advantage. The MCC parameter for these baseline classifiers is around zero, receiving very small positive or negative values.

In comparison with these results, we can clearly state that our classifiers which were trained in scattering geometry A perform on the same level as the baseline classifiers (Table:5). More importantly, the improvement of the classifier's accuracy on the training set is not followed by a similar behaviour on the test set (Figure:15a). This is a sign that the network is over-fitted. Since this performance corresponds to noiseless spectra and the classifier performs on the same levels as a random predictor we believe that the source of the over-fitting lies either in the chosen network architecture [39] or the inability of the classifier to distinguish the fast ion feature in this parameter space.

On the other hand, the classifiers that are trained in parameter space B, do not exhibit any over-fitting behaviour and they perform on a very high level ($\sim 0.96$) in all the classification metrics. For our evaluation we also used the classification metric Matthew's Correlation Coefficient (MCC). This parameter has proven to be more reliable compared to the rest of the metrics due to the balanced proportionality of each confusion matrix's class in its calculation. This property is especially important when the classes in the database have significant size differences [27]. The conclusions obtained by the other classification metrics for the performance of the classifiers in the two parameter spaces are confirmed by the MCC parameter (Table:5).

**Minimization of False Positives (FP) versus False Negatives (FN)**

The evaluation of the classification metrics is closely related to the preference of minimizing the false positive or the false negative results. The first approach optimizes the Precision while the second targets the Recall metric [25]. In our case, false positives correspond to the assignment of "Presence" to "Absence" samples and false negatives correspond to the assignment of the label "Absence" to spectra where the fast ion feature is present. Given that the fast ion feature is very hard to detect, we consider that the aim of the classifier should be to minimize the false negatives, since the false positives could be eliminated with further follow-up investigation. Fortunately, our classification results for Recall and Precision are very close and exhibit the same behavior in all our investigations.

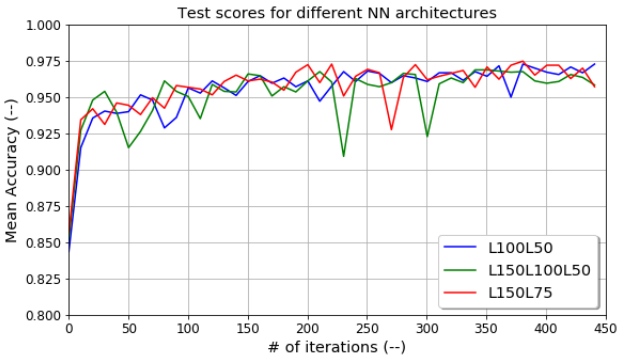
**Influence of the network's hyper-parameters**

Analogously to the examination of regressors, we investigated the influence of the classifier's architecture on the performance of the networks. Our investigation was based on the mean accuracy and the classification metrics achieved by the classifiers. The network architecture aspects we examined were the number of hidden layers and the distribution of neurons in these layers.

The influence of the classifier architecture on the mean accuracy in the two configurations can be seen in Figure:21. The classifier's architecture does not affect the achieved value at the end of the training process. However, its effect on the rate of convergence is not evident either, instead the chosen architecture affects the smoothness of the parameter's decrease. The highest fluctuations are observed when the classifiers have three hidden layers.

The effect of the hyper-parameter on the classification metrics is summarized in Tables:7 and 8 for configurations A and B, respectively. For the first parameter space, the influence of the NN architecture on all of the metrics is very weak and only observable in the F-1 score and MCC metric. On the other hand, the classification metrics in the geometry B, exhibit a slight dependency on the classifier's number of hidden layers. More specifically, a classifier with two hidden layers and less neurons per hidden layer (L100L50) demonstrates the best performance in terms of the MCC


(a) Mean accuracy evolution for different classifier architectures in parameter space A.




(b) Mean accuracy evolution for different classifier architectures in parameter space B.

Figure 21: The classifier architecture does not influence the mean accuracy that is reached at the end of the training in either parameter space. However it affects the smoothness of the parameter's decrease, especially in scattering geometry B (right). Higher fluctuations are observed when the classifier has three instead of two hidden layers (green).

metric.

Table 7: Classification metrics for different NN architectures in geometry configuration A. The hyper-parameter does not affect the performance of the classifier. The only observable difference can be seen in terms of F-1 score and MCC but the influence is weak.

| NN architecture | Precision | Recall | F-1 score | MCC |
|---|---|---|---|---|
| L100L50 | 0.5 | 0.5 | 0.49 | -0.006947573 |
| L150L75 | 0.5 | 0.5 | 0.5 | -0.0032135 |
| L150L100L50 | 0.5 | 0.5 | 0.48 | 0.000667094 |

Table 8: Classification metrics for different NN architectures in geometry configuration B. Classifiers with two hidden layers exhibit better performance. In terms of MCC, less neurons per hidden layer leads to better results.

| NN architecture | Precision | Recall | F-1 score | MCC |
|---|---|---|---|---|
| L100L50 | 0.97 | 0.97 | 0.97 | 0.947739004 |
| L150L75 | 0.97 | 0.97 | 0.97 | 0.933349727 |
| L150L100L50 | 0.96 | 0.96 | 0.96 | 0.927309359 |

The small dependency of the classification metrics on the network architecture is in accordance with the performance of classifiers with different structures presented in [39].

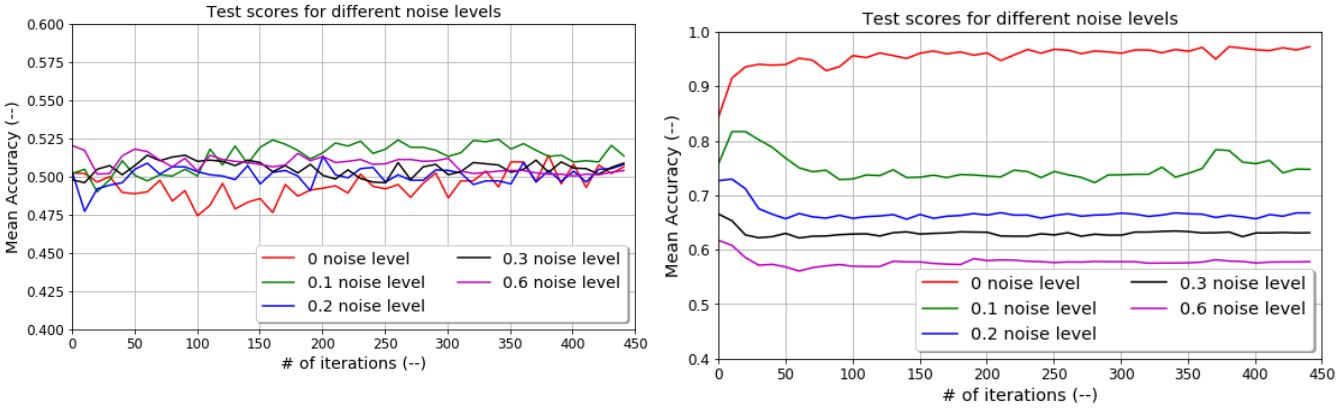
Regarding the classifiers operating in the scattering geometry A, we mentioned that they exhibit signs of over-fitting. The discrepancy between the accuracy scores for the training and test set is less pronounced but still evident for smaller networks. Therefore, it is safe to assume that the main source of over-training resides with the chosen parameter space rather than the network architecture.

### Noise addition investigation

Similar to the investigation of the neural network architecture, we examined the noise robustness of classifier L100L50. We investigated the effect of noise levels up to 60% on mean accuracy and the classification metrics (Precision, Recall, F-1 score and MCC).

First let us look into the mean accuracy of the classifiers (Figure:22). In geometry configuration B, the addition of noise worsens the performance of the classifier. However, the achieved accuracy in this configuration is higher than the one achieved at configuration A even when the noise level is at 60%.

On the other hand, the classifier that is developed in the W-7 X parameter space is not affected by noise addition. In fact, the performance for noise level of 10% is slightly better compared to the noiseless case. However we should be reluctant to derive conclusions from these results considering that the classifiers in configuration A are over-trained even on noiseless data. This feature is also present in the new parameter space when noise is added to the spectra, more specifically the discrepancy between the performance on training and test set grows for increasing noise levels.




(a) Mean accuracy evolution for different noise levels in parameter space A.

(b) Mean accuracy evolution for different noise levels in parameter space B.

Figure 22: The performance of the classifier L100L50 for spectra with different noise levels in the two parameter spaces is presented in this figure. In configuration A, an addition of 10% of noise appears to slightly improve the performance of the classifier. In parameter space B, noise addition results in worse performance but the achieved scores are better compared to the W-7 X configuration even for noise levels of 60%.

With respect to the other classification metrics, the effect of noise addition on them for each parameter space are summarized in Tables:9 and 10. For configuration A, the results from these metrics confirm the conclusion of the mean accuracy investigation. The addition of low levels of noise ($\sim 10\%$) leads to slightly better performance of the classifier, a result that is reflected on all the metrics. In parameter space B, noise addition leads to a steep decrease of all the classification metrics. Additionally, the values of the classifier for all the metrics remain higher compared to ones in configuration A.

Again, we should point out that the conclusions about the noise robustness of the classifiers in parameter space A may be misleading. As we saw above, the classifiers in this parameter space exhibit over-fitting performance even for noiseless spectra.

Lastly, a preliminary investigation showed that the classifier performance can be improved by

Table 9: Classification metrics for different noise levels in geometry configuration A. The classifier performs slightly better when low levels of noise are included ($\sim 10\%$). Further addition of noise does not affect the result significantly, but the quality of performance is already low for noiseless spectra.

| Noise level | Precision | Recall | F-1 score | MCC |
|---|---|---|---|---|
| 0 | 0.5 | 0.5 | 0.49 | -0.006947573 |
| 0.1 | 0.51 | 0.51 | 0.51 | 0.012833864 |
| 0.2 | 0.49 | 0.49 | 0.49 | -0.02079436 |
| 0.3 | 0.49 | 0.49 | 0.49 | -0.012850607 |
| 0.6 | 0.49 | 0.49 | 0.49 | -0.010427564 |

Table 10: Classification metrics for different noise levels in geometry configuration B. Addition of noise worsens the performance of the classifier. However the classifier provides better metrics compared to the W-7 X configuration even for high levels of noise ($\sim 60\%$).

| Noise level | Precision | Recall | F-1 score | MCC |
|---|---|---|---|---|
| 0 | 0.97 | 0.97 | 0.97 | 0.947739004 |
| 0.1 | 0.75 | 0.74 | 0.73 | 0.488593526 |
| 0.2 | 0.67 | 0.66 | 0.65 | 0.330122012 |
| 0.3 | 0.65 | 0.65 | 0.64 | 0.295527412 |
| 0.6 | 0.59 | 0.59 | 0.58 | 0.170982568 |

a small percentage ($\sim 1\%$) if the network is trained on a larger database.

### 6.2.4 Additional results

As it was mentioned earlier, the developed code regarding the database construction and neural network implementation was based on previous work [4]. Neural networks require a large amount of data for their training, a requirement that may cause difficulties. For our project, the database consists of synthetic spectra, therefore there is virtually no limit to the amount of spectra that can be produced. However there is a computational cost limitation, which mostly depends on the time required for the database development. Naturally, the time required for a spectrum simulation is proportional to the number of sampling frequencies of the spectrum. During this thesis project, attempts to optimize spectrum simulation run-time lead to a decrease of the required time ($\sim 400\%$) for spectra which included two and five times more sampling points compared to [4]. A further optimization of the saving process contributed substantially in the reduction of the required time for the database development.

Lastly, we should mention that the current code implementation of the model faces numerical difficulties for specific combinations of the modelling parameters in geometry configuration A (W 7-X configuration). The origin of the numerical errors lies in the exponential terms which get very small values for these parameter combinations. The issue is related to the "low" magnetic field values since the problem does not appear when B-field gets higher values. Luckily the influence of this parameter on the spectra is limited, as it can be seen in its dependency plot (Figure:8 (bottom right)). There were no similar problems in the geometry configuration B.

# 7 Summary and discussion

In this research project we investigated the feasibility of fast ion detection in CTS spectra through the application of neural networks. Towards that goal, we developed a code which simulates the spectral density function of a CTS spectrum. This code is a forward model developed in the electrostatic approximation. Our spectra simulated the response of a clean deuterium plasma with thermal bulk ion and electron populations. The fast ion population in the plasma was described by a slowing-down velocity distribution.

Our code explored too scattering geometries: configuration A simulates the geometry that is used in Wendelstein 7- X for CTS measurements; configuration B was defined in this research project and inspired by relevant literature [19]. The fast ion contribution in the spectra calculated in the second configuration was more pronounced, mainly due to the presence of a plasma resonance. Next, we constructed databases with synthetic spectra for each of these parameter spaces.

We used these databases for the training of our neural networks. For the implementation of the latter we used the models from the Python library scikit-learn [24]. Initially, we applied regressive networks to extract the fast ion density parameter from the synthetic spectra. The regressors developed in configuration B provided promising results in that aspect, but the achieved inference was not sufficiently accurate. Nevertheless this study demonstrated the importance of the scattering geometry in our endeavour.

Since fast ion inference with regression was not accurate enough, we investigated the possibility of fast ion detection through classification. For this purpose, we created a database of spectra where the fast ion component was either present or absent and assigned the appropriate labels. The performance of our classifiers in geometry configuration B reached very high accuracy ($\sim 97\%$) in label assignment to the spectra. This result is also the main conclusion of this research project: We showed that fast ion detection in CTS spectra is feasible with the application of neural networks. Furthermore the signal-to-noise ratio of the CTS signal with respect to the fast ion component can be optimized with data analysis procedures. However, the most influential factor is the scattering geometry of the measurements.

Although there is no previous classification study in the field of CTS research, the achieved accuracy levels are obviously very high. This is even more evident by the small amount of mislabeled spectra (Table:6). This research showed that neural networks can work as fast filters of CTS data and distinguish the plasma shots where the fast ion component is observable. This ability is particularly useful considering the large amount of data that result from a fusion campaign. Additionally, neural networks can fill in a second role related to fast ion detection. The results of this research in the two examined scattering geometries highlight the significance of this experimental parameter. In that scope, neural networks can be employed in the search of a scattering geometry optimized for fast ion detection. Certainly, this type of investigation should also take into consideration any limitations that may exist (*e.g.* diagnostic ports availability, magnetic field topology) for the fusion device in question. In a more general view, this research project demonstrated that neural networks can contribute in fusion research.

However, we should mention the limitations of the presented work. Firstly, our code simulates the spectral density function and not the power density CTS spectrum. This means that calculations regarding ray-tracing, beam overlap and coupling of polarization modes are not included. All these factors are needed in order to compare the synthetic spectra with experimental measurements. Furthermore, this code is developed in the electrostatic approximation which is limited by the coupling of longitudinal and transverse modes in a magnetized plasma [18].

Lastly, we need to address the "loose ends" of this research project. The result that requires further investigation is the occurrence of over-fitting behaviour in the networks' performance. Particularly, the connection of this feature with the noise addition in CTS spectra. Considering that the signal-to-noise ratio is one of the main difficulties faced by CTS, improving the noise robustness of the neural networks which analyze the data has paramount significance. The first step towards that direction would be identifying the source of the over-fitting behaviour. This could be achieved with a cross-validation algorithm.

# 8 Future work options

The research presented in this thesis provided valuable results but leaves room for improvement. This section deals with these potential improvements and possible expansions of the work.

## 8.1 Model expansion

In the development of our forward model we assumed that the plasma did not include any impurities. This assumption was implemented by calculating the electron density as the sum of the bulk and fast ion density. The latter is an expression of the quasi-neutrality condition since the deuterium ions' charge is one. However, in a realistic scenario, the plasma would contain impurities. These impurities would not only influence the electron density but also contribute to the spectrum. For example, if the model assumed carbon impurities with a Maxwellian distribution, their contribution to the spectrum would be significant due to their higher electric charge [9] [40]. If the velocity distributions of the impurity species is known, their inclusion to the current forward model can be easily implemented.

Another scenario that can be investigated is the simulation of a fusion reactor plasma. More specifically, the bulk population could be assigned the effective mass of deuterium and tritium and alpha particles could be used as the fast ions species like in [19]. Similarly to the impurities case, the higher electric charge of the alpha particles would increase their contribution to the spectrum and therefore make their detection easier [35].

Additionally, the influence of the lower hybrid resonance on the fast ion detection ability of the networks could be further investigated with the current model. Two new databases should be developed: one in a subspace of parameter space B where the resonance is always present, and one where the resonance is outside the scattered spectrum bandwidth.

Furthermore, the model could be expanded to simulate the spectral density of fast ions who do not have a slowing-down distribution. As an example, fast ions created with ICRH could be studied. Since analytical expressions of the spectral density function for these cases may not be available, this task could be pursued through the polynomial approximation that was described in Section:4.1. If that approximation is successfully implemented, then investigation of distributions which are numerically calculated by transport codes would also be possible. However, a modified version of this approximation, capable to handle cut-offs and resonances in the spectrum, may be required.

Lastly, the forward model, either with the polynomial approximation approach or with the slowing-down distribution case, could be expanded to consider fluctuations of the plasma's dielectric properties besides the electron density.

## 8.2 Improvement of the neural network implementation

Another aspect of the project that is available for improvement is connected with the neural network implementation. Particularly, there were cases where the network exhibited signs of over-fitting to data. This could be investigated by applying a cross-validation algorithm to the networks, for both classifiers and regressors [41]. In most cases, the occurrence of over-fitting accompanied the noise addition to the spectra. Resolution of the over-training issue will probably improve the fault tolerance of the estimators. Another investigation that could be conducted is the replacement of Gaussian noise signals with the expected ECE signals. The latter is the main source of noise in a CTS experiment performed in the microwave range and therefore would bring the simulated

spectra closer to an experimental case. Further optimization of the developed estimators could be conducted by performing a grid search in the network's hyper-parameter space. A grid search is used to tune the hyper-parameters of an estimator in order to optimize their cross-validation score [24].

The fast ion density is not a parameter of great interest by itself but it is a very clear indication for the presence of fast ions. The regressors developed in this thesis project were not able to infer this parameter with sufficient accuracy. Sensitivity analysis on the influence of the modelling parameters to the regressor performance can perhaps explain this result. Based on such a sensitivity analysis, reduction of the number inferred parameters may prove beneficial. Such an analysis could also give insight on the influence of the investigated parameter range and lead to the optimization of the scattering geometry parameters. The latter is a factor with significant influence on the performance of the networks.

Furthermore, the performance of the estimators could be improved by an educated initialization of the weights between the first hidden and input layer. Considering that the artifacts revealing the fast ion presence are expected away from the probing frequency, focusing the network on those spectral regions could be implemented by enhanced weight initialization of the representative neurons. Lastly, it could be interesting to see how the network performs on a set of spectra with a varied noise levels since the noise contribution is not generally constant in an experiment.

## 8.3 Further options

Follow-up research options of a more generalized nature are also available. For example, fast ion detection through regression may be more accurate for a different parameter than the fast ion density. This new parameter is not necessarily one of the fundamental modelling parameters that were used to simulate the spectrum. It is probable that a combination of those parameters or a new one based on the spectral characteristics could provide more accurate results.

If a forward model for different fast ion distributions is developed, then a network could be trained to provide classification for the different types of the distributions. In that case the network will not act as a filter for fast ion presence but as a tool for the inference of the species' distribution. Such a task would probably require deeper neural networks and larger training databases. The need for larger and more diverse dataset could be satisfied through data augmentation [27] or a more appropriate selection of data, based on number-theoretic methods (NTMs) [42].

Last but not least, a different type of neural networks than multilayer perceptrons could be applied. Radial Basis Functions [43], Hopfield Networks [44] could be some interesting options while remaining in the supervised learning domain. Restricted Boltzmann Machines (RBM) [45] are a type of networks similar to Hopfield networks but in the unsupervised learning domain which could also be a suitable choice for the network implementation. Considering the vast amount of possible network configurations, other options could be more or less suited for fast ion detection in CTS spectra.

# 9 Conclusion

The aim of this research project was to investigate the feasibility of fast ion detection in Collective Thomson Scattering spectra with neural networks. Towards that goal: we developed a forward model code for the simulation of the spectral density function, we created databases of synthetic spectra in two geometry configurations (configuration A simulates the W 7-X conditions and configuration B was inspired by related literature research) and we implemented and trained neural networks in both configurations. We examined the feasibility of fast ion inference with classifiers and regressors. The regression approach gave promising results in configuration B but the inference of fast ions was not sufficiently accurate. The classifiers in the same configuration exhibited very high accuracy and proved that neural networks can operate as fast filters of CTS data for the detection of fast ions. The next step in this research should be the improvement of the estimators' fault tolerance. In conclusion, we showed that neural networks can detect fast ions in CTS spectra and by extent contribute to fusion research. Future research on the field of nuclear fusion should take advantage of the opportunities offered by the application of neural networks.

# Part IV
# Appendix

## 10 Fast Ion Susceptibility

In the forward model where the fast ion population has a slowing-down velocity distribution, the species' susceptibility is calculated as follows [20]:

$$G_a(\mathbf{k},\omega) = \frac{-2\pi\omega_{pa}^2}{k^2}\left(\frac{2F_0 v_a}{v_a^3+v_c^3} - \frac{2F_0}{3v_a^2}\left(0.5\ln\left(\frac{(v_a+v_c)^2}{v_a^2-v_a v_c+v_c^2}\right)+\sqrt{3}\arctan\left(\frac{\sqrt{3}v_A}{2v_c-v_a}\right)\right)\right)$$
$$+\frac{\omega F_0}{k(v_a^3+v_c^3)}\left(\ln\left(\frac{kv_a-\omega}{kv_a+\omega}\right)+i\pi\right) - i\pi\frac{\omega F_0}{k(\omega/k)^3+v_c^3}$$
$$-\frac{\omega}{k}P\int_0^{v_a} dv\frac{f_{0a}}{v-\omega/k} + \frac{\omega}{k}P\int_0^{v_a} dv\frac{f_{0a}}{v+\omega/k} \quad (35)$$

The Cauchy principal value integrals are:

$$P\int_0^{v_a} dv\frac{f_{0a}}{v\pm\omega/k} = \frac{F_0}{v_c^3}\Bigg[A_\pm\ln(1+\frac{v_a}{v_c}) + B_\pm\ln(1+\frac{k}{\omega}v_a) + 0.5C_\pm\ln(\frac{v_a^2}{v_c^2}-\frac{v_a}{v_c}+1)$$
$$+\frac{2}{\sqrt{3}}(D_\pm+0.5C_\pm)\left(\arctan(\frac{2\frac{v_a}{v_c}-1}{\sqrt{3}})+\arctan(\frac{1}{\sqrt{3}})\right)\Bigg] \quad (36)$$

,where:

$$a_\pm = \pm\frac{\omega}{kv_c}$$
$$A_\pm = (3(a_\pm-1))^{-1}$$
$$B_+ = -(a_+^3-1)^{-1}$$
$$B_- = \frac{\ln(\frac{k}{\omega}v_a-1)}{(a_-^3-1)\ln(\frac{k}{\omega}v_a+1}$$
$$C_\pm = \frac{a_\pm+2}{3(a_\pm^2+a_\pm+1)}$$
$$D_\pm = -\frac{2a_\pm+1}{a_\pm+2}C_\pm$$

# 11 Scikit-learn MLP parameters

This section contains the full list of the parameters that were used to implement the multilayer perceptron regressors (Table:11) and classifiers (Table:12) according to scikit-learn library [24]. The explanation of the parameters that were assigned values different than their default can be found in Sections 5.1.3 and 5.2.3 for each type of network respectively.

Table 11: Full list of the parameters that were used in the implementation of MLP regressors through the module [24]. The symbols $N_1$, $N_2$ and $N_3$ correspond to the number of neurons in each hidden layer.

| Parameter name | Parameter value |
|---|---|
| *hidden_layer_size* | ($N_1$,$N_2$,$N_3$) |
| *activation* | "logistic" |
| *solver* | "adam" |
| *alpha* | 5 $10^{-4}$ |
| *batch_size* | 250 |
| *learning_rate* | "constant" |
| *learning_rate_init* | 0.001 |
| *power_t* | 0.5 |
| *max_iter* | 1 |
| *shuffle* | True |
| *random_state* | None |
| *tol* | $10^{-4}$ |
| *verbose* | False |
| *warm_start* | True |
| *momentum* | 0.9 |
| *nesterovs_momentum* | True |
| *early_stopping* | False |
| *validation_fraction* | 0.1 |
| *beta_1* | 0.9 |
| *beta_2* | 0.99 |
| *epsilon* | $10^{-8}$ |
| *n_iter_no_change* | 10 |

Table 12: Full list of the parameters that were used in the implementation of MLP classifiers through the module [24]. The symbols $N_1$, $N_2$ and $N_3$ correspond to the number of neurons in each hidden layer.

| Parameter name | Parameter value |
|---|---|
| *hidden_layer_size* | ($N_1$,$N_2$,$N_3$) |
| *activation* | "relu" |
| *solver* | "adam" |
| *alpha* | 5 $10^{-4}$ |
| *batch_size* | 250 |
| *learning_rate* | "constant" |
| *learning_rate_init* | 0.001 |
| *power_t* | 0.5 |
| *max_iter* | 1 |
| *shuffle* | True |
| *random_state* | None |
| *tol* | $10^{-4}$ |
| *verbose* | False |
| *warm_start* | True |
| *momentum* | 0.9 |
| *nesterovs_momentum* | True |
| *early_stopping* | False |
| *validation_fraction* | 0.1 |
| *beta_1* | 0.9 |
| *beta_2* | 0.99 |
| *epsilon* | $10^{-8}$ |
| *n_iter_no_change* | 10 |

# 12 Forward model code

1) Forward model method:

```
import scipy.constants as ct
import math
import numpy as np
import scipy.special as spc
import FastIonFunctions as func

def FastIonModel(T_e,T_i,E_a,n_i,n_a,phi,theta):

#%%
    # Declaration of physical, mathematical and unit conversion constants
    e = ct.elementary_charge
    eps_0 = ct.epsilon_0
    c = ct.physical_constants['speed of light in vacuum'][0]
    energy = ct.physical_constants['electron volt-joule relationship'][0]
    temperature = ct.physical_constants['electron volt-kelvin relationship'][0]
    m_e = ct.electron_mass
    m_D = ct.physical_constants['deuteron mass'][0]
    pi = ct.pi
    k_B = ct.Boltzmann

    #Definition of the plasma cmposition
    m_i = m_D                                   # Bulk ion species
    m_a = m_D                                   # Fast ion species
    Z_i = 1                                     # Bulk ion effective charge
    Z_a = 1                                     # Fast ion effective charge

#%%  Input parameters of the models

    #Plasma parameters
    T_e = T_e*10**3*temperature         # Electron Temperature [K] (input in eV)
    T_i = T_i*10**3*temperature         # Bulk ion Temperature [K] (input in eV)
    E_a = E_a*10**3*energy              # Fast ion energy [J] (input in eV)

    n_i = n_i*10**19                            # Bulk ion Density [m^-3]
    n_a = n_a*10**15                            # Fast Ion Density [m^-3]

    # Geometry parameters
    theta = theta*pi/180            # Scattering angle (transformed in radians)
    phi   = phi*pi/180                # Resolved angle (transformed in radians)

#%%
    #Addtional experimental parameters
    omega_i = 2*pi*140 *10**9               # Incident frequeny [rad/s]
```

```
    B = 2.2                                 # Magnetic field [T]

    #Derived parameters
    n_e = Z_i*n_i +Z_a*n_a                  # Electron density calculation


    v_e = math.sqrt(2*k_B*T_e/m_e)          # Electron thermal velocity [m/s]
    v_i = math.sqrt(2*k_B*T_i/m_i)          # Ion thermal velocity [m/s]
    v_a = math.sqrt(2*E_a/m_a)              # Fast ion "thermal" velocity [m/s]

    # Electron plasma frequency [rad/s]
    omega_pe = math.sqrt(e**2*n_e/(m_e*eps_0))

    # Fast ion plasma frequency [rad/s]
    omega_pa = Z_a*e*math.sqrt(n_a)/math.sqrt(m_a*eps_0)

    omega_ce = e*B/m_e                    # Electron cyclotron frequency [rad/s]
    omega_ci = Z_i*e*B/m_i                   # Ion cyclotron frequency [rad/s]

    rho_e = v_e/(math.sqrt(2)*omega_ce)     # Electron gyroradius [m]
    rho_i = v_i/(math.sqrt(2)*omega_ci)     # Ion gyroradius [m]

    lambda_De = math.sqrt(eps_0*k_B*T_e/(n_e*e**2))     # Debye length [m]

    k_i = math.sqrt((omega_i**2-omega_pe**2)/c**2) # Incident wavevector[1/m]
    k_d = 2*k_i*math.sin(theta/2)                  # Fluctuation wavevector[1/m]
    k_par = (k_d* math.cos(phi))          # Projection prallel to B-field
    k_per = (k_d* math.sin(phi))          # Projection perpendicular to B-field
    a = 1/(k_d*lambda_De)                 # Salpeter parameter

    v_c = 0.09*v_e
    om =[136,144,200]              #Sampling frequencies (Bandwidth,Resolution)

    omega_s = 2*pi*np.linspace(om[0],om[1],om[2])*10**9   # Scattered frequency

    omega = omega_s-omega_i                 #Resolved frequency spectrum
    omega_range = np.size(omega)

    l_max = 100   # Maximum order of summation for the susceptibility fucntions

#%%  Calculation of the susceptibility and slowing down functions
    # Electron susceptibility
    H_e = func.H_j(omega,l_max,k_per,k_par,rho_e,omega_ce,v_e,a)

    # Buslk ion susceptibility
    H_i = func.H_j(omega,l_max,k_per,k_par,rho_i,omega_ci,v_i,a)
```

```
    H_i = math.pow(Z_i,2)*(n_i*T_e/(n_e*T_i))*H_i

    # Fast ion susceptibility
    omega1 = abs(omega)       # Use of absolute value to avoid numerical errors
    G_a = func.G_a(omega1,k_d,v_c,v_a,omega_pa)

    # Dielectric function
    eps_L = 1 +H_i +H_e +G_a

    # Slowing down velocity distribution
    f_a = func.f_a(omega1,k_d,v_a,v_c)

#%% Calculation of the contributions to the spectral density function

     #Electron spectral density function according to Equation (29)

    S_e =np.zeros_like(omega)
    C_e = 2*math.sqrt(pi)/(abs(k_par)*v_e)*math.exp(-math.pow(k_per*rho_e,2))
    for w in range(0,omega_range):
        S=0
        for l in range(-l_max,l_max):
            S+=math.exp(-math.pow((omega[w]-l*omega_ce)/(k_par*v_e),2)) \
            *spc.iv(l,math.pow(k_per*rho_e,2))
        S_e[w] = math.pow(abs(1-H_e[w]/eps_L[w]),2)*S
    S_e =C_e*S_e

    # Bulk ion spectral density function according to Equation (30)

    S_i =np.zeros_like(omega)
    C_i = 2*math.sqrt(pi)*math.pow(Z_i,2)*n_i/(abs(k_par)*v_i*n_e) \
    *math.exp(-math.pow(k_per*rho_i,2))
    for w in range(0,omega_range):
        S=0
        for l in range(-l_max,l_max):
            S+=math.exp(-math.pow((omega[w]-l*omega_ci)/(k_par*v_i),2)) \
            *spc.iv(l,math.pow(k_per*rho_i,2))
        S_i[w] = math.pow(abs(H_e[w]/eps_L[w]),2)*S
    S_i =C_i*S_i

    # Fast ion spectral density function
    S_a = func.SPDF(omega,H_e,eps_L,f_a,n_a,Z_a,n_e,k_d)

    # Total spectral density function

    S_tot = S_e+ S_i+ S_a
    f = omega/(2*pi)            # Resolved frequency vector
```

```
136
137     return f,S_tot,a
```

Listing 1: Forward model method: This method simulates the spectrum with the slowing down model.

2) Fast Ion functions:

```
1  import scipy.constants as ct
2  import math
3  import numpy as np
4  import scipy.special as spc
5  import FastIonFunctions as func
6
7  def FastIonModel(T_e,T_i,E_a,n_i,n_a,phi,theta):
8
9  #%%
10     # Declaration of physical, mathematical and unit conversion constants
11     e = ct.elementary_charge
12     eps_0 = ct.epsilon_0
13     c = ct.physical_constants['speed of light in vacuum'][0]
14     energy = ct.physical_constants['electron volt-joule relationship'][0]
15     temperature = ct.physical_constants['electron volt-kelvin relationship'][0]
16     m_e = ct.electron_mass
17     m_D = ct.physical_constants['deuteron mass'][0]
18     pi = ct.pi
19     k_B = ct.Boltzmann
20
21     #Definition of the plasma cmposition
22     m_i = m_D                                  # Bulk ion species
23     m_a = m_D                                  # Fast ion species
24     Z_i = 1                                    # Bulk ion effective charge
25     Z_a = 1                                    # Fast ion effective charge
26
27 #%%  Input parameters of the models
28
29     #Plasma parameters
30     T_e = T_e*10**3*temperature        # Electron Temperature [K] (input in eV)
31     T_i = T_i*10**3*temperature        # Bulk ion Temperature [K] (input in eV)
32     E_a = E_a*10**3*energy             # Fast ion energy [J] (input in eV)
33
34     n_i = n_i*10**19                       # Bulk ion Density [m^-3]
35     n_a = n_a*10**15                       # Fast Ion Density [m^-3]
36
37     # Geometry parameters
38     theta = theta*pi/180            # Scattering angle (transformed in radians)
```

```
    phi   = phi*pi/180                # Resolved angle (transformed in radians)

#%%
    #Addtional experimental parameters
    omega_i = 2*pi*140 *10**9               # Incident frequeny [rad/s]
    B = 2.2                                 # Magnetic field [T]

    #Derived parameters
    n_e = Z_i*n_i +Z_a*n_a                  # Electron density calculation


    v_e = math.sqrt(2*k_B*T_e/m_e)          # Electron thermal velocity [m/s]
    v_i = math.sqrt(2*k_B*T_i/m_i)          # Ion thermal velocity [m/s]
    v_a = math.sqrt(2*E_a/m_a)              # Fast ion "thermal" velocity [m/s]

    # Electron plasma frequency [rad/s]
    omega_pe = math.sqrt(e**2*n_e/(m_e*eps_0))

    # Fast ion plasma frequency [rad/s]
    omega_pa = Z_a*e*math.sqrt(n_a)/math.sqrt(m_a*eps_0)

    omega_ce = e*B/m_e                   # Electron cyclotron frequency [rad/s]
    omega_ci = Z_i*e*B/m_i                  # Ion cyclotron frequency [rad/s]

    rho_e = v_e/(math.sqrt(2)*omega_ce)     # Electron gyroradius [m]
    rho_i = v_i/(math.sqrt(2)*omega_ci)     # Ion gyroradius [m]

    lambda_De = math.sqrt(eps_0*k_B*T_e/(n_e*e**2))     # Debye length [m]

    k_i = math.sqrt((omega_i**2-omega_pe**2)/c**2) # Incident wavevector[1/m]
    k_d = 2*k_i*math.sin(theta/2)                  # Fluctuation wavevector[1/m]
    k_par = (k_d* math.cos(phi))          # Projection prallel to B-field
    k_per = (k_d* math.sin(phi))          # Projection perpendicular to B-field
    a = 1/(k_d*lambda_De)                 # Salpeter parameter

    v_c = 0.09*v_e
    om =[136,144,200]              #Sampling frequencies (Bandwidth,Resolution)

    omega_s = 2*pi*np.linspace(om[0],om[1],om[2])*10**9   # Scattered frequency

    omega = omega_s-omega_i                 #Resolved frequency spectrum
    omega_range = np.size(omega)

    l_max = 100   # Maximum order of summation for the susceptibility fucntions

#%%  Calculation of the susceptibility and slowing down functions
```

```
    # Electron susceptibility
    H_e = func.H_j(omega,l_max,k_per,k_par,rho_e,omega_ce,v_e,a)

    # Buslk ion susceptibility
    H_i = func.H_j(omega,l_max,k_per,k_par,rho_i,omega_ci,v_i,a)
    H_i = math.pow(Z_i,2)*(n_i*T_e/(n_e*T_i))*H_i

    # Fast ion susceptibility
    omega1 = abs(omega)       # Use of absolute value to avoid numerical errors
    G_a = func.G_a(omega1,k_d,v_c,v_a,omega_pa)

    # Dielectric function
    eps_L = 1 +H_i +H_e +G_a

    # Slowing doum velocity distribution
    f_a = func.f_a(omega1,k_d,v_a,v_c)

#%% Calculation of the contributions to the spectral density function

     #Electron spectral density function according to Equation (29)

    S_e =np.zeros_like(omega)
    C_e = 2*math.sqrt(pi)/(abs(k_par)*v_e)*math.exp(-math.pow(k_per*rho_e,2))
    for w in range(0,omega_range):
        S=0
        for l in range(-l_max,l_max):
            S+=math.exp(-math.pow((omega[w]-l*omega_ce)/(k_par*v_e),2)) \
            *spc.iv(l,math.pow(k_per*rho_e,2))
        S_e[w] = math.pow(abs(1-H_e[w]/eps_L[w]),2)*S
    S_e =C_e*S_e

    # Bulk ion spectral density function according to Equation (30)

    S_i =np.zeros_like(omega)
    C_i = 2*math.sqrt(pi)*math.pow(Z_i,2)*n_i/(abs(k_par)*v_i*n_e) \
    *math.exp(-math.pow(k_per*rho_i,2))
    for w in range(0,omega_range):
        S=0
        for l in range(-l_max,l_max):
            S+=math.exp(-math.pow((omega[w]-l*omega_ci)/(k_par*v_i),2)) \
            *spc.iv(l,math.pow(k_per*rho_i,2))
        S_i[w] = math.pow(abs(H_e[w]/eps_L[w]),2)*S
    S_i =C_i*S_i

    # Fast ion spectral density function
    S_a = func.SPDF(omega,H_e,eps_L,f_a,n_a,Z_a,n_e,k_d)

```

```
    # Total spectral density function

    S_tot = S_e+ S_i+ S_a
    f = omega/(2*pi)            # Resolved frequency vector

    return f,S_tot,a
```

Listing 2: Fast Ion functions: This script contains the functions that are used for the simulation of the spectrum by the forward model method.

3) Database development code:

```
import numpy as np
import ForwardModel as fwd
import os
import time


#%% Parameter range + database directory definition
path =""
DatabaseFolder = "C:/Users/Chris/Desktop/Uni/Thesis/CTS/Code/Database"


parameter_names = ['T_e','T_i','E_a','n_i','n_a','phi','theta']

prange= [[1, 1, 40,1, 0, 95,10],
         [10,10,50,10,3,115,30]]
Domain =""
for i in range(len(parameter_names)):
  Domain +=parameter_names[i]+str(prange[0][i])+"-"+str(prange[1][i])+" "

N =  1                                          # Number of spectra to simulate



path = DatabaseFolder+"/"+Domain
path = path[:-1]+" Classifier"

#%% Selection of the parameter values from a normal distribution

# Fraction of spectra without the fast ion component for the classification case
percent = 0.5
fraction = int(N*percent)


T_e_lim = np.array([prange[0][0], prange[1][0]])            # in keV
```

```
T_i_lim = np.array([prange[0][1], prange[1][1]])            # in keV
E_a_lim = np.array([prange[0][2], prange[1][2]])            # in keV
n_i_lim = np.array([prange[0][3], prange[1][3]])            # in 10^19 m^-3
n_a_lim = np.array([prange[0][4], prange[1][4]])            # in 10^15 m^-3
phi_lim = np.array([prange[0][5], prange[1][5]])            # in degrees
theta_lim = np.array([prange[0][6], prange[1][6]])          # in degrees


T_e = np.random.rand(N)*(T_e_lim[1]-T_e_lim[0])+T_e_lim[0]
T_i = np.random.rand(N)*(T_i_lim[1]-T_i_lim[0])+T_i_lim[0]
E_a = np.random.rand(N)*(E_a_lim[1]-E_a_lim[0])+E_a_lim[0]
#n_e = np.random.rand(N)*(n_e[1]-n_e[0])+n_e[0]
n_i = np.random.rand(N)*(n_i_lim[1]-n_i_lim[0])+n_i_lim[0]
n_a = np.random.rand(fraction)*(n_a_lim[1]-n_a_lim[0])+n_a_lim[0]
#n_a = np.random.rand(N_a)*(n_a[1]-n_a[0])+n_a[0]
phi = np.random.rand(N)*(phi_lim[1]-phi_lim[0])+phi_lim[0]
theta = np.random.rand(N)*(theta_lim[1]-theta_lim[0])+theta_lim[0]

n_a0 = np.zeros(N-fraction) # Fraction where fast ion density is zero
n_a=np.append(n_a,n_a0,axis=0) # Fraction where fast ion density is non-zero

np.random.shuffle(n_a)

#%%
start1 =time.time()
for i in range(N):
    start2 = time.time()
    p= np.array([T_e[i],T_i[i],E_a[i],n_i[i],n_a[i],phi[i],theta[i]]).reshape(1,-1)
    w,S_tot,a = fwd.FastIonModel(T_e[i],T_i[i],E_a[i],n_i[i],n_a[i],phi[i],theta[i])

    S_tot= S_tot.reshape(1,-1)

    # Label assignment for the classification case
    if n_a[i]==0:
        flag_curr="Absence"
    else:
        flag_curr="Presence"

    print("Salpeter is ",a)
    print("Spectra calculated so far:",i+1,) # Message to follow the progress
    end2 =time.time()
    print("Loop time:",end2-start2," seconds")

    if i==0:
        resolution = np.size(w)
        flag=np.empty((N,1),dtype='<U8') # Only for classifier
```

```
        flag[i]=flag_curr                # Only for classifier
        S = np.zeros_like(S_tot)
        param = np.zeros_like(p)
        path +=" resol_"+str(resolution)
        # -- Creating or (appending to) the files containing the spectra (S),
        #the parameter (p), the frequencies (w) and the labels
        #for classification (flag)
    if os.path.isdir(path):
        flag[i]= flag_curr            # Only for classifier
        SPDF1   = S_tot
        param  = np.append(param,p,axis=0)
        S       = np.append(S,SPDF1,axis=0)
#        np.savetxt(path+'/'+'p.txt',param,delimiter='\t')
#        np.savetxt(path+'/'+'S.txt',S,delimiter='\t')
    else:
        os.makedirs(path)
        np.savetxt(path+'/p.txt',p,delimiter='\t')
        np.savetxt(path+'/S.txt',S_tot,delimiter='\t')
        np.savetxt(path+'/w.txt',w,delimiter='\t')
        # Only for classifier
        np.savetxt(path+'/flag.txt',flag,delimiter='\t',fmt="%s")

if os.path.isfile(path+'/'+'p.txt'):
    p_old = np.loadtxt(path+'/'+'p.txt').reshape(-1,len(parameter_names))
    param = np.append(param,p_old,axis=0)
if os.path.isfile(path+'/'+'S.txt'):
    S_old = np.loadtxt(path+'/'+'S.txt').reshape(-1,resolution)
    S = np.append(S,S_old,axis=0)


# Classifier case
if os.path.isfile(path+'/'+'flag.txt'):
    flag_old = np.loadtxt(path+'/'+'flag.txt',dtype='<U8').reshape(-1,1)
    flag = np.append(flag,flag_old,axis=0)

#%% Final saving process

np.savetxt(path+'/'+'p.txt',param[1::,:],delimiter='\t')
np.savetxt(path+'/'+'S.txt',S[1::,:],delimiter='\t')

# Only for classifier
np.savetxt(path+'/'+'flag.txt',flag,delimiter='\t',fmt="%s")

end1 =time.time()
print("Total time:",end1-start1," seconds")
```

Listing 3: Database development: This script contains the shell that was used to create the database by calling the forward model method in the two parameter spaces.

# 13 Neural networks code

1) Regression network training code:

```
import os,time
import numpy as np
from sklearn.neural_network import MLPRegressor
import ANNFunctions as funcs
import pickle
from sklearn.preprocessing import StandardScaler

#%% Database loading

# Definition of the database for the network's training
DatabaseFolder = "C:/Users/Chris/Desktop/Uni/Thesis/CTS/Code/Database"

parameter = ['T_e','T_i','E_a','n_i','n_a','phi','theta']

prange= [[1, 1,40,1,0,100,10],
             [10,10,50,10,3,120,30]]
Domain =""
for i in range(len(parameter)):
  Domain +=parameter[i]+str(prange[0][i])+"-"+str(prange[1][i])+" "


loadpath = DatabaseFolder+"/"+ Domain
loadpath = loadpath[:-1]                    # Removing the last space

resolution = 200                            # Number of sampling frequencies
loadpath += "resol_"+str(resolution)

# Loading of the spectra, the frequency range and the associated parameters

SPDF_raw = np.loadtxt(loadpath+"/S.txt")
param_raw = np.loadtxt(loadpath+"/p.txt")
freq_raw = np.loadtxt(loadpath+"/w.txt")

spec_range = int((freq_raw[-1]-freq_raw[0])/10**6)     # Spectral range in MHz

spec_dens = resolution/spec_range                   # Spectral density in MHz^-1
N_database  = SPDF_raw.shape[0]                     # Number of spectra
N_samples  = freq_raw.shape[0]     # Number of sampling points in each spectrum

```

```
#%% Data preparation ( Normalization and Noise addition)

# Data splitting in training set and test set
N_training = int(0.9*N_database)
N_test = int(0.1*N_database)

S_train = SPDF_raw[:N_training,:]
S_test  = SPDF_raw[N_training:N_training+N_test,:]

# Determine the parameters that have a finite range in the database
p_tomodel = list(np.where(np.std(param_raw,axis=0)>1E-6)[0])
N_p    = len(p_tomodel)               # Number of input parameters in the model

param      = np.zeros((param_raw.shape[0],N_p))
for i in range(len(p_tomodel)): param[:,i]=param_raw[:,p_tomodel[i]]
param_train = param[:N_training,:]
param_test  = param[N_training:N_training+N_test,:]

# Normalization of the spectra
scaler = StandardScaler()
scaler.fit(S_train)

S_train_sc = scaler.transform(S_train)
S_test_sc = scaler.transform(S_test)

## Noise addition
noiserel = 0                                # Noise level definition

# Adding noise to the training set
# Mean maximum spectral value serves as noise reference level
noiseref        = np.mean(np.max(S_train_sc,axis=1))
# Noise  amplitude is defined as fraction of average maximum spectral value
noise           = noiserel*noiseref
Addnoise_train        = np.random.randn(N_training,N_samples)*noise
S_train_sc         = S_train_sc + Addnoise_train

# Adding noise to the test set
# Mean maximum spectral value serves as noise reference level
noiseref        = np.mean(np.max(S_test_sc,axis=1))
# Noise  amplitude is defined as fraction of average maximum spectral value
noise           = noiserel*noiseref
Addnoise_test   = np.random.randn(N_test,N_samples)*noise
S_test_sc       = S_test_sc + Addnoise_test

#%% Network initialization

```

```
path =""
NetworkFolder = "C:/Users/Chris/Desktop/Uni/Thesis/CTS/Code/Network"

path = NetworkFolder+"/"+Domain
path = path[:-1]                                #Removing the last space

batch_size =250     # Setting the batch size for the training
N_epochs = 5        # Number of training epochs is defined explicitly

# Number of iterations for 1 training epoch
N_iter = int(N_training/batch_size)

# Defining the network's architecture
layerSizes=(150,75)

layerstr=''
for i in range(len(layerSizes)): layerstr += 'L'+str(layerSizes[i])
NetworkName     = 'NN_'+layerstr+'bs'+str(batch_size)+'n'+\
                  str(100*noiserel)

path = path+"resol_"+str(resolution)+NetworkName+"/"

#%% Network training

making, saving = 0,1

make=0
if making ==0:
    epochs_done = 0
    while os.path.isfile(path+'NN'+str(epochs_done+1)+'.txt'):
        epochs_done += 1

    if epochs_done > 0:
        print('Loading MLP, '+str(epochs_done)+' epochs were already completed')
        fileObject = open(path+'NN'+str(epochs_done)+'.txt','rb')
        mlp=pickle.load(fileObject)
        fileObject.close()
        scores=np.loadtxt(path+'scores'+str(epochs_done)+'.txt')
        errors=np.loadtxt(path+'errors_progress'+str(epochs_done)+'.txt')
        errorsabs=np.loadtxt(path+'errors'+str(epochs_done)+'Ntest'+str(N_test)\
                             +'.txt')
    else:
        print('Training network because not yet present ...')
        make=1
else:   make=1


```

```
if make ==1:
    p_range = (np.max(param,axis=0)-np.min(param,axis=0)).reshape(1,-1)

    Time_comp = 0
    Time_comp = S_train_sc.shape[1]*layerSizes[0]+S_test_sc.shape[1]\
                *layerSizes[0]
    Time_comp+= N_p*layerSizes[-1]
    for i in range(len(layerSizes)-1):
        Time_comp += layerSizes[i]*layerSizes[i+1]

    print("The time complexity of the network is: ",Time_comp)

     # Definition of the regressor parameters
    mlp = MLPRegressor(hidden_layer_sizes=layerSizes,max_iter=1,warm_start=True,\
                       activation='logistic',batch_size=batch_size,alpha=5e-4,\
                       learning_rate='adaptive',shuffle=True)

while epochs_done < N_epochs:

    print(' ')
    print('Working on epoch '+str(epochs_done+1)+'/'+str(N_epochs)+', '\
          +str(batch_size)+' per batch, '+str(N_iter)+' iterations')

    scores      = np.zeros((N_iter,2))
    errors      = np.zeros((N_iter,N_p))
    errorsrel   = np.zeros((N_iter,N_p))
    times       = np.zeros(N_iter)
    time0       = time.time()
    for i in range(N_iter):
        mlp.fit(S_train_sc,param_train) # Fitting to training set
        param_inv   = mlp.predict(S_test_sc) # Performance on test set

        # Defining the score and error metrics of the regressor
        errors[i,:] = np.mean(np.abs((param_inv-param_test)),axis=0)/p_range
        errorsrel[i,:]=np.mean(np.abs((param_inv-param_test)/param_test),axis=0)
        scores[i,0] = mlp.score(S_train_sc,param_train)
        scores[i,1] = mlp.score(S_test_sc,param_test)
        # Time estimation until training is complete
        ETA         = (N_iter/(i+1)-1)*(time.time()-time0)
        if (i+1) % 10 == 0:
            print('# '+str(i+1)+'/'+str(N_iter)+', $R^2$ = '\
                  +str("%0.2f" % scores[i,1])+\
            ', error n_a = '+str("%0.2f" % errorsrel[i,4])\
            +', ETA '+str("%0.0f"%ETA)+' s')
            print("Loss function:" ,mlp.loss_)
    epochs_done += 1
    errorsabs=np.abs((param_inv-param_test)/param_test)
```

```
180
181     # Saving of the network and its performance metrics
182     funcs.SaveNN(mlp,scores,errors,errorsabs,errorsrel,path,N_test,epochs_done)
183 else:
184     param_inv    = mlp.predict(S_test_sc)
```

Listing 4: Regressor training: This script makes the initialization and training of a regressor. It contains the loading of the database, the data preparation (normalization and noise addition), the training of the network and the saving of its performance metrics.

2) Classification network training code:

```
1  import os,time
2  import numpy as np
3  from sklearn.neural_network import MLPClassifier
4  import ANNFunctions as funcs
5  import pickle
6  from sklearn.preprocessing import StandardScaler
7  from sklearn.metrics import classification_report,matthews_corrcoef,\
8  confusion_matrix
9
10
11 #%% Database loading
12
13 # Definition of the database for the network's training
14 DatabaseFolder = "C:/Users/Chris/Desktop/Uni/Thesis/CTS/Code/Database"
15
16 parameter = ['T_e','T_i','E_a','n_i','n_a','phi','theta']
17
18 prange= [[1, 1,40,1,0,95,10],
19          [10,10,50,10,3,115,30]]
20 Domain =""
21 for i in range(len(parameter)):
22   Domain +=parameter[i]+str(prange[0][i])+"-"+str(prange[1][i])+" "
23
24
25 loadpath = DatabaseFolder+"/"+ Domain
26 loadpath = loadpath[:-1] +" Classifier"               #Removing the last space
27
28 resolution = 200    # Number of sampling frequencies
29 loadpath += " resol_"+str(resolution)
30
31  # Loading of the spectra, the frequency range and the associated labels
32 SPDF_raw = np.loadtxt(loadpath+"/S.txt")
33 flag = np.loadtxt(loadpath+"/flag.txt",dtype='<U8')
34 freq_raw = np.loadtxt(loadpath+"/w.txt")
```

```
35
36  spec_range = int((freq_raw[-1]-freq_raw[0])/10**6)      # Spectral range in MHz
37
38  spec_dens = resolution/spec_range                   # Spectral density in MHz^-1
39  N_database  = SPDF_raw.shape[0]                     # Number of spectra
40  N_samples  = freq_raw.shape[0]       # Number of sampling points in the spectra
41
42  #%%
43  # Data splitting in training set and test set
44  N_training = int(0.9*N_database)
45  N_test = int(0.1*N_database)
46
47  S_train = SPDF_raw[:N_training,:]
48  S_test  = SPDF_raw[N_training:N_training+N_test,:]
49
50  flag_train = flag[:N_training]
51  flag_test = flag[N_training:N_training+N_test]
52
53
54  # Normalization of the spectra
55  scaler = StandardScaler()
56  scaler.fit(S_train)
57
58  S_train_sc = scaler.transform(S_train)
59  S_test_sc = scaler.transform(S_test)
60
61  ## Noise addition
62  noiserel = 0                                # Noise level definition
63
64  # Adding noise to the training set
65  # Mean maximum spectral value serves as noise reference level
66  noiseref        = np.mean(np.max(S_train_sc,axis=1))
67  # Noise  amplitude is defined as fraction of average maximum spectral value
68  noise           = noiserel*noiseref
69  Addnoise_train        = np.random.randn(N_training,N_samples)*noise
70  S_train_sc         = S_train_sc + Addnoise_train
71
72  # Adding noise to the test set
73  # Mean maximum spectral value serves as noise reference level
74  noiseref        = np.mean(np.max(S_test_sc,axis=1))
75  # Noise  amplitude is defined as fraction of average maximum spectral value
76  noise           = noiserel*noiseref
77  Addnoise_test   = np.random.randn(N_test,N_samples)*noise
78  S_test_sc       = S_test_sc + Addnoise_test
79
80  #%% Network initialization
81
```

```
path =""
NetworkFolder = "C:/Users/Chris/Desktop/Uni/Thesis/CTS/Code/Network"

path = NetworkFolder+"/"+"Classifiers/" +Domain
path = path[:-1]                                #Removing the last space

batch_size =250      # Setting the batch size for the training
N_epochs = 5         # Number of training epochs is defined explicitly

# Number of iterations for 1 training epoch
N_iter = int(N_training/batch_size)

# Defining the network's architecture
layerSizes=(150,75)


layerstr=''
for i in range(len(layerSizes)): layerstr += 'L'+str(layerSizes[i])
NetworkName     = 'Classifier_'+layerstr+'bs'+str(batch_size)+'n'+\
                   str(100*noiserel)#+'/'

path = path+"resol_"+str(resolution)+NetworkName+"/"
#%% Network training

making, saving = 0,1

make=0
if making ==0:
    epochs_done = 0
    while os.path.isfile(path+'NN'+str(epochs_done+1)+'.txt'):
        epochs_done += 1

    if epochs_done > 0:
        print('Loading MLP, '+str(epochs_done)+' epochs were already completed')
        fileObject = open(path+'NN'+str(epochs_done)+'.txt','rb')
        mlp=pickle.load(fileObject)
        fileObject.close()
        scores=np.loadtxt(path+'scores'+str(epochs_done)+'.txt')

    else:
        print('Training network because not yet present ...')
        make=1
else:   make=1


if make == 1:
    # Definition of the classifier parameters
```

```
    mlp = MLPClassifier(hidden_layer_sizes=layerSizes,max_iter=1,\
          warm_start=True,activation='relu',batch_size=batch_size,alpha=5e-4,\
          shuffle=True)

while epochs_done < N_epochs:

    print(' ')
    print('Working on epoch '+str(epochs_done+1)+'/'+str(N_epochs)+', '\
          +str(batch_size)+' per batch, '+str(N_iter)+' iterations')

    scores      = np.zeros((N_iter,2))

    times       = np.zeros(N_iter)
    time0       = time.time()
    for i in range(N_iter):
        mlp.fit(S_train_sc,flag_train) # Fitting to training set
        flag_inv   = mlp.predict(S_test_sc) # Performance on test set
        scores[i,0] = mlp.score(S_train_sc,flag_train)
        scores[i,1] = mlp.score(S_test_sc,flag_test)

        ETA         = (N_iter/(i+1)-1)*(time.time()-time0)
        if (i+1) % 10 == 0:
            print('# '+str(i+1)+'/'+str(N_iter)+',Mean accuracy = ' \
                  +str("%0.2f" % scores[i,1])+'\
            , ETA '+str("%0.0f"%ETA)+' s')
            print("Loss function:" ,mlp.loss_)
    epochs_done += 1
    funcs.SaveClassifier(mlp,scores,path,N_test,epochs_done)
else:
    flag_inv   = mlp.predict(S_test_sc)

#%% Printing of the network's classification metrics
target_names = [mlp.classes_[0],mlp.classes_[1]]

# Precision, Recall and F-1 score are included in the report
print(classification_report(flag_test, flag_inv, target_names=target_names))
report = np.array([classification_report(flag_test, flag_inv,\
                                         target_names=target_names)])
np.savetxt(path+"/Classification report.txt",report,fmt="%s")

# Confusion matrix
conf_mat = confusion_matrix(flag_test,flag_inv)
print("Confusion matrix:\n",conf_mat)
np.savetxt(path+"/Confusion matrix.txt",conf_mat,fmt='%.2f')

# Matthew's correlation coefficient
```

```
print("Matthews correlation coefficient:",matthews_corrcoef(flag_test,flag_inv))
```

Listing 5: Classifier training: This script makes the initialization and training of a classifier. It contains the loading of the database, the data preparation (normalization and noise addition), the training of the network and the saving of its performance metrics.

3) Neural network functions:

```
import numpy as np
import os
import pickle


#%% Method that saves the regresion network after each training epoch

def SaveNN(mlp,scores,errors,errorsabs,errorsrel,path,N_test,epochs_done):
    if not os.path.isdir(path): os.makedirs(path)
    fileObject = open(path+'NN'+str(epochs_done)+'.txt','wb')
    # Saving the network model
    pickle.dump(mlp,fileObject)
    fileObject.close()
    # Saving scores, and error metrics
    np.savetxt(path+'scores'+str(epochs_done)+'.txt',scores,delimiter='\t')
    np.savetxt(path+'errors_progress'+str(epochs_done)+'.txt',errors,delimiter='\t')
    np.savetxt(path+'errors'+str(epochs_done)+'Ntest'+str(N_test)+'.txt',errorsabs,delimiter='\t')
    np.savetxt(path+'Relative errors'+str(epochs_done)+'.txt',errorsrel,delimiter='\t')
    print(' ')
    print('Saved Network and training progress')

    return

#%% Method that saves the classification network after each training epoch

def SaveClassifier(mlp,scores,path,N_test,epochs_done):
    if not os.path.isdir(path): os.makedirs(path)
    fileObject = open(path+'NN'+str(epochs_done)+'.txt','wb')
    # Saving the network model
    pickle.dump(mlp,fileObject)
    fileObject.close()
    # Saving scores
    np.savetxt(path+'scores'+str(epochs_done)+'.txt',scores,delimiter='\t')
    print(' ')
    print('Saved Network and training progress')


    return
```

Listing 6: Neural network functions: This script contains the methods that save the neural networks (classifiers or regressors) and the metrics that indicate their progress.

# List of Tables

# List of Figures

# List of source codes

# Bibliography

# References


[1] *Chemistry libre texts*. URL: chemwiki.ucdavis.edu (visited on 09/27/2018).

[2] *IPP W 7-X*. URL: https://www.ipp.mpg.de/w7x (visited on 09/28/2018).

[3] Alexandre Bovet. *Suprathermal ion transport in turbulent magnetized plasmas*. 2015.

[4] J. van den Berg et al. "Fast analysis of collective Thomson scattering spectra on Wendelstein 7-X". In: *Review of Scientific Instruments* 89.8 (2018), p. 083507. ISSN: 0034-6748. DOI: 10.1063/1.5035416. URL: http://aip.scitation.org/doi/10.1063/1.5035416.

[5] Dmitry Moseev. *Fast Ion Dynamics in ASDEX Upgrade and TEXTOR Measured by Collective Thomson Scattering*. 2011. ISBN: 9788755039582.

[6] D. Y. Rhee et al. "Calculated collective Thomson scattered spectra from tokamak plasma with non-Maxwellian ions". In: *Review of Scientific Instruments* 61.10 (1990), pp. 3217–3219. ISSN: 00346748. DOI: 10.1063/1.1141638.

[7] J. S. Machuzak et al. "Development of high-power millimeter and submillimeter wavelength collective Thomson scattering diagnostics for energetic ion measurements in tokamaks". In: *Review of Scientific Instruments* 61.11 (1990), pp. 3544–3547. ISSN: 00346748. DOI: 10.1063/1.1141567. URL: https://doi.org/10.1063/1.1141567.

[8] Henrik Bindslev. *On the Theory of Thomson Scattering and Reflectometry in a Relativistic Magnetized Plasma On the Theory of Thomson Scattering and Reflectometry in a Relativistic Magnetized Plasma*. Vol. 663. December. 1992. ISBN: 8755018742.

[9] I. H. Hutchinson. *Principles of Plasma Diagnostics*. 2002. ISBN: 9780511613630. DOI: 10.1017/CBO9780511613630. arXiv: arXiv:1011.1669v3. URL: http://ebooks.cambridge.org/ref/id/CBO9780511613630.

[10] D. Moseev et al. "Comparison of measured and simulated fast ion velocity distributions in the TEXTOR tokamak". In: *Plasma Physics and Controlled Fusion* 53.10 (2011). ISSN: 13616587. DOI: 10.1088/0741-3335/53/10/105004.

[11] E E Salpeter. "Electron Density Fluctuations in a Plasma". In: *Phys. Rev.* 120.5 (Dec. 1960), pp. 1528–1535. DOI: 10.1103/PhysRev.120.1528. URL: https://link.aps.org/doi/10.1103/PhysRev.120.1528.

[12] T. P. Hughest and S. R.P. Smith. "Effects of plasma dielectric properties on Thomson scattering of millimetre waves in tokamak plasmas". In: *Journal of Plasma Physics* 42.2 (1989), pp. 215–240. ISSN: 14697807. DOI: 10.1017/S0022377800014318.

[13] M. Salewski et al. "Comparison of fast ion collective Thomson scattering measurements at ASDEX Upgrade with numerical simulations". In: *Nuclear Fusion* 50.3 (2010). ISSN: 17414326. DOI: 10.1088/0029-5515/50/3/035012.

[14] M. Nishiura et al. "Spectrum response and analysis of 77 GHz band collective Thomson scattering diagnostic for bulk and fast ions in LHD plasmas". In: *Nuclear Fusion* 54.2 (2014). ISSN: 00295515. DOI: 10.1088/0029-5515/54/2/023006.

[15] J Rasmussen et al. "Consistency between real and synthetic fast-ion measurements at ASDEX Upgrade". In: *Plasma Physics and Controlled Fusion* 57.7 (2015), p. 075014. URL: http://stacks.iop.org/0741-3335/57/i=7/a=075014.

[16] C M Bishop. "Neural networks for pattern recognition". In: *Journal of the American Statistical Association* 92 (1995), p. 482. ISSN: 01621459. DOI: 10.2307/2965437. arXiv: 0-387-31073-8.

[17] David Kriesel. "A Brief Introduction to Neural Networks". In: *Retrieved August* (2005), p. 244. ISSN: 1432-0711. DOI: 10.1016/0893-6080(94)90051-5. arXiv: arXiv:1011.1669v3. URL: http://www.dkriesel.com/en/science/neural%7B%5C_%7Dnetworks.

[18] I Abramovic et al. "Collective Thomson scattering data analysis for Wendelstein 7-X". In: *Journal of Instrumentation* 12.08 (2017), p. C08015.

[19] T. P. Hughes and S. R P Smith. "Calculations of Thomson scattering functions for alpha particle diagnostics in JET plasmas". In: *Nuclear Fusion* 28.8 (1988), pp. 1451–1457. ISSN: 17414326. DOI: 10.1088/0029-5515/28/8/012.

[20] L. Vahala, G. Vahala, and D. J. Sigmar. "Effects of alpha particles on the scattering function in CO 2 laser scattering". In: *Nuclear Fusion* 26.1 (1986), pp. 51–60. ISSN: 0029-5515. DOI: 10.1088/0029-5515/26/1/005. URL: http://stacks.iop.org/0029-5515/26/i=1/a=005?key=crossref.ec1c718c7317b69956e2154e57f06b08.

[21] M. Salewski et al. "Tomography of fast-ion velocity-space distributions from synthetic CTS and FIDA measurements". In: *Nuclear Fusion* 52.10 (2012), p. 103008. URL: http://stacks.iop.org/0029-5515/52/i=10/a=103008.

[22] Bing Wang et al. "Artificial neural networks for the prediction of peptide drift time in ion mobility mass spectrometry". In: *BMC Bioinformatics* 11.1 (Apr. 2010), p. 182. ISSN: 1471-2105. DOI: 10.1186/1471-2105-11-182. URL: https://doi.org/10.1186/1471-2105-11-182.

[23] Frank Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6 (1958), pp. 386–408.

[24] F Pedregosa et al. "Scikit-learn: Machine Learning in {P}ython". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[25] Jason Brownlee. *Brownlee*. URL: https://machinelearningmastery.com.

[26] Wikipedia. *Conf_Matrix_wiki*. URL: https://en.wikipedia.org/wiki/Confusion%7B%5C_%7Dmatrix.

[27] Davide Chicco. "Ten quick tips for machine learning in computational biology". In: *BioData Mining* 10.1 (Dec. 2017), p. 35. ISSN: 1756-0381. DOI: 10.1186/s13040-017-0155-3. URL: https://doi.org/10.1186/s13040-017-0155-3.

[28] Wikipedia. *Recall_Wiki*. URL: https://en.wikipedia.org/wiki/Precision%7B%5C_%7Dand%7B%5C_%7Drecall.

[29] Walber. *Walber_Recall*. URL: https://commons.wikimedia.org/w/index.php?curid=36926283.

[30] Wikipedia. *MC_Wiki*. URL: https://en.wikipedia.org/wiki/Matthews%7B%5C_%7Dcorrelation%7B%5C_%7Dcoefficient.

[31] John D Gaffey. "Energetic ion distribution resulting from neutral beam injection in tokamaks". In: *Journal of Plasma Physics* 16.2 (1976), pp. 149–169. DOI: 10.1017/S0022377800020134.

[32] PlasmaPy Community et al. *PlasmaPy: an open source community-developed Python package for plasma physics*. This work was partially supported by the U.S. Department of Energy. Apr. 2018. DOI: 10.5281/zenodo.1238132. URL: https://doi.org/10.5281/zenodo.1238132.

[33] Heaton Jeff. *Heaton*. URL: https://www.heatonresearch.com.

[34] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[35] M Salewski et al. "Comparison of collective Thomson scattering signals due to fast ions in ITER scenarios with fusion and auxiliary heating". In: *Plasma Physics and Controlled Fusion* 51.3 (2009), p. 035006. URL: http://stacks.iop.org/0741-3335/51/i=3/a=035006.

[36] S. K. Nielsen et al. "Recent development of collective Thomson scattering for magnetically confined fusion plasmas". In: *Physica Scripta* 92.2 (2017). ISSN: 14024896. DOI: 10.1088/1402-4896/92/2/024001.

[37] R.E. Aamodt and D.A. Russell. "Alpha particle detection by electromagnetic scattering off of plasma fluctuations". In: *Nuclear Fusion* 32.5 (1992), p. 745. URL: http://stacks.iop.org/0029-5515/32/i=5/a=I03.

[38] Angel Fernando Kuri-Morales. "The Best Neural Network Architecture". In: *Nature-Inspired Computation and Machine Learning*. Ed. by Alexander Gelbukh, Félix Castro Espinoza, and Sofía N. Galicia-Haro. Cham: Springer International Publishing, 2014, pp. 72–84. ISBN: 978-3-319-13650-9.

[39] Philippe Thomas and Marie-Christine Suhner. "A New Multilayer Perceptron Pruning Algorithm for Classification and Regression Applications". In: *Neural Processing Letters* 42.2 (Oct. 2015), pp. 437–458. ISSN: 1573-773X. DOI: 10.1007/s11063-014-9366-5. URL: https://doi.org/10.1007/s11063-014-9366-5.

[40] M Stejner et al. "Measurements of plasma composition in the TEXTOR tokamak by collective Thomson scattering". In: *Plasma Physics and Controlled Fusion* 54 (Dec. 2011), p. 015008. DOI: 10.1088/0741-3335/54/1/015008.

[41] Poole David L. and Mackworth Alan K. *Artificial Intelligence: Foundations of Computational Agents*. New York, NY, USA: Cambridge University Press, 2010. ISBN: 9780521519007.

[42] Fei Tong and Xila Liu. "Samples selection for artificial neural network training in preliminary structural design". In: *Tsinghua Science and Technology* 10.2 (2005), pp. 233–239. ISSN: 10070214. DOI: 10.1016/S1007-0214(05)70060-2.

[43] D S Broomnhead Lowe and D. "RADIAL BASIS FUNCTIONS, MULTI-VARIABLE FUNCTIONAL INTERPOLATION AND ADAPTIVE NETWORKS". In: *Networks* 4148 ().

[44] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. arXiv: arXiv:1411.3159v1. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.79.8.2554.

[45] Geoffrey E. Hinton and T. J. Sejnowski. *Learning_and_relearning_in_Boltzmann_machines.pdf*.