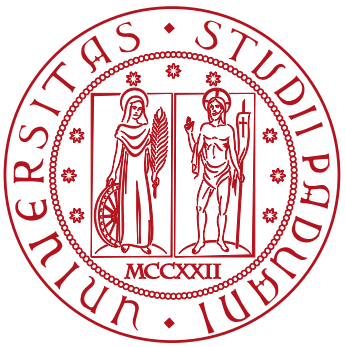



# UNIVERSITÀ DEGLI STUDI DI PADOVA

**Dipartimento di Fisica e Astronomia "Galileo Galilei"**

**Master Degree in Physics of Data**


**Final Dissertation**

## Machine learning approaches for plasma state monitoring in Tokamaks


**Thesis supervisor**

**Dr. Alessandro Pau**

**Thesis co-supervisors**

**Prof. Gianluigi Serianni**

**Dr. Olivier Sauter**

**Candidate**

**Cristina Venturini**


**Academic Year 2021/2022**

# Abstract

Data science methods from the fields of machine learning and artificial intelligence (ML/AI) offer several opportunities for enabling and accelerating progress toward the realization of fusion energy.
The massive amount of data available from current operating tokamaks, together with the exponential growth of computing power and cloud computing technologies, have enabled new scenarios in the framework of advanced data analysis and processing. In particular, machine learning methods are used to maximize the information extracted from measurements, systematically fuse multiple data sources and infer quantities that are not directly measured or cannot be easily computed in real-time during the experiments.
The work proposed in this thesis focuses on the reconstruction of the electron temperature 1D plasma profile from the TCV tokamak through a neural network model. An autoencoder structure with recurrent layers is employed, which succeeds in solving the reconstruction task on a difficult fusion dataset.

EPFL
SWISS
PLASMA
CENTER

# Contents

# Introduction

The world lacks safe, low-carbon, large-scale and cheap energy alternatives to fossil fuels. The need for reducing emissions has never been so strong as in recent years: with the issue of global warming taking center stage in the social and political discussion, both governments and citizens are increasingly demanding that sustainable alternatives, in all fields, be employed.
In Fig. 1 it can clearly be seen that the main responsible for greenhouse gas emissions in the world is linked to energy production, which output $\approx 15$ billion tonnes of carbon dioxide-equivalents in 2018; the second largest producer is transport, with half the emissions. This clearly shows that the search for a clean energy production method is of the utmost importance and urgency.

At current, we face the so called energy problems: the lack of access to

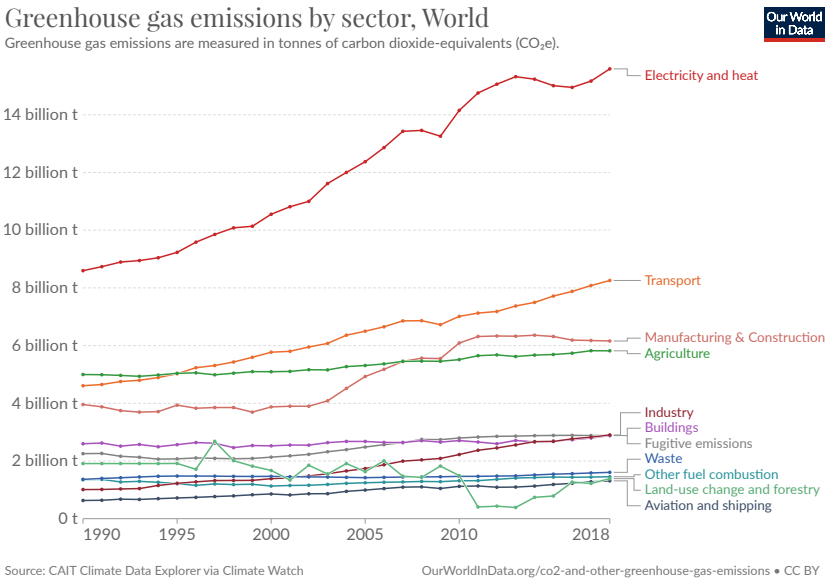


Figure 1

energy for countries that have low carbon emissions and too high greenhouse

gas emissions from those that do have access to energy. Many are the efforts and attempts towards finding a solution, which may very well be fundamental for the future of humanity.
Low-carbon energy production alternatives are already employed, but they are simply not as efficient as fossil fuels and gas and are therefore (for now) not sufficient by themselves.
In the World Energy Outlook 2021 [1] by the International Energy Agency it is stated that to reach the Net Zero Emissions (NZE) scenario by 2050, the share of energy produced by nuclear power needs to increase from less than 5% in 2020 to almost 15% in 2050. This trend is only expected to grow, and given the ongoing public debate on the safety of nuclear fission, the success of nuclear fusion could be a relevant factor.
Moreover, if one does not consider the NZE scenario, which is the best outcome humanity can hope for, we are looking at a likely future environment where the global electricity demand has greatly increased by 2050. In fact, given the currently implemented policies, it will rise from 589.1 EJ in 2020 to 743.9 EJ in 2050). It is once again apparent that the need for large scale, sustainable, safe and low-carbon electricity generation is extremely relevant.
Currently, the state of research concerning the manufacturing of a fusion reactor is still in an experimental phase. At present, in the south of France, 35 nations around the world are collaborating to build the biggest fusion experiment in the world: ITER. The aim is for it to be the next big leap towards the realization of a commercial reactor: it is set to prove the scientific and technological feasibility of fusion.
Extensive research has been carried out in the past years to prepare for ITER: model based simulations, study of materials and radiations, development of new diagnostics and many more. The experimental conditions are extremely demanding and the slow pace of the breakthroughs in this area of research is directly linked to the complexity of the problem, both from a physical and technological point of view.
Applications able to catalyze this process are a great asset: recently, data science methods from the field of artificial intelligence (AI), in particular machine learning (ML), have started to offer several opportunities for enabling and accelerating progress in this field. Given the massive amount of data produced by current operating machines and the exponential growth of computing power and cloud computing technologies, the employment of these new methods comes as a natural fit.
Machine learning methods can be used to maximize the information extracted from measurements, systematically fuse multiple data sources and infer quantities that are not directly measured or cannot be easily computed in real-time during the experiments.
In particular, in this work a neural network model is presented which, by fusing different data sources and exploiting the knowledge of the plasma confinement state, reconstructs a key 1D kinetic plasma profile: the elec-

tron temperature $T_e$.
Being able to monitor these quantity is of paramount importance for machine performance: it would allow to improve continuous control and plasma state monitoring, as well as the estimation of the proximity to disruptive boundaries (regions of the operating space in which stable operation of the machine is not possible).

# Chapter 1

# Physics Background

## 1.1 Nuclear Fusion

Nuclear fusion was first suggested in 1920, when Arthur Eddington proposed hydrogen-helium fusion as the primary source of stellar energy. In the following 20 years, the theoretical basis were studied and at the beginning of the 1940s the first experiments to reproduce fusion on earth were being carried out.
Research towards developing controlled fusion inside reactors is ongoing and no ready-to-use technology is available as of now.
Physically, the process of nuclear fusion consists in two light nuclei that fuse together, forming a heavier nucleus and releasing energy in the form of kinetic energy of the nucleus itself and of other particles (*e.g.* neutrons, photons) produced in the reaction.
There are many possible nuclear fusion reactions, but the main ones in term of favourable energy output are:

$$^{2}\mathrm{D} + {}^{3}\mathrm{T} \longrightarrow {}^{4}\mathrm{He} + {}^{1}\mathrm{n} + 17.6\ \mathrm{MeV}$$

$$^{2}\mathrm{D} + {}^{2}\mathrm{D} \longrightarrow {}^{3}\mathrm{He} + {}^{1}\mathrm{n} + 3.27\ \mathrm{MeV}$$

$$^{2}\mathrm{D} + {}^{2}\mathrm{D} \longrightarrow {}^{3}\mathrm{T} + {}^{1}\mathrm{H} + 4.03\ \mathrm{MeV}$$

$$^{2}\mathrm{D} + {}^{3}\mathrm{He} \longrightarrow {}^{4}\mathrm{He} + {}^{1}\mathrm{H} + 18.3\ \mathrm{MeV}$$

For two nuclei to fuse, a substantial energy barrier, the Coulomb barrier, needs to be overcome. This is the barrier put in place by the electrostatic repulsive force between the positively charged protons of the nuclei. Only when the nuclei are close enough for long enough the nuclear force can win over the electrostatic force and allow fusion to happen.
Given the relative velocity $v$ of the two reactant nuclei, the reaction cross section $\sigma$ gives a measure of the probability of a fusion reaction. If in general

one has a high number of reactant nuclei, it will be necessary to work with velocity distributions, typically Maxwell-Boltzmann. By calling $\langle\sigma\rangle$ and $\langle v\rangle$ the average of $\sigma$ and $v$ with respect to their distributions, the reaction rate can be then defined as:

$$f = n_1 n_2 \langle\sigma v\rangle$$

where $\langle\sigma v\rangle$ is the reactivity. In Fig. 1.1 is reported the reactivity as a func-

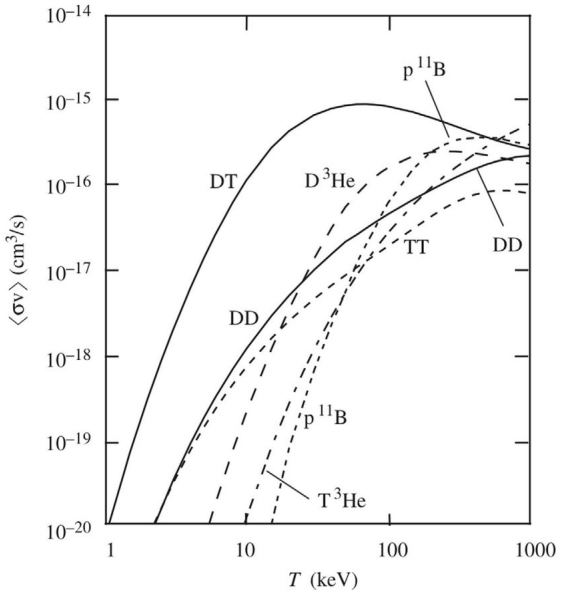


Figure 1.1: Reactivity $\langle\sigma v\rangle$ as a function of the temperature $T$ for various fusion reactions. Figure taken from [2]

tion of temperature for three commonly considered fusion reactions. The nuclei involved are isotopes of hydrogen and are chosen for their lightness (they only contain one proton), which results in a lower Coulomb barrier. It can be seen that $\langle\sigma v\rangle$ grows from practically zero at room temperature to relevant values at temperatures of 10-100keV, at which the reactants exist in a plasma state. This is the reason for which current experiments operate at high temperatures with reactants in the plasma state.
Moreover, Fig. 1.1 also tells us why deuterium tritium (DT) reaction is the preferred one for ongoing fusion experiments: it peaks in reactivity at a lower temperature and higher value.

## 1.2 Tokamak

The research towards building a fusion reactor has been ongoing since the 1940s, without successful attempts at producing more electrical energy than

the one needed to power the machine.
More than one paradigm has been tested in the experimental campaigns towards fusion power, the main two being magnetic confinement and inertial confinement.
A useful quantity to consider to understand these two different mechanisms is the triple product $nT\tau_E$, where $n$ is the plasma density (assuming that the densities of tritium and deuterium are equal), $T$ is the plasma temperature and $\tau_E$ is the confinement time. The bigger the triple product, the better the performance of the fusion machine [3].
In magnetic confinement, the goal is to confine a hot plasma for a long time, so $\tau_E$ needs to be maximized. In inertial confinement on the other hand, the density $n$ is maximized, while the confinement time $\tau_E$ is really short.
Magnetic confinement can be realized with different machine configurations: tokamak, stellarators, compact toroids, reversed field pinch (RFP). Currently, the most promising one is the tokamak, a fusion reactor of toroidal shape in which a hot plasma at low pressure is confined with multiple sets of magnetic coils placed outside the low pressure chamber.
The biggest fusion experiment in the world, ITER, involves a tokamak machine. It is set to achieve for the first time a positive energy balance, producing more energy than the one needed to power it. The next projected step is DEMO, a reactor prototype which will have to demonstrate the economic viability of a fusion reactor.
Given the relevance of the Tokamak in the current experimental situation, in this chapter the basic functioning concepts of this machine will be outlined. First, a brief discussion of the power balance of the machine will be presented (Section 1.2.1), then an explanation of how magnetic confinement works (Section 1.2.2) and the theory of toroidal equilibrium will be outlined(Section 1.2.3). Subsequently, plasma heating methods (Section 1.2.4) and plasma instabilities (Section 1.2.5) will be presented, finally followed by a short introduction of the transport problem in tokamaks (Section 1.2.6).

### 1.2.1 Power balance

In general, a machine that aims to achieve nuclear fusion needs to generate enough energy to overcome its energy losses. This concept can be formalized by giving an examination of the energy balance for a fusion power plant using a hot plasma and magnetic confinement. The quantities to be considered are as follows:

- External power; the energy needed to drive the device, namely the power injected into the plasma to heat it, $P_H$.

- Fusion power; the energy generated by the fusion reactions. The power

per unit volume can be written as

$$p_f = \frac{1}{4} n^2 \langle \sigma v \rangle E_f \tag{1.1}$$

where $E_f$ is the energy produced by a single reaction (17.6 $MeV$ for D-T), $n = n_T + n_T$ and $n_D = n_T = n/2$, with $n_D$ and $n_T$ the densities of deuterium and tritium respectively.
If one considers D-T fusion reactions the fraction of $E_f$ carried by the neutrons (4/5) exits the machine, so it is not involved in further heating of the plasma. The remaining $E_f$ fraction (1/5) is associated to the $\alpha$ particles which, being charged, remain trapped inside the plasma and release their energy through collisions:

$$p_\alpha = \frac{1}{4} n^2 \langle \sigma v \rangle E_\alpha \tag{1.2}$$

$$E_\alpha = \frac{E_f}{5} \tag{1.3}$$

- Radiation loss; the energy lost in the form of light leaving the plasma. This can happen due to multiple mechanisms, mainly bremsstrahlung, line radiation and cyclotron radiation. In first approximation, line radiation can be ignored (negligible if the impurity level is reduced to very low values) and cyclotron radiation (the plasma is opaque at those frequencies, so most of the emitted radiation is re-absorbed). Bremsstrahlung, on the other hand, is intrinsic and unavoidable, since it originates from the acceleration and deceleration of charged particles during Coulomb interactions. For bremsstrahlung, and thus, in first approximation, for radiation losses, the power lost per unit volume is:

$$p_R = p_b = \alpha_b n^2 T^{1/2} \ \mathrm{Wm}^{-3} \tag{1.4}$$

where $\alpha_b = 5.35 \times 10^{-37} \ \mathrm{Wm}^3\mathrm{keV}^{-1/2}$

- Transport losses; the energy lost in the form of particles leaving the plasma, both as convection and conduction. This quantity is difficult to estimate, since the problem of transport in fusion plasmas is still unsolved. Experimentally, it is quantified through the energy confinement time, $\tau_E$, which is defined through:

$$P_L = \frac{W}{\tau_E} \tag{1.5}$$

where $W = wV$ is the total plasma thermal energy and $P_L = p_L V$ are the total power losses, with $V$ the plasma volume. The interpretation of $\tau_E$ is that if, ideally, one could switch off all heating sources and radiation losses, the stored energy in the plasma would decay exponentially with a characteristic time of $\tau_E$.

Furthermore, in stationary conditions (absent/negligible fusion reactions) $P_H = P_L + P_R$. Thus, since experimentally the value of $P_H$ is known, one can write:

$$\tau_E = \frac{W}{P_H - P_R} \tag{1.6}$$

First, it is to be observed that, even considering the ideal case of absence of transport losses, in order to obtain a positive energy balance, the power lost by bremsstrahlung would still put a lower limit on the plasma temperature. This value corresponds to 4.4 keV for D-T and 30 keV for D-D.
On the other hand, if considering transport losses, we can define the ignition condition: $\alpha$ particle heating balances energy losses and the plasma is self-sustained, meaning $P_H = 0$ or

$$p_\alpha = p_L + p_b \tag{1.7}$$

By inserting Eq. (1.2), Eq. (1.4), Eq. (1.5), Eq. (1.6) in Eq. (1.7) a condition on the product of density and energy confinement time, as a function of temperature, is obtained:

$$n\tau_E = \frac{12T}{\langle \sigma v \rangle E_\alpha - 4\alpha_b T^{1/2}} \tag{1.8}$$

If one considers the plot of the $n\tau_E$ curve for D-T it can be seen that it displays a minimum at $T = 25 - 30$ keV. Even so, a fusion reactor will likely be operated at a lower temperature, around $10 - 20$ keV: this is due to the higher power density associated to these temperatures.
In the $10 - 20$ keV range, by using an approximation for the reactivity (which works within a 10% discrepancy), neglecting radiation losses (which are relatively low at this temperature for D-T) and considering parabolic radial profiles for density and temperature, a new condition is obtained:

$$nT\tau_E = 5 \times 10^{21}\ \mathrm{m^{-3}keVs} \tag{1.9}$$

which is based on the product of temperature, density and confinement time. The quantity in Eq. (1.9) is referred to as triple product.
For current fusion research, the ignition condition is no longer a desirable achievement. This regime might seem advantageous, because one wouldn't need to spend power running the external heating system, if not for a transient period at the beginning. Nonetheless a tool for active control of the plasma temperature would be lacking.
A more relevant parameter is the $Q$ parameter, an energy gain factor which represents the ability of the plasma to be a net energy producer. It is the ratio between the power produced through fusion and the power required to

heat the plasma:

$$Q = \frac{\frac{1}{4}n^2\langle\sigma v\rangle E_f V}{P_H} \tag{1.10}$$

Ignition corresponds to $Q \to \infty$ ($P_H = 0$). The condition $Q = 1$ is referred to as break-even: fusion produces the same amount of energy used to heat the plasma. For $Q = 5$ the external heating is equal to the $\alpha$ particles heating (Eq. (1.3)), so only half of the power losses need to be compensated with external heating. The target value for ITER is set at $Q = 10$, while for a profitable reactor $Q = 50$ is the minimum acceptable value.

### 1.2.2 Magnetic confinement

Thermonuclear fusion plasmas cannot directly come into contact with the walls of the tokamak: due to their high temperature they would erode the materials and as a consequence lower their own temperature.
In order to avoid this issue, one needs to confine the plasma inside the chamber so that it won't touch the walls. Such condition can be achieved through the use of magnetic confinement, which means employing magnetic fields to confine the charged particles in the plasma in a specific region of space.
Indeed, if one considers a charged particle in a magnetic field, it will follow a spiral orbit with radius, called Larmor radius, equal to:

$$\rho_L = \frac{mv_\perp}{qB} \tag{1.11}$$

where $v_\perp$ is the velocity component perpendicular to the magnetic field and $q$ is the particle charge.
This property leads to the confinement of the particle in the two directions perpendicular to the magnetic field, if the Larmor radius is significantly smaller than the size of the tokamak chamber. Furthermore, recalling that the velocity is mainly dictated by the thermal velocity, which goes as $(T/m)^{1/2}$ it follows that as the temperature of the plasma increases a bigger magnetic field will be required.
As for what concerns the magnetic field, a uniform magnetic field is insufficient for good confinement, since particles still have one direction along which to escape the chamber. By making the field non uniform one can realize the magnetic mirror effect, which consists in making the field more intense at the ends of the plasma column so that particles mostly bounce back and remain in the volume. Nonetheless, this configuration suffers from significant losses, driven by Coulomb collisions and instabilities; the development of the latter can be linked to the velocity distribution, which can become strongly different from a Maxwellian.
To avoid longitudinal losses, the most straightforward way is to close the

magnetic field lines on themselves, obtaining a toroidal configuration.
In practice, this corresponds to placing the coils as to form a toroidal solenoid. However, even this configuration presents modest confinement properties. In fact, the field generated by a toroidal solenoid is not uniform, but decays from the torus major axis as $1/R$. This is at the origin of a drift motion which causes the center of the circular orbits to translate in the direction parallel to the magnetic field, as well as in the perpendicular one, with a drift velocity given by:

$$\mathbf{v_d} = \frac{mv_\perp^2}{2qB}\frac{\mathbf{B}\times\nabla B}{B^2} + \frac{mv_\parallel^2}{qB}\frac{\mathbf{R_C}\times\mathbf{B}}{R_c^2 B} \tag{1.12}$$

where the first term is known as gradient drift and is linked to the non-uniformity of the magnetic field $\mathbf{B}$, while the second term is called curvature drift and is related to the radius of curvature of the magnetic field lines $\mathbf{R_C}$.
It can be noted that these two contributions are dependent on the charge of the particles involved, so that the motion associated to these effects will be in opposite direction for ions and electrons.
This motion leads to charge separation and the formation of an electric field, to which is associated a drift velocity:

$$\mathbf{v_d} = \frac{\mathbf{E}\times\mathbf{B}}{B^2} \tag{1.13}$$

which is derived considering a charged particle in stationary electric and magnetic fields. Note that the direction of motion is independent on the charge of the particle.
This last contribution to the drift is responsible for the loss of confinement. To mitigate this loss, one can superimpose to the toroidal component of the magnetic field, generated by the coils, a poloidal one generated by a current flowing in the plasma itself. By superimposing these two components one obtains field lines that run around the torus helicoidally: the field has a *rotational tranform*.
In this configuration, the motion of the particle orbit centers will be such that when they are located in the torus upper part the drift will move them away from the upper edge, and when they are in the lower part they will be moved towards the upper part. Overall, along a full poloidal rotation the effects of the drift motion cancel out.
In a Tokamak, this field lines configuration is achieved by multiple set of coils.
The toroidal component is generated by coils placed as to approximate a toroidal solenoid: the approximation lies in the number of coils that are employed, which varies based on cost, the necessity to fit diagnostics and heating systems between the coils and the need to reduce the non uniformities due to the distance between coils.

The poloidal field component is generated by a toroidal current (plasma current) flowing in the plasma, which in turn is induced by a system of coils in which a time-dependent current is flowing. This current causes a variation of the magnetic flux and by Faraday's law this induces a toroidal electric field, that finally generates the plasma current. This second set of coils is set at the centre of the tokamak and acts as the primary winding of a transformer, with the plasma acting as the secondary.

There is a third and last set of coils which is responsible for the position and

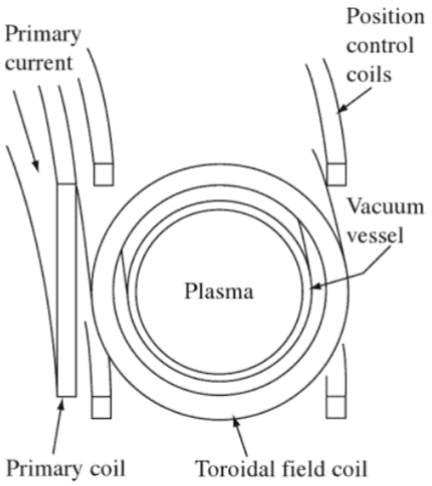


Figure 1.2: Scheme of the three sets of coils in a tokamak machine. Figure taken from [3]

shape of the plasma column; in these, current flows parallel to the plasma. A scheme of the three sets of coils can be seen in Fig. 1.2.

### 1.2.3 Toroidal equilibrium

Considering the case of a fully ionized plasma, which coincides to the ones produced in Tokamaks, the equilibrium condition can be expressed by adopting the magnetohydrodynamic (MHD) point of view. In this framework the plasma is seen as a conducting fluid. In particular, the equation of motion can be written in the same way as for a neutral fluid, with an additional term related to the Lorentz force:

$$mn\left(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\,\mathbf{v}\right) = \mathbf{j}\times\mathbf{B} - \nabla p + mn\nu\nabla^2\mathbf{v} \tag{1.14}$$

where $m$ is the ion mass, $n$ is the plasma density, $\mathbf{v}$ its velocity and $p$ its pressure; $\mathbf{j}$ and $\mathbf{B}$ are current density and magnetic field, $\nu$ is the viscosity. The pressure is defined as the product of density and temperature.

If, in first approximation, plasma motion is neglected($\mathbf{v} = 0$) a force equilibrium relation between the pressure gradient and the Lorentz force is obtained:

$$\mathbf{j} \times \mathbf{B} = \nabla p \tag{1.15}$$

Starting from this equation, making the further assumption of working with an axysymmetric toroidal plasma then the so called Grad-Shafranov equation can be derived [3].
The Grad-Shafranov equation gives as solution a function that describes the magnetic surfaces of such configuration. Indeed, in an axysimmetric toroidal equilibria the magnetic field lines lie on nested magnetic surfaces, wrapped around a circular line called magnetic axis.
The solution is expressed in cylindrical coordinates $(R, Z, \phi)$ as can be seen

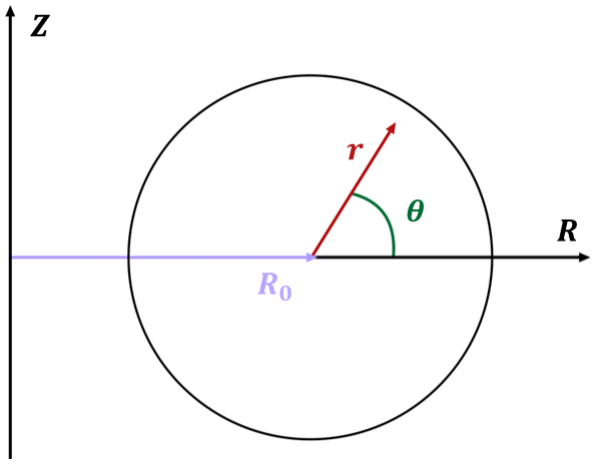


Figure 1.3: Scheme of the cilindrical and polar coordinates

in Fig. 1.3. The direction described by the unit vector $\mathbf{e}_\phi$ is the toroidal direction while the $RZ$ plane in Fig. 1.3 is called poloidal plane.
Symmetry with respect to $\phi$ is assumed, which is approximately true for tokamaks [3].
By writing the magnetic field as the vectorial sum of toroidal and poloidal component one has:

$$\mathbf{B} = B_\phi \mathbf{e}_\phi + \mathbf{B}_p \tag{1.16}$$

A function $\psi(R, Z)$ can be thus defined such that:

$$B_R = -\frac{1}{R}\frac{\partial \psi}{\partial Z} \tag{1.17}$$

$$B_Z = \frac{1}{R}\frac{\partial \psi}{\partial R} \tag{1.18}$$

It can be easily verified that these expressions satisfy the solenoidality condition which in this case can be written as [3]:

$$\frac{1}{R}\frac{\partial}{\partial R}\left(RB_R\right) + \frac{\partial B_Z}{\partial Z} = 0 \tag{1.19}$$

or in more compact form:

$$\mathbf{B}_p = \frac{1}{R}\left(\nabla\psi \times \mathbf{e}_\phi\right) \tag{1.20}$$

Moreover, the function $\psi$ satisfies $\mathbf{B}\cdot\nabla\psi = 0$ and thus it can be implied that $\psi$ is constant along each magnetic field line. This furthermore infers that $\psi$ is constant over each magnetic surface: once $\psi\left(R,Z\right)$ is described the magnetic surfaces directly follows as the loci of the points on which $\psi$ is constant.
The Grad-Shafranov equation lends the form of $\psi\left(R,Z\right)$, which is called poloidal flux function, as solution.
Similarly to what has been written for the magnetic field, one can work with the current density, defining:

$$j_R = -\frac{1}{R}\frac{\partial f}{\partial Z} \tag{1.21}$$

$$j_Z = \frac{1}{R}\frac{\partial f}{\partial R} \tag{1.22}$$

and by comparing these relations with Ampere's law $\nabla\times\mathbf{B} = \mu_0\mathbf{j}$ one realizes:

$$f = \frac{RB_\phi}{\mu_0} \tag{1.23}$$

With a reasoning similar to that used for $\psi$ it can be concluded that $\mathbf{j}\cdot\nabla f = 0$, so $f$ is constant along the current density lines.
From Eq. (1.15) it can be understood that $\mathbf{j}$ and $\mathbf{B}$ are orthogonal to the pressure gradient. This leads to the fact that pressure is constant along the magnetic field lines and therefore surfaces, also that the pressure is constant along the current density lines. Thus, current density lines lie on the magnetic surfaces. Finally, since $f$ is constant on current density lines it will also be constant on magnetic surfaces. From this, it can be concluded that it is possible to express $f$ and $p$ as functions of $\psi$ only:

$$f = f\left(\psi\right) \tag{1.24}$$

$$p = p\left(\psi\right) \tag{1.25}$$

$f\left(\psi\right)$ and $p\left(\psi\right)$ are the input to the Grad-Shafranov equation, which can be finally written as [3]:

$$R\frac{\partial}{\partial R}\left(\frac{1}{R}\frac{\partial\psi}{\partial R}\right) + \frac{\partial^2\psi}{\partial Z^2} = -\mu_0 R^2 p'\left(\psi\right) - \mu_0^2 f\left(\psi\right) f'\left(\psi\right) \tag{1.26}$$

or in more compact form, by defining the operator $\Delta^*$:

$$\Delta^* = R\frac{\partial}{\partial R}\left(\frac{1}{R}\frac{\partial}{\partial R}\right) + \frac{\partial^2}{\partial Z^2} \tag{1.27}$$

$$\Delta^*\psi = -\mu_0 R^2 p' - \mu_0^2 f f' \tag{1.28}$$

Boundary conditions hold great importance in the resolution of the equation and are in practice determined by the currents flowing in the coils for plasma control (position and shape).

### 1.2.4 Plasma heating

To produce a sufficient number of fusion reactions in the volume, plasma temperature needs to reach a high enough value, as explained in Section 1.1. Thus arises the need for a system to heat the plasma and balance the inevitable losses due to imperfect confinement.
It is found [3] that as temperature rises, energy losses in the plasma grow consequently. If one remains at low enough temperatures, such that fusion reactions can be neglected, losses will be balanced by the ohmic heating alone. On the other hand, when fusion reactions become relevant, which is desirable inside a tokamak, additional heating will be needed.
The plasma current has the byproduct of heating the plasma by Joule effect. Nonetheless, this heating alone is not enough to bring the plasma to a high enough temperature. For a fully ionized plasma, its resistivity can be expressed [3] through the Spitzer-Härm relation:

$$\eta\,(\Omega\mathrm{m}) = 1.65 \times 10^{-9}\frac{\ln\Lambda}{T_e\,(\mathrm{keV})^{3/2}} \tag{1.29}$$

from which can be deduced the dependence of resistivity on temperature, which goes as $T_e^{-3/2}$. As the plasma gets hotter, it becomes less resistive, already signaling the difficulty of reaching thermonuclear conditions by ohmic heating alone. Indeed, to reach high enough temperature it would be needed to sustain intense magnetic fields, of the order of 13 T. [4]
It is thus clear that there is a need for additional heating systems, which in general fall into two categories:

- Neutral beam heating, the injection of energetic beams of neutral atoms;
- RF heating, injection of electromagnetic waves at radiofrequency or in the microwave range

As for neutral beam heating, it consists in injecting neutral particles (charged ones would be deflected by the magnetic field) in the plasma volume, which are then ionized by collisions with the plasma particles. As a consequence,

they are confined by the magnetic field and gradually release their energy to the plasma by means of thermalization. The position where most of the thermalization happens can be controlled by tuning/tailoring the beam energy: preferably one would want the bulk of the energy to be deposited in the core.
The beam particles and plasma particles can interact through different processes, namely charge exchange, ion-driven ionization and electron-driven ionization. Different processes dominate at different energies of the beam.
It can be concluded, after carrying out some calculations on the cross sections, that required energy of the beam for a tokamak with a minor radius of a few metres is of the order of the MeV. From the technological point of view, this represents quite a challenge.
As for the direction of injection, it is preferred to be as tangential as pos-

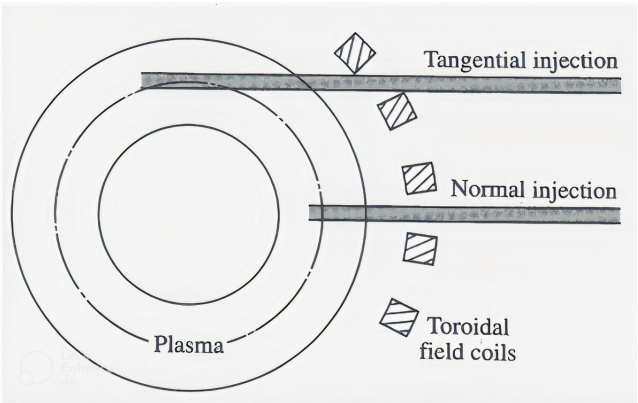


Figure 1.4: Scheme of NBI injection system. Figure taken from [3]

sible, as can be seen from Fig. 1.4. A normal injection could result in losses due to the ripple of the toroidal field or in damage resulting from depositing the beam power over the internal wall after incomplete absorption inside the plasma. Nonetheless, actually realizing tangential injection depends on the actual space availability amongst the coils; often positioning of the NBI is a compromise between actual space and desire for tangential injection.
For what concerns RF heating, the functioning is based on injecting electromagnetic waves in the plasma in an effective way. The effectiveness lies in the ability for the radiation to propagate, without being reflected, up to a plasma region where it is absorbed, transferring its energy to the plasma particles.
There are three main schemes for radiation injection:

- ion cyclotron resonance heating (ICRH), operating at frequencies from 30 to 120 MHz, depending on the magnetic field. The idea is to inject waves with a frequency close to the ion cyclotron frequency. Given that the magnetic field decays as $1/R$, resonance will take place in a

specific region where the condition $\omega = n\omega_{ci}$, with $n$ integer and $\omega_{ci}$ the ion cyclotron frequency, is satisfied [3]. By controlling the frequency, one can control the power deposition location.

- lower hybrid resonance heating (LH), it operates from 1 to 8 GHz, frequencies high enough that waveguides can be used for power transmission. The frequency is such that $\omega_{ci} << \omega << \omega_{ce}$ and the associated wave is called lower hybrid wave. This scheme results in electron heating, though the main interest lies in its ability to drive current in the plasma.
- electron cyclotron resonance heating (ECRH), uses waves with frequencies between 100 and 200 GHz, that is in the microwave domain. The resonance in this case is with the electron gyrotron frequency. This setup is also used to drive the current in the plasma.

### 1.2.5 Instabilities

One of the mechanisms impacting the most the energy confinement, together with transport losses and the other phenomena described in the previous sections, is the appearance of plasma instabilities, typically of Magneto Hydro-Dynamic (MHD) nature.
An interesting quantity linked to plasma instabilities is the $q$ value, which is defined as:

$$q = \frac{1}{2\pi} \oint \mathrm{d}\phi \tag{1.30}$$

where $\mathrm{d}\phi$ is the infinitesimal variation of the toroidal angle. A $q$ value of 1 corresponds to a field line that comes back to its starting poloidal position after making one toroidal turn. $q$ describes how tightly (loosely) wound the helix of the magnetic field line is.
Actually, by making some assumptions on the current profile and approximating the torus with a periodic cylinder one can find [3] the following expressions for $q$ at the edge and at the core, respectively:

$$q_a = \frac{2\pi a^2 B_\phi}{\mu_0 I R_0} \tag{1.31}$$

$$q_0 = \frac{2B_\phi}{\mu_0 j_{\phi 0} R_0} \tag{1.32}$$

where $a$ is the minor radius of the torus, $B_\phi$ is the toroidal magnetic field component, $j_{\phi 0}$ is the toroidal current density at the core and $I$ is the plasma current.
Similarly, another important quantity is defined as:

$$\beta = \frac{p}{B^2/\left(2\mu_0\right)} \tag{1.33}$$

meaning the ratio of kinetic to magnetic pressure. Noticing that the triple product $n\tau_E T$ is proportional to $B^2\beta\tau_E$,it can be said that high values of $\beta$ are desired. In fact, together with other indicators, it represents a common figure of merit of the efficiency of the energy confinement.
In general, instabilities in a plasma can be classified as:

- current driven instabilities, which arise from gradients in the current density profile;
- pressure driven instabilities, which arise from the effect of pressure gradient and magnetic field curvature.

Morevoer, instabilities described within the framework of MHD (*ideal modes*) can be divided from those that depend on the finite plasma resistivity (*resistive modes*).
Ideal modes are the most violent instabilities in a plasma: they grow at really high rates and almost inevitably lead to disruption, meaning an abrupt loss of the plasma thermal and magnetic confinement, potentially damaging the device. On the other hand, resistive modes can be sustained by a plasma, which as a result of these instabilities becomes unstable but not necessarily terminates the discharge.
Resistive modes, nonetheless, are still an inconvenience: these instabilities can have negative effects on plasma confinement, by modifying the magnetic surface topology. If resistive modes manage to destroy magnetic surfaces, this gives rise to field lines which ergodically fill a region of space, which will become a region of poor confinement.
In both cases, these instabilities exhibit an infinite spectrum of modes, which depend, in the approximation of torus with large aspect ratio, on the poloidal and toroidal angles as follows:

$$\exp\left[i\left(m\theta + n\phi\right)\right] \tag{1.34}$$

with $m$ and $n$ called the poloidal and toroidal wavenumber, respectively.
Most unstable modes are those respecting the resonance condition:

$$q = -\frac{m}{n} \tag{1.35}$$

where $q$ is the safety factor as described above. This means that the perturbation wavefront is a helix with the same pitch as the magnetic field lines. When this occurs, the perturbation shifts all the points on a magnetic field line by the same quantity and no deformation of the field line happens.
Indeed, the magnetic field lines can be visualized as rods made of flexible materials: they are naturally opposed to increases in their curvature or compression against each other. This means that when a field line is curved by an instability, it intrinsically exhibits a stabilizing effect. On the contrary, when the resonant condition holds, no deformation is present and thus the

mode is not stabilized. The outcome is that resonant modes are more unstable and dangerous than non resonant ones.
A notable phenomenology linked to the resonance condition is the onset of the so called sawtooth oscillations, which are associated to the development of a resistive instability with $m = 1$ and $n = 1$. When the temperature at the core grows, resistivity decreases and the current density increases, causing $q_a$ (Eq. (1.31)) to become lower than one. At this point, the resonant mode surface is present inside the plasma and feeds the instability growth, which ruptures magnetic surfaces in the core, leading to an increase of energy transport and cooling of the core. This allows $q_a$ to become larger than one again, the magnetic surface to reconstitute and the cycle to begin again.
On the other hand, MHD instabilities can grow, destroying plasma confinement, and lead to the dramatic outcome of fast discharge termination, known as disruption. The consequence is a rapid release of the plasma thermal energy to the vessel of the tokamak.
Disruptions can lead to overheating and damage of plasma facing components; moreover, the fast shutdown of the plasma induces strong electromotive forces which drive large currents that interact with the magnetic field, leading to substantial structural damage.
It is thus of paramount importance that tokamaks operate far from the disruptive boundaries.
Situations giving rise to disruptions, as far as classical stability limits are concerned, are mainly two: the lowering of the edge $q$ below the critical value $q_a = 2$ or the growth of the plasma density above a certain threshold, called Greenwald density:

$$\bar{n}\left(10^{20}\mathrm{m}^{-3}\right) < \frac{I}{\pi a^2}\left(\mathrm{MA/m}^2\right) \tag{1.36}$$

A third operational limit related to instability development, for pressure driven instabilities, translates in a condition on $\beta$:

$$\beta\,(\%) < g\frac{I\,(\mathrm{MA})}{a\,(\mathrm{m})\,B_\phi\,(\mathrm{T})} \tag{1.37}$$

$$g = 3.5 \tag{1.38}$$

where $g$ is a factor coming from theoretical assumptions and is called Troyon factor. Often, in practice the normalized $\beta$ is used:

$$\beta_N = \beta\,(\%)\,\frac{a\,(\mathrm{m})\,B_\phi\,(\mathrm{T})}{I\,(\mathrm{MA})} \tag{1.39}$$

so that the condition becomes $\beta_n \leq g$.

### 1.2.6 Transport and confinement

In a plasma, energy losses can be driven by different phenomena. Most of the times, losses driven by turbulence phenomena are predominant and surpass those driven by collisions. This results in a difficulty in describing the relationship between the energy confinement and the machine plasma parameters, since energy losses driven by turbulence phenomena are not completely modelled.
In general, transport of energy, mass and momentum in the plasma can be of two forms:

- classical or neoclassical transport, driven by collisions, for which a predictive theoretical description can be given;
- anomalous or turbulent transport.

Anomalous transport is driven by small scale instabilities which do not have a destructive effect on the discharge.
Given the presence of turbulent transport, which is not theoretically formalized, predicting transport coefficients, and thus the confinement time, is in practice extremely complex. An attempt at squaring this issue lies in the development of empirical laws from experimental data, linking energy confinement time to plasma parameters.
There are various scaling laws, one of the most sophisticated being the so called ITER-89P:

$$\tau_E^{\mathrm{ITER89-P}}\,(\mathrm{s}) = \frac{I\,(\mathrm{MA})^{0.85}\,R_0^{1.2} a^{0.3} \kappa^{0.5} n\left(10^{20}\mathrm{m}^{-3}\right)^{0.1} B_\phi^{0.2} A_i^{0.5}}{P\,(\mathrm{MW})^{0.5}} \tag{1.40}$$

where $I$ is the plasma current, $R_0$ and $a$, respectively, the major and minor axis of the tokamak in metres, $\kappa$ the plasma elongation, $A_i$ the mass number of the ions (1 for hydrogen, 2 for deuterium) and P is the additional heating power.
The scaling law of Eq. (1.40) shows a dependence from the heating power $P^{-0.5}$. The regime in which this law holds is called *L-mode* where L stands for low confinement.
In 1991, the research group working on the ASDEX tokamak discovered a transition to an improved confinement regime, which could be obtained by reaching a sufficiently high level of additional heating power. In this regime, the energy confinement time is almost doubled with respect to *L-mode*; this regime is called *H-mode*, where H stays for high confinement.
Characteristic of the *H-mode* is the formation of a transport barrier in the edge region of the plasma, where steep gradients of density and temperature develop. As a consequence, density and temperature (especially at the core) are increased.
One of the most reliable scaling expressions since 1998 for the H-mode thermal energy confinement time is the so called IPB98y2 scaling which reads

as follows:

$$\tau_{\mathrm{th},98y2} = 0.0562\ I_p^{0.093} B_t^{0.15} n_{19}^{0.41} P_L^{-0.69} R^{1.97} \epsilon^{0.58} \kappa_a^{0.78} M^{0.19} \tag{1.41}$$

where $\tau_{\mathrm{th},98y2}$ is the confinement time in s, $I_p$ is the plasma current in MA, $B_t$ is the toroidal magnetic field in T, $n_{19}$ is the electron density in $10^{19}\mathrm{m}^{-3}$, $P_L \equiv P - dW/dt$ is the loss power in MW ($P$ is the heating power and $W$ is the stored energy), $R$ is the major radius, $\epsilon = a/R$ is the inverse aspect ratio (with $a$ the minor radius), $\kappa_a$ is the elongation and M is the ion mass number.

# Chapter 2

# TCV

The Tokamak à configuration variable (TCV) (Fig. 2.1) is a research fusion reactor of the Swiss Plasma Center (EPFL), located in Lausanne, Switzerland.

It is a medium sized machine with its main characteristic being that it

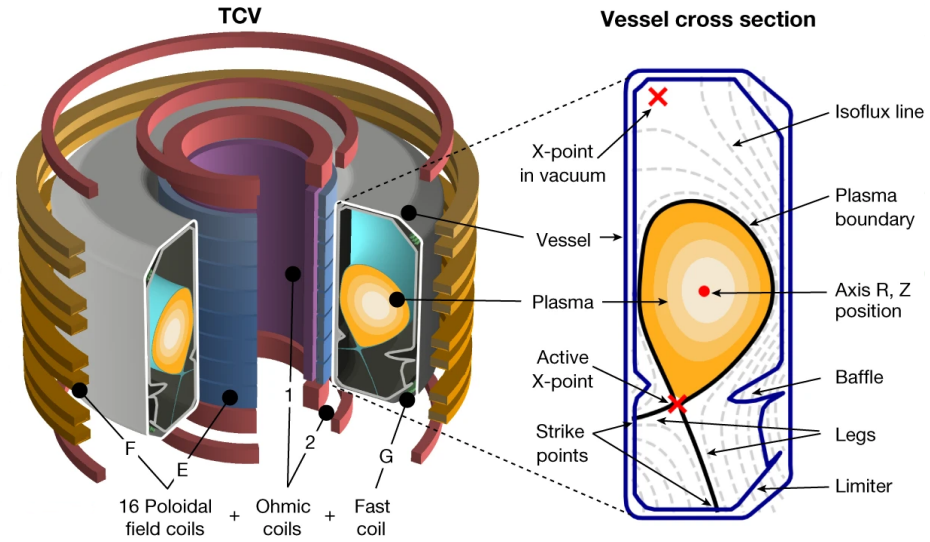


Figure 2.1: Representation of the TCV machine. Figure from [5]

can produce a vast array of plasma shapes, hence the name, which literally means "variable configuration tokamak". The main aim of TCV is to investigate effects of plasma shape on tokamak physics, so the machine has been designed such that it can produce diverse plasma shapes without requiring hardware modifications. [6]

TCV is also characterized by extreme flexibility in plasma positioning.

Such flexibility in positioning and shapes is granted by the shape of the

chamber which develops in height.
For plasmas with moderate elongation it is not difficult to produce a configuration which allows for a variety of plasma shapes. On the other hand, for highly elongated plasmas, a close fitting passive shell becomes necessary, and that is in conflict with the idea of variable shape.
TCV has thus been designed such as to represent a compromise between maximum shape variability and a good passive vertical stability. [6]
To satisfy these opposing requirements, TCV displays a number of unusual design features.
First, the poloidal field system consists of an ohmic heating (OH) transformer and 16 independently powered shaping coils located between the vacuum vessel and the toroidal field coils, which are supplemented by two internal coils to stem axisymmetric instabilities with high growth rates. [6][7]
The vacuum vessel is a continuously welded structure with low toroidal resistance (55p$\Omega$) and a nearly rectangular cross-section, with a height to width ratio of 3 (major radius of 0.88m and a minor radius of 0.25m). [6]
First wall protection is made of high purity, isotropic graphite tiles B211.
TCV sustains a vacuum toroidal field up to 1.5 T, and plasma current up to 1 MA. [7]
The focus of TCV recent experimental program is on a determined search for alternative and unconventional configurations in view of meeting one of the primary challenges for a DEMO reactor: the need to handle higher heat fluxes than ITER. [7]

## 2.1 Far InfraRed interferometer (FIR)

The knowledge of the electron density profile of tokamak plasmas is essential.
The Far InfraRed interferometer (FIR) is a diagnostic which measures the line-integrated electron density along parallel chords in the vertical direction.
On TCV, a 14-channel Mach-Zehnder type interferometer is employed.
TCV also present a state of the art Thomson Scattering (TS) diagnostic for temperature and density profiles reconstruction, which relies on good calibration. The FIR is also used to calibrate the TS. [8]
The FIR has an acquisition frequency of 10kHz, as opposed to the TS acquisition frequency of 20-60Hz.
The system is made up of a FIR (FarInfraRed with $CH_2F_2$ difluoromethane gas) laser, pumped by a $CO_2$ laser, and emitting a continuous wave at 184.3$\mu$m, and a multi-element detector unit (InSb hot-electron bolometer). [8]
The laser beam is divided into a reference beam, which is frequency shifted by a rotating grating, and 14 probe beams passing the plasma at different

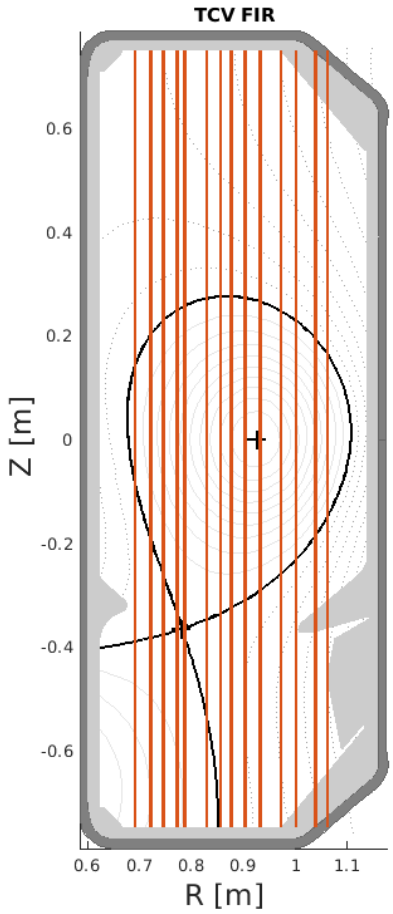


Figure 2.2: FIR diagnostic schematic

radial positions.
The reference laser beam is used as the refence with respect to which a phase dealy is computed. Namely, when the probe beams pass through the plasma, the difference in refractive index will cause a phase delay.
For the wavelength and polarization of the FIR beams, the refractive index of the plasma is directly related to the electron density. In this way, the system is able to provide continuous measurements of the line-integrated density along 14 chords.
The system is fully automated and part of the basic and essential diagnostic set in operation for each TCV shot.
The measurement along the central chord is used for real time control of the plasma density.

## 2.2 Thomson Scattering

The Thomson scattering (TS) system is the main diagnostic for the measurement of the spatial profiles of the electron temperature and density on TCV.
The working principle of the TS is based on the scattering of electromagnetic radiation from the plasma electrons, which allows under certain circumstances to measure electron temperature and density.
In particular, when a laser beam is sent through a plasma, the electrons are accelerated in the laser oscillating field and re-emit radiation.
Assuming that the wavelength of the laser is much smaller than the Debye

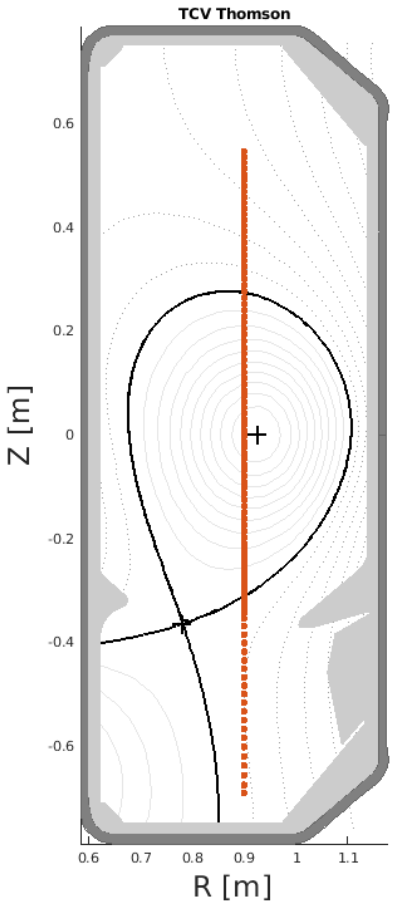


Figure 2.3: Thomson scattering (TS) diagnostic schematic

length, there is no correlation between the emissions from different electrons and therefore one speaks about incoherent Thomson scattering.

Under this assumption, the electron temperature can be determined from the broadening of the scattered radiation spectra while the density is proportional to the total scattered power.
A laser beam passes the plasma in vertical direction at R=0.9m (mid radius of the TCV vessel) and profiles are measured along such laser Fig. 2.3. There are multiple wide-angle camera lenses with the goal of collecting the scattered light from the observation volumes in the plasma and focusing it onto sets of fiber bundles.
Currently, there are 109 observation positions covering the region from $Z = -69$cm to $Z = +55$cm with a spatial integration length that depends on the channel location. Previously, the observation points were 89.
Measurements of the electron temperature over a range from 6eV to 20keV are permitted. Some plasma positions are more favourable than others.
The sampling rate of the measurements is determined by the repetition rate of the high-power Nd:YAG lasers, which emit at a wavelength of 1.06$\mu$m.[9]
The system is made up of 3 lasers combined in a cluster to build a beam which goes through the plasma as a narrow fan. Nonetheless, it appears as a single laser when viewed by the detection optics.
Using a combination of 3 lasers is favourable because it allows [9]:

- flexibility in timing. Indeed, each laser operates at a fixed repetition rate o 20Hz but triggering them separately with different delays allows to adapt sampling rates according to the measurements being done.

- flexibility in effective pulse energy. When performing measurements at low electron density it is required to have higher energies per laser pulse, or it wouldn't be possible to maintain the appropriate signal-to-noise ratio. Triggering lasers simultaneously helps increasing energy.

As anticipated when discussing the FIR diagnostic, for measurements of the electron density the system has to be absolutely calibrated.
A comparison of the line integrated density with the results from a multi-chord FIR interferometer is performed to derive a correction factor and obtain consistency. [9]

## 2.3 Soft X-rays

The Duplex Multiwire Proportional soft X-ray counter (DMPX) is a 64-channel soft X-ray detector located under TCV, viewing the plasma vertically.
Measurement coming from this diagnostic can be used as a proxy for electron temperature.
Soft X-ray emission, which is emission with energy typically under 50keV, in tokamak plasmas is composed of:

- Bremsstrahlung, from electron-ion Coulomb collisions
- electron-ion recombinations
- line emission

The power spectral radiation densities can be precisely derived for the phenomena listed above. [10] [11]
In the absence of metallic impurities in the plasma, the contribution of the line emission can be neglected for the energy range the diagnostic is sensitive to.
For the first two emission types, which compose the continuum of the soft

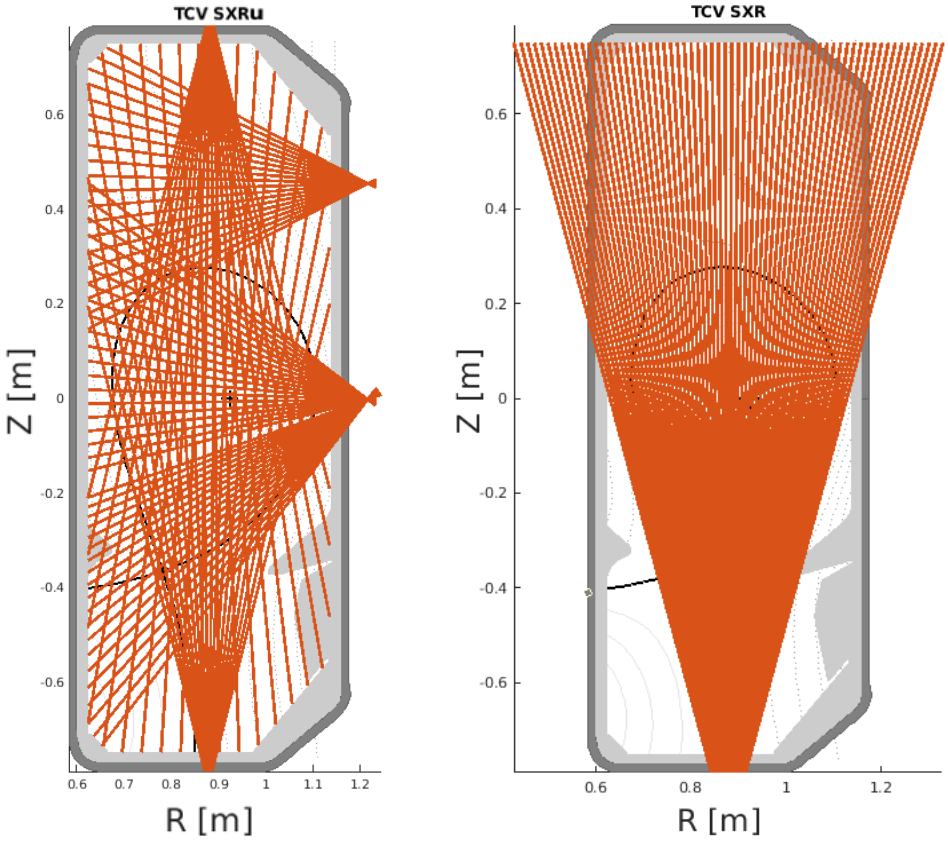


Figure 2.4: Soft X-rays diagnostic, on the right the most recent upgrade.

X-ray radiation, the local power spectral densities can be written as:

$$j(\nu) = 1.54 \times 10^{-38}\ n_e^2 Z_{\text{eff}}\ \frac{e^{-\frac{h\nu}{T_e}}}{\sqrt{T_e}} \cdot f(T_e) \quad [\text{Wm}^{-3}\text{eV}^{-1}] \tag{2.1}$$

where $T_e$ is the electron temperature in [eV], $f(T_e)$ is a complicated function of $T_e$ that is different according to the phenomena, $n_e$ the electron density

in [$m^{-3}$], $Z_{eff}$ is the effective charge, and $h\nu$ the energy of the photon in [eV].
Soft X-ray photons interact with the matter mainly by photo-electric absorption: they are absorbed by an atom which is thus ionized. The resulting free charges are then used for the detection of the incident soft X-ray.
The detection happens in a wire-chamber filled with gas where an electrical field is applied between an array of thin wires (anodes) and 2 plane cathodes, one on each side of the wire array. When the incident soft X-ray photon interacts with an atom of the detection gas, an electron and ion pair are produced and subsequently accelerated by the electrical field. While migrating towards the closest wire-anode, the electron further collides with other detection gas atoms and ionizes them triggering an avalanche process that takes place near the wire. As a result of this, the wire chamber measures a slight change in t(mainly the ions) migrating in the electrical field.
In Fig. 2.4 can be seen a schematic representation of the Soft X-rays detector on TCV.
The detector is made up of two superimposed wire-chambers sensitive to soft X-ray emission with energy between 3 and 30keV.
This diagnostic has high time- and space-resolutions. The acquisition is done at 200kHz and the mean distance separating two lines-of-sight is of 7.9mm for the top chamber and 16.3mm for the bottom chamber (when considering the equatorial plane of TCV vacuum vessel). The wire-chambers operate in the proportional regime, meaning that the measured signal, integrated along the lines-of-sight, is proportional to the mean power of incident soft X-ray flux.

# Chapter 3

# Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) which includes methods developed in the second half of the XX century in several scientific communities under different names, such as computational statistics, neural network, pattern recognition, data mining and many others.
The main characteristic of ML is that the machine that needs to solve the task is not specifically programmed for it but *learns* autonomously how to do it.
The learning process is made possible by data, which is also one of the reasons ML has recently seen a big development. Indeed, for these methods to work, large quantities of data are needed, and the computational power to handle them has only recently stepped up to the task.
The ML framework can be divided in sub-groups according to the way in which the learning process happens, depending on the kind of signal it is used to learn and the feedback the system ha access to. Based on this, one can individuate:

- **Supervised learning**; feed the model examples of the outputs associated to the inputs and expect it to learn the underlying rule associating input to output. Classic supervised learning tasks are regression and classification. [12] More formally, one says that supervised learning consists in training a model $f : x \to y$ such that $f(x_i) \approx y_i$ $\forall i = 1, ..., N$.
  This is achieved by minimizing iteratively a *loss function*, which intuitively represents the cost associated with the model; quantitatively it indicates the difference between the predictions of the model $f(x_i)$ and the known labels $y_i$.

- **Unsupervised learning**; the system is not fed any example of output, it is expected to learn exclusively from data. Typical examples of this kind of tasks are clustering problems. [13]

- **Reinforcement learning**; the system interacts with a dynamic environment with the goal of completing a task, the learning happens through a teacher which only says if the goal has been reached. This kind of algorithms are used for example to develop self-driving cars or artificial agents that can play (complicated) games with human counterparts. [14]

These different kind of tasks can be tackled with a variety of approaches, which can be most varied. For instance:

- Neural Networks (NN), an adaptive system which updates its structure based on the information flowing inside it
- Bayesian Networks (BNN), based on probabilistic inference
- Decision trees, which can be thought of as a collection of if-else rules
- Support Vector Machines (SVM), used to solve supervised classification, typically binary.

In particular, recently, NN have been receiving considerable attention from several fields of application. These algorithms have proved to be extremely well performing in a variety of tasks, including time-series prediction, classification, regression and pattern recognition [15]. They are used for facial recognition systems, data mining purposes and also financial and meteorological prediction [16].
The scientific community similarly manifests growing interest towards this kind of applications, which can represent an important catalyst for scientific discovery. The most famous example probably being AlphaFold, a neural network model by DeepMind [17] which is capable to accurately predict 3D models of protein structures.
The nuclear fusion community is also steering its attention in this direction and several works have been published which employ neural networks for a variety of tasks. [18] [19] [20] [21] [22] [23][24] [25] [26] [27] [5]
Different priority research opportunities have been recently discussed by various fusion international agencies such as IAEA (Technical Meeting on Artificial Intelligence for Nuclear Technology and Applications), the Department of Energy Offices of Fusion Energy Science (FES) and Advanced Scientific Computing Research (ASCR) and EUROfusion. Directions that have been indicated worth exploring include:

- Machine Learning Boosted Diagnostics, where machine learning methods are used to maximize the information extracted from measurements, systematically fuse multiple data sources and infer quantities that are not directly measured or cannot be easily computed in real-time during the experiments. Both supervised and unsupervised learning techniques can be employed: the former to be used on data ex-

tracted from the diagnostic measurements, the latter to explore the complex high dimensional data.

- Data-Enhanced Physics Informed Prediction, where fault and off-normal events detection, such as disruptions, are essential to be predicted, or mitigated in the worst case. Such predictions, properly combined with model-based approaches, can improve the response of the Plasma Control System.

Indeed, magnetic confinement fusion experiments lend themselves particularly well for a machine learning task. The produced data are complex, high dimensional and noise prone. Being able to employ ML models effectively in such a context could prove an actual game changer.

## 3.1 Neural Networks

(Artificial) Neural Networks (NN) are a computational model made up of artificial neurons, inspired by actual biological neural networks that constitute animal brains.
In these structures neurons are connected with each other, by so called *edges*, that model the synapses in the brain. Through these connections the neurons exchange signals, which are real numbers obtained by computing some non-linear function of the sum of the inputs to the neuron. The strength of the signal in the connection is decided by the *weight*, which adjusts as learning proceeds. Neurons are typically structured in layers; different layers may perform different transformations on their inputs.
The starting seed for NN was planted in 1943 by McCulloch and Pitts in a paper presenting a threshold linear combinator, accepting as input multiple binary data and with a single binary output; by combining many of such elements one can create a network which is able to compute some boolean functions. In the following years the study on the topic continued until the first neural network scheme, the perceptron, was introduced in 1958 by Rosenblatt. This marked a significant advancement, since the perceptron has variable synaptic weights and so it is able to learn. This means that the input is weighted according to weights that change in value as the learning process progresses.
In 1974 Werbos presented a way to train the Multi Layer Perceptron (MLP), which is a multi-layered network of perceptrons (namely an input layer, hidden layer with non-linear activation and output layer). Ten years later, in 1986, Hinton and Williams, based on the work of Werbos, developed the error backpropagation (BP) algorithm, which is still today amongst the best known and most efficient ways of training a neural network. The algorithm consists in modifying the connections amongst the neurons (weights) so that the network response gets closer to the desired one. Thanks to this new tech-

nique, more complex functions can be approximated, including the problem of non linearly separable data.

### 3.1.1 Training and inference

Training a neural network means iteratively updating the weights, moving in the direction that minimizes a loss function, which can take up different forms depending on the task one has to perform. The loss function defines the performance measure to be optimized during training.
A variety of algorithms can be employed to perform the training process, the most used one being stochastic gradient descent (SGD).
The gradient descent method allows one to find the local minimum of a function in an N dimensional space. The function under consideration will be the loss, of whose the gradient can be computed using the backpropagation algorithm introduced previously. BP is composed of two steps:

- Forward pass; in this phase, after a random initialization of the weights, the vectors in input are passed through all the layers of the network. The weights are fixed.

- Backward pass; this is the step in which by comparing the output of the network with the desired one, the error signal is computed. This error is back propagated through the network.

In stochastic gradient descent weights are updated based on an approximate value of the gradient (*e.g.* the value of the gradient of a single sample), while in gradient descent the update is performed with the value of the gradient over the whole dataset. As a compromise between computing the gradient of a single data point and that of the whole dataset, usually one computes the gradient over a subset of samples, called minibatch.
Assuming that the loss function can be decomposed as a sum over training examples, it can be written:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{data}} L\left(\boldsymbol{x}, y, \boldsymbol{\theta}\right) = \frac{1}{m}\sum_{i=1}^{m} L\left(\boldsymbol{x^{(i)}}, y^{(i)}, \boldsymbol{\theta}\right) \tag{3.1}$$

where $m$ is the number of training samples. Then, the gradients of the loss with respect to the weights $\boldsymbol{\theta}$ can be written as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{x^{(i)}}, y^{(i)}, \boldsymbol{\theta}\right) \tag{3.2}$$

Furthermore if the gradient is computed over a subset of samples of size $m'$ one writes:

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L\left(\boldsymbol{x^{(i)}}, y^{(i)}, \boldsymbol{\theta}\right) \tag{3.3}$$

Finally, the update rule for the weights is written as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g} \tag{3.4}$$

where $\epsilon$ is called learning rate and can be tuned to optimize learning. Learning rates can also be adaptive, changing their value according to current and past gradient values.
The basic training process described above can be arbitrarily enriched through the use of several methods. Amongst these are dropout layers, first and second order optimization methods (using first and second derivatives of the loss function), regularizers, weights initialization and adaptive learning rates.
The dataset used for training is called training dataset. Samples in this dataset should be as many as possible and come from the distribution one wants to model. Commonly used is also the so called validation dataset, which is employed to choose the best set of hyperparameters in the network. Hyperparameters are non learnable parameters which can be related to the architecture of the network (number of hidden units, number of hidden layers) or to more refined components (optimizers and regularizers tuning).
After performing training, the network is optimized to perform the devised task and one can move to the inference regime. Feeding previously unseen values to the network a correct prediction is expected as output. These new data are referred to as the test dataset, which remains strictly unseen by the network up until the moment inference is performed and the network performance is tested.
The assumption is that data used for training and test come from the same distribution, which is an important thing to keep in mind when constructing a dataset. If this hypothesis is not realized, performance will directly be affected.
In the real world it is impossible to precisely respect this request but still one should strive to be as close as possible to it.

Over the years, the family of NN has grown substantially, and now includes a variety of techniques which have reached state of the art performance in multiple domains. Amongst the main breakthroughs are convolutional neural networks for processing of visual data, long short-term memory for speech recognition and text-to-speech applications, generative adversarial networks for tasks which include winning a game or deceiving the opponent.

## 3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of neural networks conceptualized to process data that has a known grid-like topology. In particular, one can think of a time series as a 1D grid, or of an image as a 2D grid of pixels. CNNs employ a linear operation different from matrix moltiplication, called convolution, which can be defined in several ways, the most used one being:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{3.5}$$

which is written for a 2D input $I$ and kernel $K$, where the kernel values are the parameter to be learnt.
In Fig. 3.1 is represent graphically the operation of Eq. (3.5) for a 2D input.

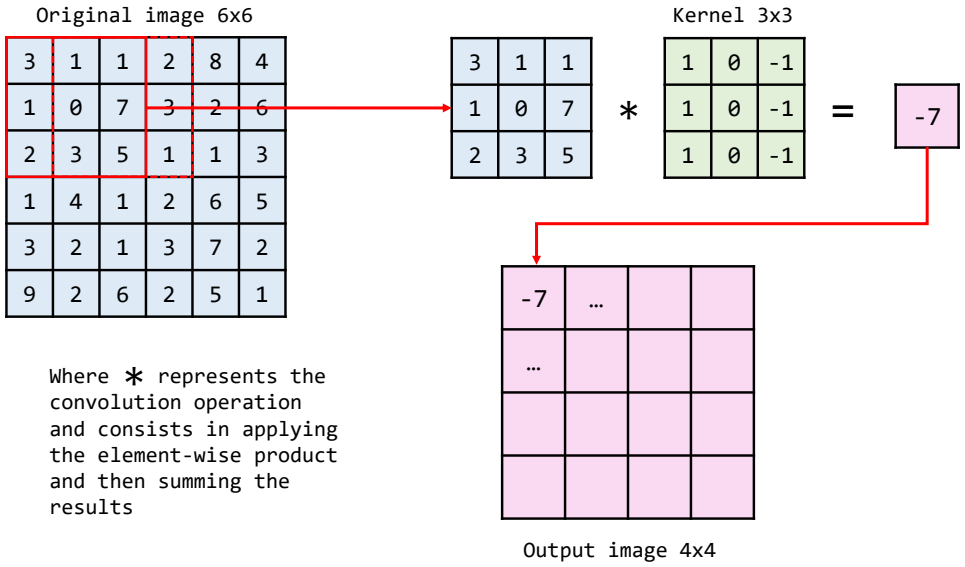


Figure 3.1: Representation of the convolution operation with a kernel in a CNN

Convolution is based on three main ideas that allow it to build efficient ML models and that are responsible for its wide usage and success: sparse interactions, parameter sharing and equivariant representations.
Fully connected neural networks optimize a weight matrix representing the interaction of each input unit with each output unit for every layer. On the other hand, CNNs have typically **sparse interactions** (connections), realized by making the kernel smaller than the input. This leads to the possibility of detecting small features in the input (*e.g.* edges in an image) and also to a reduction of the memory and computational cost (less information to store and smaller matrices to multiply).

Moreover, in deep CNNs, it may happen that units in deeper layers interact with a larger portion of the input. This results in the ability of describing complex interactions between many variables, being able to construct the description of such interactions from elementary building blocks which only describe sparse interactions.
Moreover, in a CNN the value of the weight applied to one input is related to the value of a weight applied elsewhere, leading to what is called **parameter sharing**. This results because of the application of the kernel, which slides on the input and is progressively convolved with it at every position. As a consequence, rather than learning $m \times n$ parameters as in a fully connected network, only $k$ parameters need to be optimized (and stored), where $k$ is the number of elements of the kernel.
Lastly, CNN layers represent the only way in which we can linearly implement **equivariant representations** in neural networks. The existence of equivariant representations is what allows models that employ CNN layers to develop edge detectors in images or specific event detectors when processing time series.
An equivariant mapping is a mapping which preserves the algebraic structure of a transformation; in particular, in the case of a translation, one practically has that when the input is translated of a certain amount the output is translated of the same amount.
Thinking in terms of neural networks, let's consider an image classification task which resulted in a feature mapping $\phi$; we also have an input image $X_1$ that we translate by means of a transformation $T$, obtaining an image $X_2 = T(X_1)$. If the mapping $\phi$ is translationally equivariant, then $F_2 = \phi(X_2) = T(F_1) = T(\phi(X_1))$. It can be demonstrated that by requiring the existence of a neural network layer thought of as an operator that is linear and shift invariant, the expression of a convolution layer as described in Eq. (3.5) can be derived [28].
The three motives described above are the reasons for which convolution is used. Architectures that employ CNN layers are under constant evolution, becoming more and more complicate, having, in general, around a million parameters. Such models present a variety of layers and can be specialized to many different tasks. Nonetheless, the basic building blocks are the ones presented in this section.
As for what concerns training of CNN, it is carried out making use of backpropagation algorithm. Most commonly employed is minibatch stochastic gradient descent (SGD) with the loss function to be optimized varying greatly depending on the task.

## 3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are neural networks specialized in dealing with sequential data. RNN can scale to sequences that are much longer than those that can be processed by networks which are not specialized in this sense.
Recurrent networks make use of parameter sharing across different parts of the model. This makes it possible to extend and apply the model to examples of different forms, meaning that, for example, in making predictions one is not restricted only to sequences with the same length of the ones used for training. This feature is also very important to be able to detect if an event is happening, whether it is found at the beginning or the end of the sequence.
Let us note that parameter sharing in RNN is different from the one found in CNN. For convolutional layers, the sharing of parameters across time lies in the fact that the same convolutional kernel is used at each time step. Nonetheless, this kind of parameter sharing not as refined as in RNN, since each piece of the output sequence is function only of a small number of neighboring pieces of the input.
On the other hand, in RNN each member of the output is obtained using the same update rule of the previous output; this means that each output is a function of the components of the previous output.
Given the recurrent nature of these networks a useful way to construct a representation is by unfolding them into a computational graph which sequentially displays the chain of events. This description directly shows the sharing of parameter across the network structure. In Fig. 3.2 an example

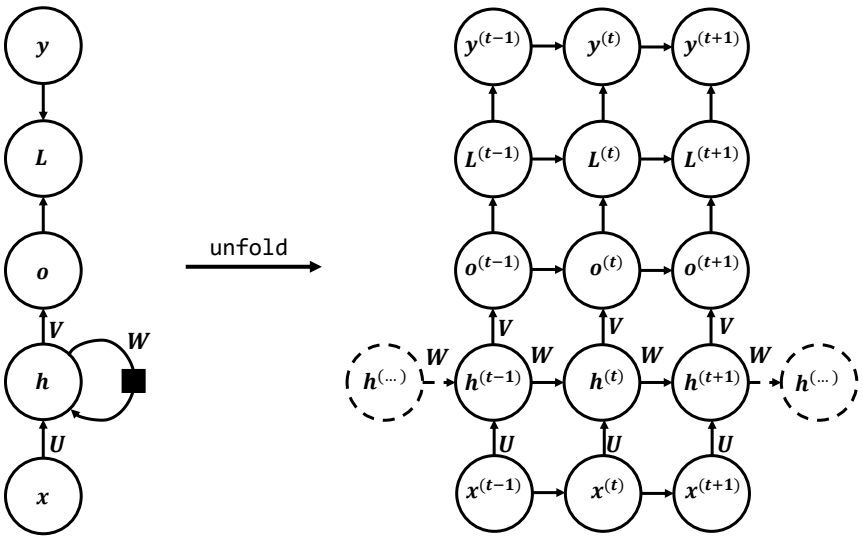


Figure 3.2: Representation of the unfolding of the recurrent connections in an RNN

of a RNN with connections between input-hidden units (parametrized by the matrix U) and hidden-output units (parametrized by the matrix V) can be seen; it also features recurrent connections between hidden-hidden units, parametrized by the matrix W. The black square indicates the passing of one time step.
On the right the unfolded graph is presented; one can clearly see the parameter sharing in the fact that the matrices U,V and W are the same across the whole graph.
The network represented in Fig. 3.2 produces an output at each time step and presents recurrent connections between hidden units. In general, RNN can be designed in different ways. For example, they may produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step or they could read an entire sequence and produce a single output, with recurrent connections between hidden units.
The unfolded graph representation also allows us to realize that the training of RNN can be performed by means of backpropagation, where the gradients are iterated backwards through time; from this, the method is called backpropagation through time (BPTT). This kind of algorithm cannot be performed in parallel, since the forward pass is inherently sequential: for the computation of each time step one needs the previous one. Moreover, states computed in the forward pass need to be stored to be used during the backward pass, so also the memory cost is considerable ($O(\tau)$, with $\tau$ length of the sequence).
Nonetheless, even considered the shortcomings of BPTT this method remains the most widely used training algorithm for RNN; other methods, such as Reservoir Computing [26], have been developed but are not as efficient at BPTT.
One big issue of RNN lies in the capability of learning long-term dependencies. This problem arises because gradients propagated over many time steps tend to vanish or, more rarely, explode. Moreover, even if one assumes to be able to solve this problem, it remains the fact that the weights given to long-term dependencies are exponentially smaller than the ones given to short-term ones.
Specifically, in RNN the same function is composed over many time steps. This composition can be treated as a matrix multiplication. For example if one considers for simplicity a network with only recurrent connections between hidden units, no input and no activation function the update relation can be written as:

$$h^{(t)} = W^{\intercal} h^{(t-1)} \tag{3.6}$$

if the recursive relation for $h^{(t-1)}$ up to $h^{(0)}$ is used and one assumes that W is diagonalizable it is clear it can be written:

$$h^{(t)} = (W^t)^\intercal h^{(0)} = Q^\intercal \Lambda^t Q h^{(0)} \tag{3.7}$$

The update rule depends directly on the value of the eigenvalues raised to the power $t$: for large $t$, those smaller than 1 will eventually decay to 0, those bigger than 1 will explode. The result will be that all components of $h^{(0)}$ which are not aligned to the largest eigenvalue will eventually vanish.
One may think that the problem could be easily solved by staying in regions of the parameter space where gradients do not vanish or explode. Unfortunately, to model long-term dependencies it is necessary to enter these region of space. This means that it is not impossible to train networks to model long-term dependencies, only that it will take a long training time given that the signal about the long-term dependency is vanishing exponentially. In practice, this results in unsuccessful attempts at training RNN with SGD already for sequences off length 20.
To overcome this issue, a variety of approaches have been proposed, such as Long Short Term Memory (LSTM), but learning long-term dependencies remains one of the biggest challenges in deep learning.

### 3.3.1 Long Short Term Memory

Long Short Term Memory (LSTM) models are a kind of gated RNN, designed to solve the problem of vanishing or exploding gradients and thus being able to learn long-term dependencies in sequences.
The main idea is to create paths through time for which one does not encounter vanishing derivatives. In order to do so, gated units operate on the value of the weights at each time step, modifying them. In particular, the focus is not only on being able to store information for a long duration, but also on the capability of recognizing which information has become irrelevant and discard it.

The repeating module of a LSTM network is more complex than that of a standard RNN. There can be many different variants of LSTM units, the standard one being composed of 4 neural network layers, interacting between them. In Fig. 3.3 a representation of such LSTM unit is displayed.
The flow of information inside the cell is regulated by the cell state $C_t$, which is represented by the top horizontal black line. The LSTM can add or remove information from the cell state by means of structures called gates. These gates are composed of a sigmoid/tanh layer and a pointwise multiplication operation.
Referencing Fig. 3.3, we'll walk through the functioning of a LSTM cell step by step. Starting on the left, the first layer one encounters is the so called forget layer, which takes care of throwing away information that is no longer relevant. By looking at the previous hidden layer $h_{t-1}$ and current input $x_t$,

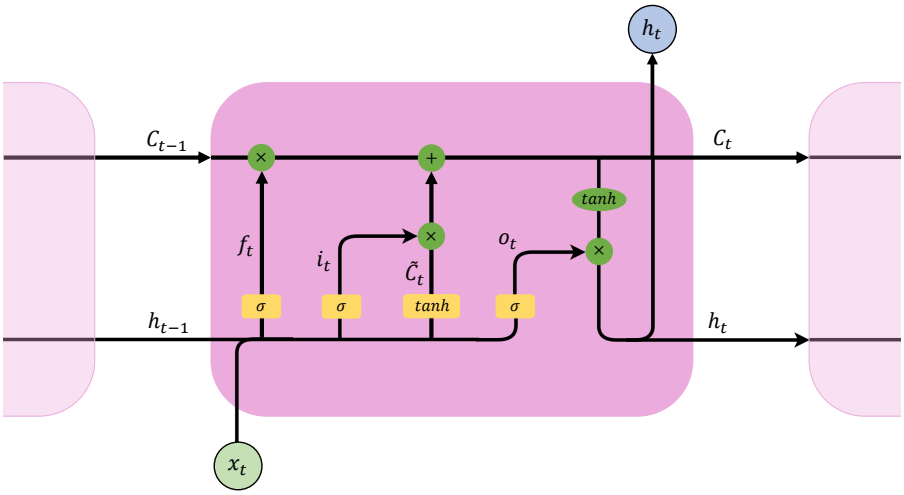


Figure 3.3: Representation of the LSTM cell

it outputs a number between 0 and 1 for each number of the previous cell state $C_{t-1}$. The equation is as follows:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{3.8}$$

After having decided what to forget, the next step is to decide which parts of the new input are relevant to the cell state. This is handled by two layers. The first is a sigmoid layer (the input gate) that decides which values will be update, while the second is a tanh layer that takes care of producing the vector of new candidate values $\tilde{C}_t$ that could be added to the cell state. To create the update to the state these two outputs will be combined. the equation for this second step is:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{3.9}$$

$$f_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{3.10}$$

Given the quantities just computed, one can proceed to update the cell state as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.11}$$

After having updated the state, generating the output is next. In order to do so some filtering is added to the cell state, as follows:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \tag{3.12}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{3.13}$$

Once again, the sigmoid layer decides which parts of the cell state $C_t$ are going to be output and the tanh takes care of pushing the values of the cell state between $-1$ and 1.

## 3.4 Autoencoders

Autoencoders are neural networks devised to reproduce the input as output. The basic structure of an AE can be observed in Fig. 3.4.

They are composed of two parts, an encoder function $h = f(x)$ and a de-

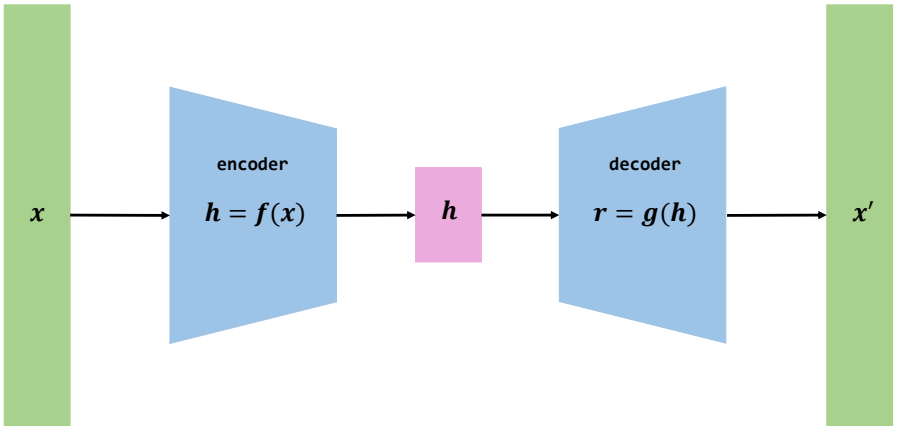


Figure 3.4: Schematic representation of an autoencoder

coder function $r = g(h)$. Autoencoders also feature the presence of a hidden layer $h$, which should learn a useful representation of the input.

Clearly, obtaining perfect reconstruction $(g(f(x)) = x)$ everywhere is not particularly useful so AE are designed to be unable to copy the input perfectly. This forces the model to learn the key charachteristics of the input, since it cannot simply reproduce it without loss.

Moreover, one speaks of overcomplete autoencoders whene the hidden layer has a dimension equal or greater than the input. Actually, the more interesting case is when the autoencoder is undercomplete, so that the hidden layer constructs a low dimensional representation of the input.

Training of an autoencoder happens with the same methods employed for feedforward neural networks, namely descending gradients through backpropagation. The goal is to minimize the reconstruction error, which is often expressed through the mean square error (MSE) as:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\|\hat{y}^{(i)} - y^{(i)}\|^2 \tag{3.14}$$

Autoencoders with linear activation functions and MSE as reconstruction error actually learn to span the space space as PCA. If one uses nonlinear activation functions, the representation learnt is a more powerful generalization of PCA.

Autoencoders can be constructed to deal with several different kinds of inputs and they can be composed of different neural network layers, from convolutional to recurrent to simple feed forward ones.

# Chapter 4

# Data

Constructing the dataset to train the model is a fundamental step which needs to be carried out carefully.
To construct the samples for the neural network model data that are coming from the TCV machine are employed.
The plasma discharges that are considered represent several experimental conditions: no filtering based on the experimental configuration has been made.
In table Table 4.1 are displayed the signals considered for this work, indicating the diagnostic that produced them, a brief description and the unit of measure. Some of the diagnostic mentioned in Table 4.1 are presented in Chapter 2, such as Thomson Scattering (TS), Far Infra-Red interferometer (FIR) and Duplex Multiwire Proportional soft X-ray counter (DMPX). As for the others, the foil bolometers (BOLO) are a diagnostic which detects the total incident power from both photons and neutral atoms; LIUQE is a tokamak equilibrium reconstruction code. Let it be noted also that for powers_calc the diagnostic column is empty: this feature is computed starting from other values as follows:

$$\text{powers\_calc} = P_{\text{OHM,calc}} + 10^6 P_{\text{NBI}} + 10^6 P_{\text{ECRH}} \tag{4.1}$$

where $P_{\text{OHM,calc}}$ is the ohmic power, $P_{\text{NBI}}$ and $P_{\text{ECRH}}$ are, respectively, the NBI and ECRH injected power.
Due to the extreme experimental conditions, even if the diagnostics employed are state of the art, the hardware can undergo failures or produce incorrect data because of various reasons. Hence, raw data coming from the diagnostics need to be cleaned and pre processed before being passed as input to the neural network.
This process has been executed carefully in subsequent steps, which are now going to presented in detail.

| **Feature** | **Diagnostic** | **Description** | **Unit of measure** |
|---|---|---|---|
| FIR | FIR | Averaged line-integrated density | $m^{-3}$ |
| WP | LIUQE | Plasma stored energy | J |
| powers_calc | | Total input power - losses | W |
| PradTot_RT | BOLO | Total radiated power | kW |
| KAPPA | LIUQE | Elongation | a u |
| DELTA | LIUQE | Triangularity | a u |
| Q95 | computed | Edge Safety factor | a u |
| H98y2calc | computed | ITER H98y2 scaling factor | a u |
| P_LH_RT | computed | LH power scaling | W |
| LI | LIUQE | Internal inductance | a u |
| VOL | LIUQE | Plasma volume | $m^3$ |
| AREA | LIUQE | Plasma poloidal cross section area | $m^2$ |
| BETAN | LIUQE | Normalized beta | a u |
| Te_rho | TS | Electron temperature; norm. flux coord. $\rho \in [0, 1]$ | eV |
| Ne_rho | TS | Electron density; norm. flux coord. $\rho \in [0, 1]$ | $m^{-3}$ |
| SSX | DMPX | Soft X-ray emission; norm. flux coord. $\rho \in [-1, 1]$ | $W/m^3$ |

Table 4.1: Summary of the features used in this work

## 4.1 Data structure

After a shot (discharge) of the TCV machine, raw data coming from the diagnostics are processed and stored in the MDSplus database. Developed jointly by the Massachusetts Institute of Technology, the Fusion Research Group in Padua, Italy (Istituto Gas Ionizzati and Consorzio RFX), and the Los Alamos National Lab, MDSplus is the most widely used system for data management in the magnetic fusion energy program [29]. It consists in a set of software tools for data acquisition and storage and a methodology for management of complex scientific data. The system was designed to enable users to easily construct complete and coherent data sets.
Various codes are run, to output interpretable signals and calculations of parameters and derived quantities are carried out. Fit codes reconstruct profiles.
Starting from these raw signals, by using a set of MATLAB routines, multiple Apache Parquet files are generated, to keep track both of the actual value of the signal and of the coordinate system of the diagnostic with which it has been acquired.
The MATLAB routines employed in this work are part of what is called dis_tool software [30], which is officially employed for the EUROfusion Disruption Database and various international collaborations. Moreover, most of the MATLAB routines developed specifically for this thesis have been completely integrated in the dis_tool software.
The Apache Parquet format has been chosen for its several advantages. First of all, it is a language agnostic format: this bodes well with the fact that these files are being used by several different collaborations which may not all use the same programming language to access the data. Moreover, Apache Parquet lends itself well for column-based files, allowing a per-column access as well as very efficient row filtering, so that one can skip over the non-relevant data very quickly. Furthermore, it provides highly efficient data compression and decompression as well as occupying less storage space than other tabular formats (*e.g.* CSV).
For each discharge, two parquet files are available; for a shot number `000000`, they are named as follows:

- `TCV_DATAno000000.parquet`

- `TCV_COORDno000000.parquet`

The `TCV_DATA` files contain the actual values of the signals. The data are stored so that the columns are the different diagnostics and each row is a time instant; the sampling frequency is 10 kHz for all the signals (they have all been resampled to the same frequency by means of one of the MATLAB routines quoted above). The first column of each file carries the time information.

For the multichannel diagnostics multiple columns are present: one for each channel. In the case of the Thomson Scattering (TS) ans Soft X-rays (SSX) the reported values do not refer to the actual physical channels but to a remapping according to the normalized flux coordinate $\rho$, varying between 0 (magnetic axis) and 1 (plasma boundary) or between $-1$ (plasma boundary on the High Field Side HFS) and 1 (plasma boundary on the Low Field Side LFS). After this procedure, one obtains:

- for TS, 200 channels with $0 \leq \rho \leq 1$. The distribution of channels is not uniform: indeed, 65% of the available channels is devoted to the pedestal region, which by covering $0.8 < \rho < 1$ represents only 15% of $\rho$ range of values.
  Of these 200 channels, only 1 every 3 is retained, for a total of 67 channels.
- for SSX, 51 channels with $-1 \leq \rho \leq 1$.

Moreover, the data from TS is not the raw signal of the diagnostic but the output of a fit routine called LIUQE.
The `TCV_COORD` file contains information about the multichannel diagnostics which involve information about the coordinates of line measurements (all of them except for TS). Specifically, for each channel, for each time instant, are reported the two points of intersection of the LCFS, which identifies the plasma boundary, with the line of sight of the diagnostic. Given the fact that in TCV the plasma shape and position in the chamber can vary considerably, a specific line of sight can at one time cross a considerable portion of plasma and at another barely touch it. This information is thus necessary to interpret the line integrated values available for these kind of diagnostics.
Lastly, a file in JSON format, `TCV_diags.json`, contains the geometric mapping of all the diagnostics, based on a reference system with origin in the center of the tokamak and axis $R$ and $Z$ (horizontal and vertical, respectively).
Given the set of all possible diagnostics available (in principle) during a certain shot, not all discharges actually have the same diagnostics configuration available and selected; as stated above, the experimental conditions are quite extreme and malfunctioning can be expected. In some cases, the files are missing some columns, due to the fact that for that particular shot the signal from some diagnostics may not have been available. This calls for a first post processing step: all the parquet files are rewritten with the same set of columns, adding columns of NaN in the cases with missing diagnostics.

## 4.2 Data cleaning

At this point of the processing, data may contain various elements which cannot be digested by a machine learning model: values which are not avail-

able, infinite ones and values which are simply not physical.
By using a MATLAB routine, this issue is tackled in the following order:

1. Remove the infinite values (`Inf`) and substitute them with `NaN` (not a number). Infinite values are the result of saturated diagnostic or general malfunctioning at a specific time instant. As for what concern their occurrence, they are not at all frequent in the data.

2. Non physical, non plausible or non admissible values are present in the data. A nonphysical value could be the density or injected power $\leq 0$; a very unlikely one could be the safety factor bigger than 10 or the $\beta_N \geq 3.5$, with

$$\beta_N = \beta \frac{aB_T}{I_p} \tag{4.2}$$

$$\beta = \frac{\langle p \rangle}{B^2/2\mu_0} \tag{4.3}$$

   where $a$ is the minor radius in m, $B_T$ is the toroidal magnetic field in T, $I_p$ is the plasma current in MA and $\beta$ is a coefficient expressing plasma performance, given by the ratio of the mean plasma pressure to the magnetic pressure associated to the mean total field strength.
   For each feature, a range of admissible values has been chosen, both by making physical considerations and by plotting the empirical cumulative density function (ECDF), discarding values for which the curve has a value $\geq 0.9995$.
   In some cases, in order to be able to decide on a range, it has been necessary to first apply a transformation (logarithmic) to the values of the feature.
   Let it be noted that, at this point of the workflow, the modulus of the toroidal magnetic field was computed, and consequently the features which use the value of the magnetic fieldcarry this information.
   Apart from removing values which are non physical, this fine procedure of individuating allowed values is needed to cut long tails from the distribution of the feature; long tails notoriously make machine learning tasks significantly more difficult. Moreover in this work, by cutting these long tails extreme values of the features are cut, which are particularly exotic and thus are not affecting the applicability of the model.

3. As a last step, the `NaN` values are taken care of. These can derive from many situations (the diagnostic missed some data, that specific diagnostic wasn't used for that specific shot).
   `NaN` are treated differently based on where they are located inside the shot. Two possible cases present:

- The `NaN` are present in sequence at the beginning or/and at the end of the shot. This is the case of a diagnostic that, for example, hasn't started data acquisition until a certain time instant or that has ceased working before the end of the shot. These sequences of `NaN` are left untouched. The alternative would be to substitute them with the first (last) available value, but this would mean modifying the data distribution, even significantly if the sequence are long (as it is the case for some shots).

- The `NaN` are in the middle of the shot, enclosed by available values. In this case, linear interpolation is performed to retrieve the values and being able to use the data to subsequently construct samples. Linear interpolation could seem a bold choice, since it uses both past and future information to fill missing values. If then one uses this dataset to train a network with time series (as it will be the case in this work) it could be argued that the future information is revealed by the filled values and therefore the network could have an unfair advantage in the learning process, which wouldn't be available in a real-time setting.
  Nonetheless, this is not the case here: no actual future information is being used. Future information is used, for example, when performing a moving average with a window that sits across the current time step $i$: after applying this operation the time step $i$ contains information about $i + n$, where $n$ is half the width of the window. On the contrary, in our case, there is a missing diagnostics of which we need to reconstruct the signal in a way which is as close as possible to what it would have been, had the diagnostic not failed.The only use of future information is to retrieve the shape of the signal where it is not available, but it is simply a reconstruction. Furthermore, the alternative to interpolation would be to substitute the `NaN` with the value of the last available time step; in cases with long `NaN` sequences this could mean ending up with data which are unusable to construct proper training samples (they are not representative of how that feature actually evolves in time).

Let us note that, even if for the specific application in this work not all of the signals have been employed, the cleaning procedure described above has been carried out for all available signals.
In Fig. 4.1, Fig. 4.2, Fig. 4.3 and Fig. 4.4 you can see an example of the normalization process, for the total radiated power PradTot_RT (Table 4.1).

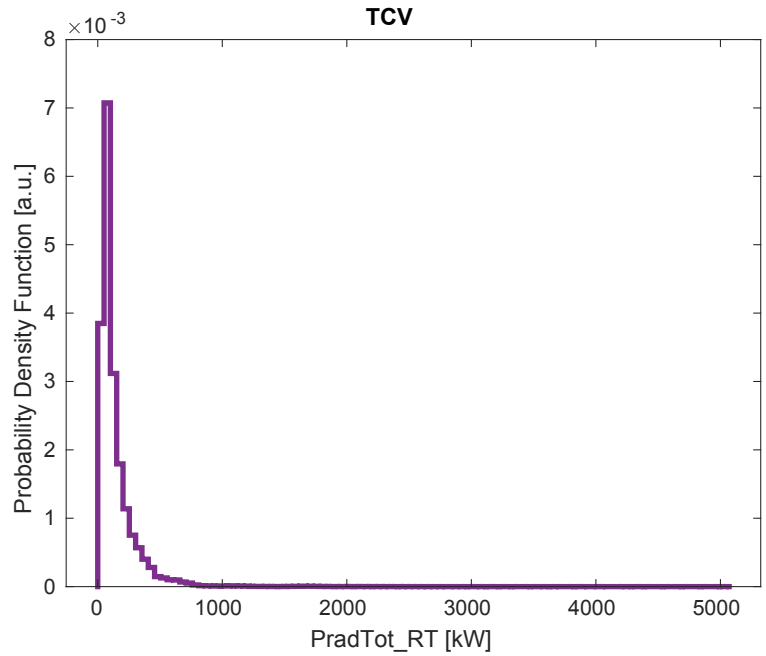


Figure 4.1: Probability density function for PradTot_RT (*cfr* Table 4.1), for 1886 discharges.

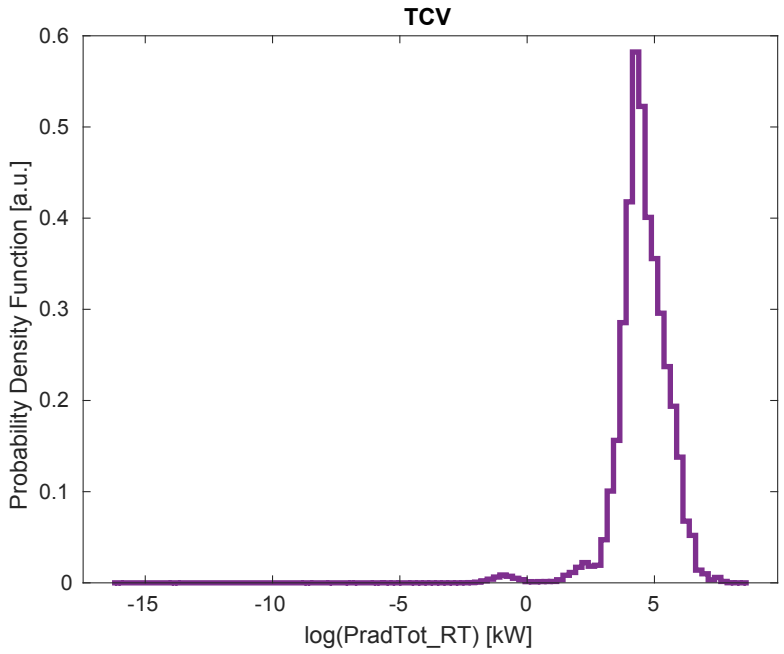


Figure 4.2: Probability density function for log (PradTot_RT) (*cfr* Table 4.1), for 1886 discharges.

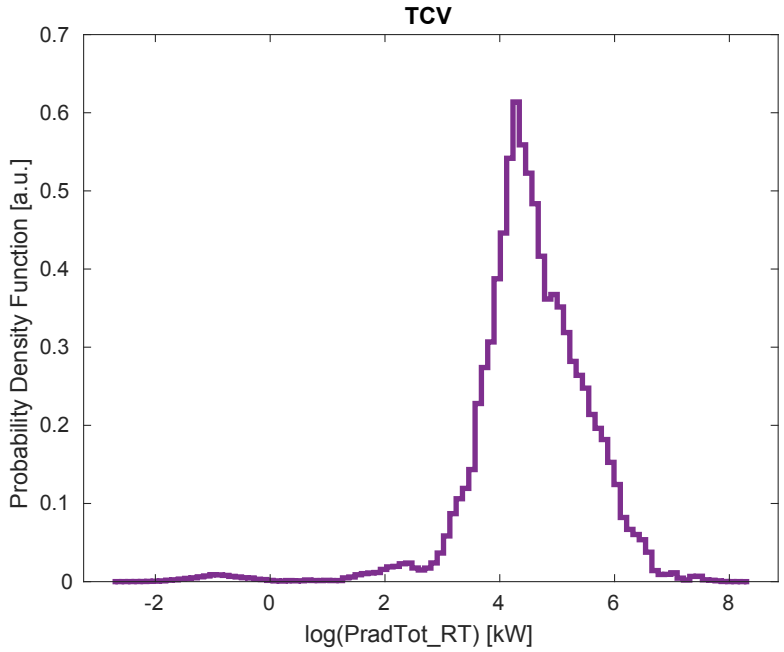


Figure 4.3: Cut probability density function for log (PradTot_RT) (*cfr* Table 4.1), for 1886 discharges.

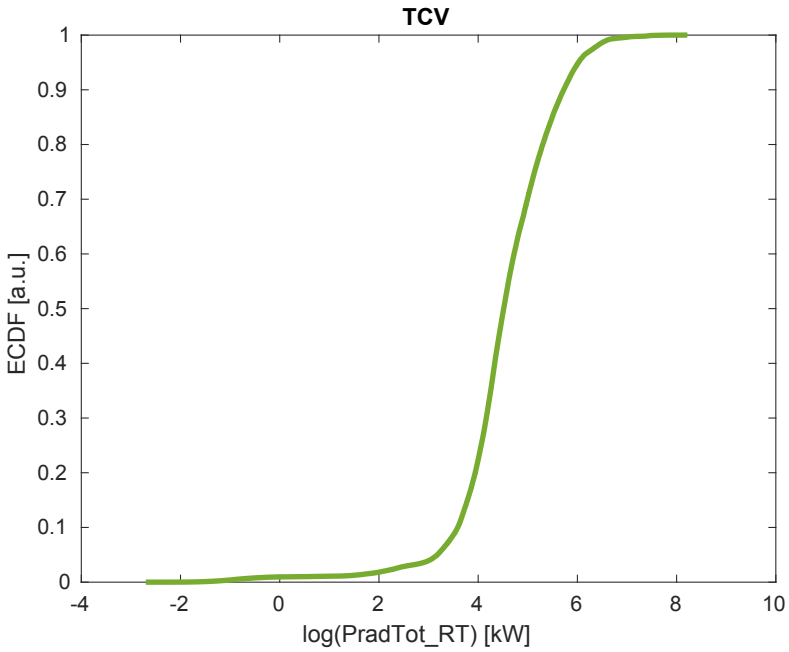


Figure 4.4: Empiric cumulative density function (ECDF) for log (PradTot_RT) (*cfr* Table 4.1), for 1886 discharges.

## 4.3 Data normalization

The final step of preprocessing consists in normalizing the data. This part of the procedure has been carried out only for the signals that have been employed for this specific work. For more details on the chosen signals refer to Table 4.1.
Normalization is an essential step which involves the transformation of features in a common range.
This allows to deal with two main issues of data which inhibit the learning process of ML algorithms: the presence of dominant features and outliers. After normalization, features that have greater numeric values cannot dominate the ones which have smaller numeric values [31]. In this work, for example, before normalization the density of the plasma is in the order of $10^{19}$ m$^{-3}$, while the parameter $\beta_N$ (Eq. (4.2)) is in the range $[0, 3.5]$; training with this data would clearly result in bad performance of the network.
There exist several different normalization procedures, which normalize the data within a certain range based on different statistical measures from the raw data.
In this work, the type of normalization was individually picked for each feature. This choice is justified by the fact that the distribution of the features across the dataset is quite varied. Each of them has different pathologies: some present more outliers, others have some tails.
In particular, the distribution of each signal is plotted across the whole dataset of discharges, normalized according to seven possible normalization methods. Then, a choice can be made based on how the un-normalized and the normalized plots look like in terms of solving the above stated pathologies and modifying the variance of the distribution.
The normalization strategies considered are as follows:

1. **Mean and standard deviation based normalization methods**; in this case the measures employed are the mean and standard deviation of the raw data. Amongst the pool of existing variations are chosen:

   - *Z-score (ZS)*; the resulting features have zero mean and a unit variance. Given an instance $n$ of feature $i$, $x_{i,n}$, the transformation lends:

   $$x'_{i,n} = \frac{x_{i,n} - \mu_i}{\sigma_i} \tag{4.4}$$

   where $\mu_i$ and $\sigma_i$ represente the mean and standard deviation of the $i$-th feature.

   - *Pareto Scaling (PS)*; analog to the ZS but the scaling factor is now the square root of the standard deviation. The new feature thus have

a variance equal to the standard deviation of un-normalized features.

$$x'_{i,n} = \frac{x_{i,n} - \mu_i}{\sqrt{\sigma_i}} \tag{4.5}$$

This method minimizes the contribution of noise in the data and improves the representation of lower concentrated features. Furthermore, it keeps the structure of data partially intact. [31]

- *Variable stability scaling (VSS)*; this is an extension of the ZS, with the introduction of the Coefficient of Variation (CV) as a scaling factor. Such coefficient is defined as the ratio of the mean to its standard deviation, so that:

$$x'_{i,n} = \frac{x_{i,n} - \mu_i}{\sqrt{\sigma_i}} \frac{\mu_i}{\sigma_i} \tag{4.6}$$

The scope of the coefficient is to give a higher importance to features with a small standard deviation and consequently lower importance to those that have a large one.

2. **Minimum–Maximum value based normalization methods**; in these methods the maximum and/or minimum of the raw data are used for rescaling. Amongst all the possibilities are chosen:

- *Min-Max (MM)*; data are scaled linearly to predefined lower and upper bounds. The range of rescaling is usually $[-1, 1]$ or $[0, 1]$. Generally:

$$x'_{i,n} = \frac{x_{i,n} - min(x_i)}{max(x_i) - min(x_i)} (nMax - nMin) + nMin \tag{4.7}$$

where $min$ and $max$ represent the minimum and maximum of the $i$-th feature and $nMax$, $nMin$ the new maximum and minimum values after normalization. In this work, $nMin = 0$ and $nMax = 1$, so the range is $[0, 1]$.

3. **Median and median absolute deviation normalization**; for this methods the values of the median and the median absolute deviation from the raw data are employed. The transformation of $x_{i,n}$ is:

$$x'_{i,n} = \frac{x_{i,n} - med(x_i)}{MAD(x_i)} \tag{4.8}$$

where $med$ represents the median of each $i$-th feature and $MAD = med(|x_{i,n} - med(x_i)|)$. This procedure is analogous to the ZS one, using the median as a statistical property, as opposed to the mean. This results in enhanced robustness towards outliers and extremes, given the insensitivity of the median towards such values. Nonetheless, this method fails to rescale the data to a common numerical range when data varies with respect to time.

4. **Sigmoidal normalization**; this method consists in a non-linear transformation of the data and it is particularly suited for reducing the effect of outliers. The two main sigmoidal functions usually employed are:

   - *Hyperbolic tangent (HT)*; this normalization transforms the raw data in the range $[-1, 1]$ using the hyperbolic tangent function. It is as follows:

$$q_{i,n} = \frac{x_{i,n} - \mu_i}{\sigma_i} \tag{4.9}$$

$$x'_{i,n} = \frac{1 - e^{-q_{i,n}}}{1 + e{-}q_{i,n}} \tag{4.10}$$

     This kind of normalization helps to scale the outliers without affecting the rest of the data, which are scaled linearly. It prevents the data to be pushed in a narrow range due to the presence of outliers.

   - *Logistic sigmoid (LS)* or *SoftMax*; this normalization transforms the raw data in the range $[0, 1]$ using the logistic sigmoid function.

$$q_{i,n} = \frac{x_{i,n} - \mu_i}{\sigma_i} \tag{4.11}$$

$$x'_{i,n} = \frac{1}{1 + e{-}q_{i,n}} \tag{4.12}$$

     This kind of transformation is useful when data are not evenly distributed around the mean: outliers which lie away from $\mu_i$ are squashed exponentially.

   The main advantage of these normalizations is the ability of mapping the outliers along the tails of the given range (dependent on the function). Meanwhile, data that lie within a standard deviation of the mean are mapped to an almost linear region, preserving their significance.

In the following table (Table 4.2) each feature and the chosen normalization are reported. For the multichannel diagnostics each channel has been normalized with the same method.

| Feature | Normalization | Range |
|---|---|---|
| FIR | SoftMax | $[0.17, 1]$ |
| WP | Tanh | $[-0.64, 1]$ |
| powers_calc | Tanh | $[-1, 0.9]$ |
| PradTot_RT | Tanh | $[-1, 1]$ |
| KAPPA | MinMax | $[0, 0.99]$ |
| DELTA | MinMax | $[0, 1]$ |
| Q95 | Tanh | $[-0.8, 0.9]$ |
| H98y2calc | Tanh | $[-0.6, 0.8]$ |
| P_LH_RT | ZScore | $[-2, 3.65]$ |
| LI | Tanh | $[-0.96, 0.98]$ |
| VOL | SoftMax | $[0.04, 0.91]$ |
| AREA | SoftMax | $[0.05, 0.86]$ |
| BETAN | Tanh | $[-0.64, 1]$ |
| Te_rho | Tanh | $[-1, 0.9]$ |
| Ne_rho | Tanh | $[-1, 1]$ |
| SSX | Tanh | $[-1, 1]$ |

Table 4.2: Summary of the features used in this work, their normalization and allowed range of variation after the normalization.

Let it be noted that even after the normalization some features were still presenting some spikes in the extremal regions, which would bias the model. These are due to the presence of values coming from extreme experimental configurations. When plotting the histogram of the feature, the number of bins is automatically chosen based on the number of counts and the range of normalization. A relevant number of really low/high (not significant) values can then be put all in the same bin, which will be seen as a spike at one of the extremes of the range. By only plotting these extreme regions it can be seen that the contribution is actually negligible.
Therefore the choice has been made, in some cases, to further cut the distribution range as to exclude these spikes; such information is also reported in Table 4.2. Note that these cut imply that `NaN` values may appear in the middle of the discharge, therefore one further step of linear interpolation is carried out, with the same modality described in the previous section.

As a last step after normalization, all the `NaN` values at the beginning and end of each shot are rigidly cut, in order to have files which contain only available values and are ready to use to construct the dataset. To achieve this, the file is scanned to individuate the feature(s) which has (have) the longest sequence of `NaN` at the beginning/end of discharge; this allows to individuate the first and last available time instant of the new file, which will have no `NaN` present.

# Chapter 5

# Neural network model

As briefly outlined in Introduction, the goal of this thesis is to develop a neural network model for the reconstruction of the electron temperature and density profiles. In this chapter, the two network architectures I developed and the results obtained through both of them will be presented in detail.
First, the inspiration and concept behind the architectures will be presented, followed by the models details in terms of neural network layers and implementing framework.
Next, results will be discussed, obtained with different combinations of regularization and optimization techniques, as well as a different number of points in the profile spatial resolution, samples in the dataset and normalizations.
Of the two developed architectures, only one was able to perform the task successfully. Nonetheless both networks will be presented in detail, since developing the working one was a direct consequence of understanding the failure of the first one.

## 5.1 Model inspiration

The main inspiration for the network architecture first came from the work of Matos et al (2020) [23]. In this article, the authors develop a neural network model for the classification of plasma confinement states (Low, Dither, High) and the detection of ELMs (Edge Localized Modes [32]). The architecture is called ConvLSTM, and it is composed of convolutional layers and LSTM layers. This combination of layers is intended to be able to capture temporal relations in the data at different time scales: longer for the LSTM layers, shorter for the convolutional ones.
For this work, since we are dealing with kinetic plasma profiles, which have the structure of a time series with space resolution, being able to capture temporal correlations is of the utmost relevance. Given the good results ob-

tained by [23], the model was constructed with the same base architecture. Nonetheless, since the goal of this work is to perform a reconstruction task, not a classification one, the structure of autoencoders seemed the most natural fit.
Drawing further inspiration from the work of Zhu *et al* [33], I construct an autoencoder with the base structure of the ConvLSTM network. In [33], the authors develop an autoencoder model with LSTM layers with the goal of reconstructing and forecasting multi-dimensional time-series data. Furthermore, they employ the latent embedded representation of the autoencoder, concatenated with additional cathegorical information, as the input to a network of fully connected layers, with which forecasting and uncertainty estimation are performed.
Combining these two sources of inspiration the model is thus shaped as a symmetric autoencoder, with the same kind of layers in reversed order both for the encoder and the decoder. The latent representation is embedded in the first LSTM layer (last layer of the encoder).
This first architecture, which will be referred to as AEConvLSTM, could not perform the task in a satisfying manner. The model was unable to reconstruct the profiles accurately and resorted to produce in output the same average result for all inputs.
This led to a complete rethinking of the network architecture, with the primary goal of simplifying the data structure, which was deemed too complex. As a result, a different network was further developed, with a simpler architecture: still an autoencoder, with only LSTM layers and a fully connected one for readout.
Results improved substantially, thus the final model developed for this work is the second one, which from now on will be simply called AE_LSTM.

## 5.2 Dataset

Concerning the dataset with which the network is trained, it is constructed starting from the normalized parquet files presented in Chapter 4.
The networks take as input a multidimensional array, which can be different from the one it is given as output. Indeed, the input contains the profile to reconstruct (whichis reminded can be the electron temperature or density) and can additionally feature some 0D variables, which can aid reconstruction by means of additional information.
The additional 0D features are chosen based on their relevance to the output signal, namely they can be divided in:

- averaged line-integrated density from the FIR;
- plasma stored energy;
- total input power minus the losses;

- LH power scaling;
- Internal inductance (proxy for plasma current);
- equilibrium parameters: H98y2 scaling factor, edge safety factor and normalized beta;
- shape parameters: elongation, triangularity, plasma volume, plasma poloidal cross section area.

It is possible to also employ an ulterior 1D profile, which is the signal coming from the Soft X-ray emission. It has 51 available channels and can be used as a proxy for the Thomson electronic temperature.

The algorithm which builds the data samples does so randomly.
A reduced subset of the parquet files available (126) is considered, in order to ensure the variability of the dataset. Indeed, with 1886 discharges available, if one makes an estimate of the number of samples that can be extracted given a time sequence length of 5000 time steps (500ms), a number of $\approx 25$ million possible samples is obtained. The risk of extracting from such a large pool of possibilities is to end up covering only a small part of the profiles variability.
This effect is exacerbated by the fact that in the dataset, the majority of the sequences that can be extracted from the discharges are either of ohmic type or with low auxiliary power, which naturally skews the dataset towards this type of sequences; moreover this inevitably reduces the reconstruction capability of the network when faced with non ohmic sequences, such as disruption avoidance ones which include a significant amount of additional heating.
Nonetheless, in doing this reduction, the overall distribution of the dataset is not changed, as can be seen in Fig. 5.1, where, as an example, the probability density function of Ch. 10 of the TS for the electron temperature is plotted, both for the whole 1886 discharges and for the reduced subset of 126. Notice that to aid visualization the distributions are smoothed with a kernel smoothing function. For each discharge in this subset of the database (to which is associated a shot number), there are a certain number of possible starting times, given the length of the time sequence needed to construct the sample.
After having built a set of all the shot numbers and associated possible starting times, a number equal to `#training + #validation + #test` pairs is randomly extracted. Each sample is then constructed by reading the parquet file of the shot from the starting time `start_time` up to `start_time + time_spread*time_base`, where `time_base` is needed to convert `time_spread` from number of time steps to the same time unit of `start_time`.
The data generation routine is implemented in Python, in particular sam-

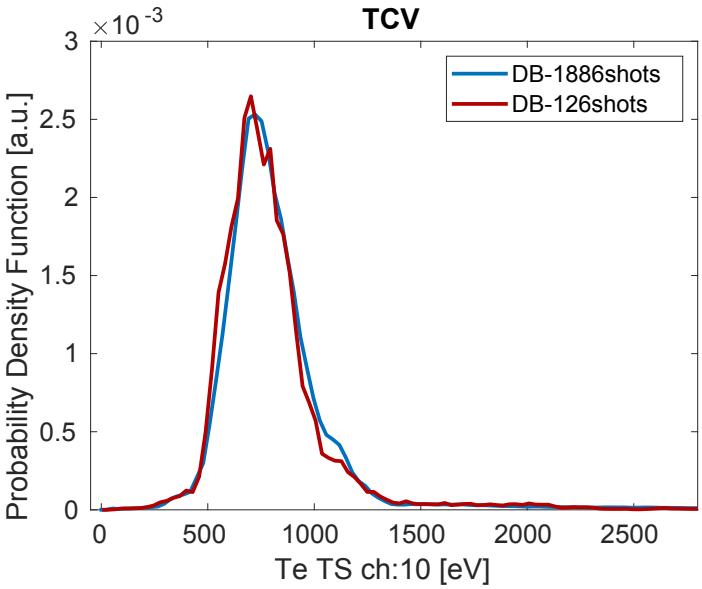


Figure 5.1: Probability density functions for channel Te_rho_z10 of TS, for 1886 discharges and for the reduced subset of 126

ples are constructed as multi dimensional numpy arrays. Nonetheless, the samples are written to a format different from the `.npy` native numpy format.

Indeed, when working with big amounts of data (as is the case in this work, with the training dataset of AEConvLSTM being $\approx$ 67GB) it is important to optimize the input pipeline as to not overload it and create bottlenecks. In order to avoid this, the TFRecord file format is adopted, which is Tensorflow's own binary storage format.

The binary format has the advantage of being lightweight: this can have significant impact on the performance of the input pipeline, and therefore on the training time of the model. Binary data takes up less space on disk and is faster to read.

Moreover, being TFRecord native of Tensorflow, it is optimized for use with the library in different ways. It is seamlessly integrated with the `tf.data.Dataset` object of Tensorflow, which allows to preprocess data in multiple ways. For example, it allows prefetching of samples, batching and shuffling, as well as mapping operations along the dataset and generating an iterator object.

All of these actions can be carried out without loading the dataset in memory, which for this work is of fundamental importance since it simply can't fit.

Thus, the numpy arrays with the data need to be converted in a format suitable to be written to a TFRecord file. This process involves two steps

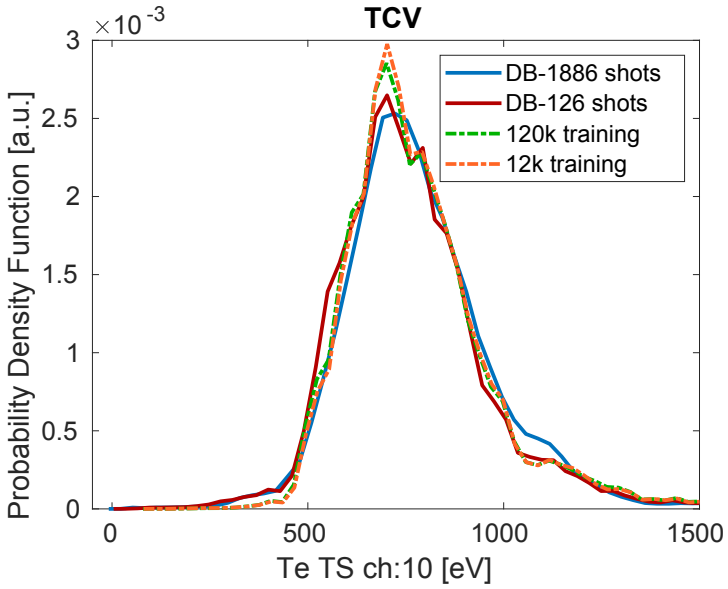


Figure 5.2: Probability density functions for channel Te_rho_z10 of TS, for 1886 discharges and for the reduced subset of 126

[34]:

- convert the numpy array into a `tf.Tensor` of type `tf.string` containing the data in a binary string format, which is suitable to work with `tf.train.Example`
- write a `tf.train.Example` message which details the structure of the sample; in this case, what are the input $X$ and the output $y$

After having properly prepared the data, each sample is simply written to a different TFRecord file.

Firstly, this allows to occupy less memory during the writing process, as one generates the multidimensional array and writes it to disk straight away. This means to avoid having multiple multidimensional arrays, each with $\sim 10^9$ elements, loaded in memory, which clearly makes the process less heavy.

Moreover, this choice leads to an ease of reading: when constructing the `tf.train.Dataset` object is much more flexible having a different file for each sample.

Lastly, this allows one to shuffle the samples by simply shuffling the names of the TFRecord files, which is a much quicker operation than shuffling elements of the dataset. Indeed, the `shuffle` method of the dataset object works by filling a buffer with buffer_size (a parameter to fix) elements, then randomly samples elements from this buffer, replacing the selected elements with new elements. In principle, to achieve perfect shuffling one should fix

buffer_size greater than or equal to the size of the dataset. Lower values of buffer_size lead to a suboptimal shuffling. [35]
Given the size of the dataset, it is clear to see that by using this method the only possible shuffling would be sub optimal. To avoid this problem, the `list_files` method of `tf.train.Dataset` is used, which creates a dataset of strings corresponding to the file names. At this point, one can employ the shuffle method on this dataset of strings, since the whole dataset can now easily fit into memory. After, by means of the `interleave` and `map` method the strings dataset becomes a tensors dataset to be used for training or evaluation.

Throughout this work, various sizes of training, validation and test sets are employed, depending on the model being developed. Nonetheless, the same proportions are maintained, so that the training set is bigger than validation and test sets. A standard 80/20 split between training and validation/test is employed.
In particular, the same graphs as in Fig. 5.1 are presented for two of the datasets that will be introduced and employed in Chapter 7, as to show that the sampling process constructing the datasets is properly done. Moreover, Fig. 5.2 shows that the implicit assumption in building a ANN model is respected: the distributions of training, validation and test set are the same as the distribution of the whole dataset.

# Chapter 6

# AEConvLSTM

In this chapter we present the details of the AEConvLSTM architecture. The model is implemented in Tensorflow 2.3.0 [36]. For further details on the Anaconda environment used for developing this work, see [37].

## 6.1 Architecture

As previously mentioned the model architecture is an autoencoder with different types of layers. In particular, a detailed graphical representation can be seen in Fig. 7.2.

The input/output array is schematically represented in Fig. 6.1. Input and output have shape, respectively:

(`time_spread // stride, conv_window, no_feat_input`)

(`time_spread // stride, conv_window, no_target_ch`)

Where:

- `time_spread` is the length in time steps of the time sequence considered to build the sample. `stride` is the number of points shifted over for subsampling of the time sequence: one point is taken every `stride` points. The operation `//` indicates the integer result of the division between the two numbers. This is the length of the sequence in input to the LSTM layer. In the figure, this is indicated as 300, which is a typical length considered throughout this work.
- `conv_window` is the size of the convolutional window, therefore of the input to the convolutional layer. It is fixed to 40 throughout this work.
- `no_feat_input` (`no_target_ch`) is the number of features (channels) in input (output). In the figure, 147 is indicated as input (67 from

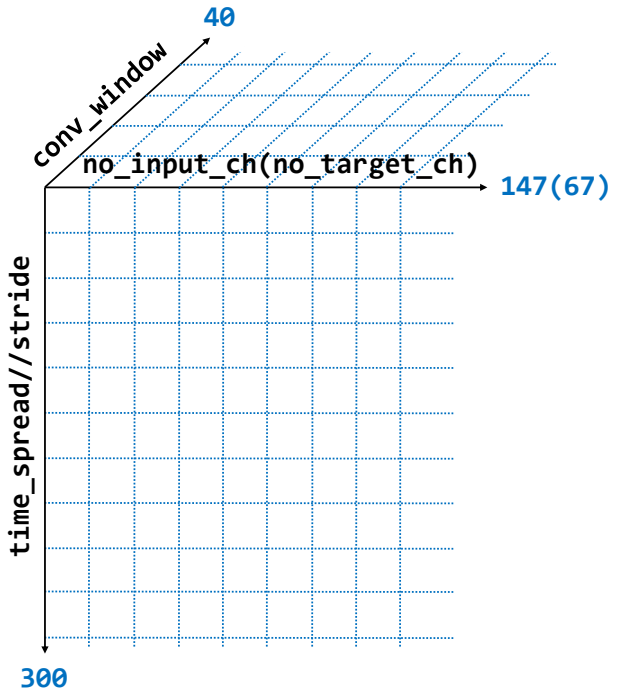


Figure 6.1: Graphical representation of the input (output) to (of) the AEConvLSTM architecture

TS temperature, 67 from TS density and 13 0D features) and 67 as output (TS temperature/density).

Referencing Fig. 6.2, starting from the left, the structure of the network is as follows:

- a convolutional layer with `no_feat_input` input channels and 32 output channels; a kernel size of 3 and a stride set to 1 so that there is no reduction in the convolutional input size, which is `conv_window`.
  The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window, no_feat_input`)
  (`batch_size, conv_window,` 32)

- another convolutional layer with 32 input channels and 64 output channels; a kernel size of 3 and a stride set to 1 so that there is no reduction in the convolutional input size, which is `conv_window`.
  The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window,` 32)
  (`batch_size, conv_window,` 64)

- a dropout layer with dropout rate a hyperparameter to be optimized. The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window,` 64)

  (`batch_size, conv_window,` 64)

- after flattening the output of the previous dropout layer, a fully connected layer with 64×`conv_window` input units and 16 outputs. The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window` × 64)

  (`batch_size,` 16)

- an LSTM layer with 32 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 16 (the output units of the previous dense layer). The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 16)

  (`batch_size, time_spread // stride,` 32)

- an LSTM layer with 32 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 32. The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 32)

  (`batch_size, time_spread // stride,` 32)

- a fully connected layer with 32 input units and 64×`conv_window` outputs. The array in input and output to this layer have shape, respectively:

  (`batch_size,` 32)

  (`batch_size, conv_window` × 64)

- a reshaping layer. The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window` × 64)

  (`batch_size, conv_window,` 64)

- a dropout layer with dropout rate a hyperparameter to be tuned. The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window,` 64)

  (`batch_size, conv_window,` 64)

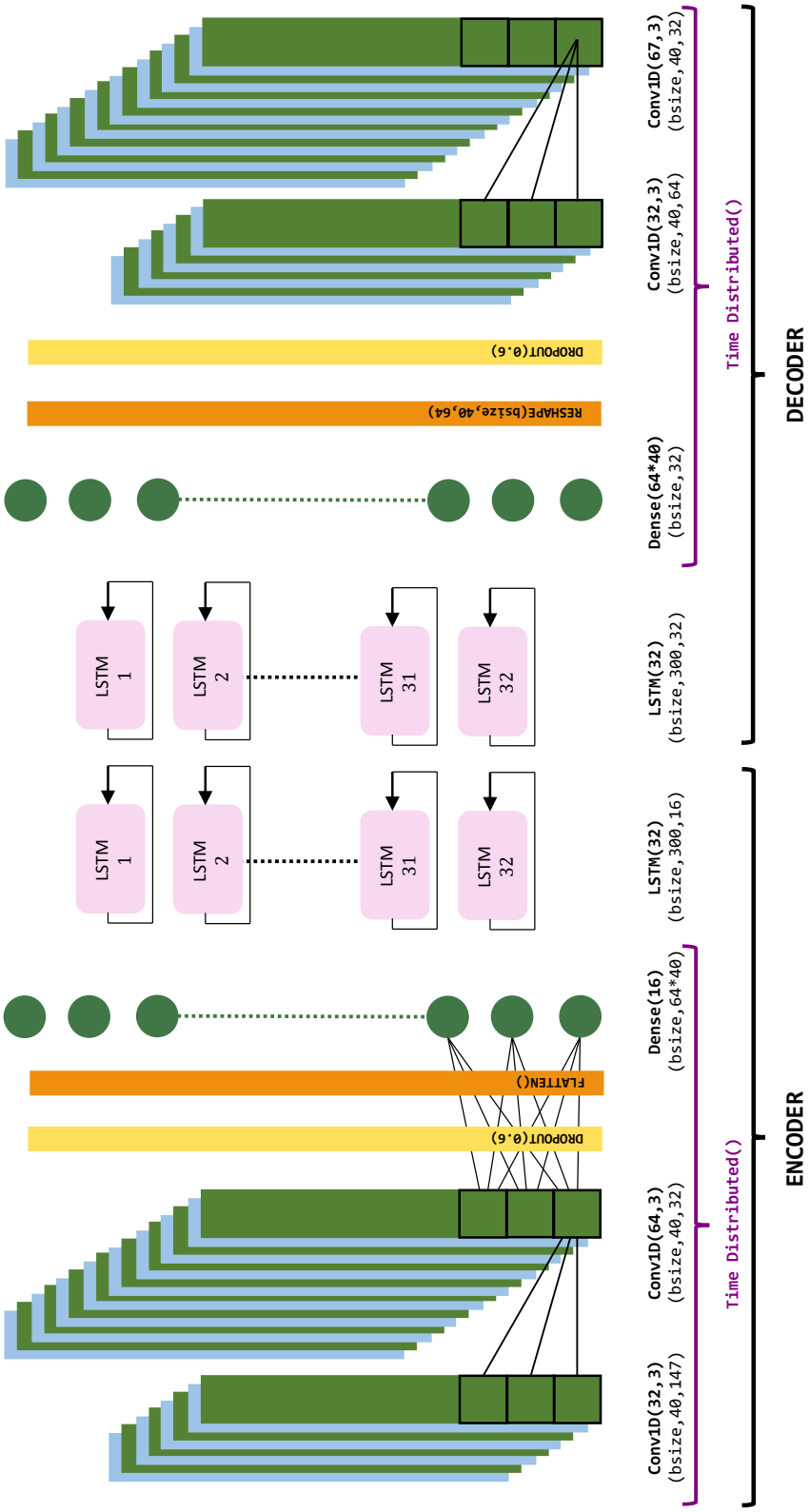


Figure 6.2: Graphical representation of the AE_ConvLSTM architecture

- a convolutional layer with 64 input channels and 32 output channels; a kernel size of 3 and a stride set to 1 so that there is no reduction in the convolutional input size, which is `conv_window`.
  The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window,` 64)
  (`batch_size, conv_window,` 32)

- a convolutional layer with 32 input channels and `no_target_ch` output channels; a kernel size of 3 and a stride set to 1 so that there is no reduction in the convolutional input size, which is `conv_window`.
  The array in input and output to this layer have shape, respectively:

  (`batch_size, conv_window,` 32)
  (`batch_size, conv_window, no_target_ch`)

Notice how the length of the sequence (`time_spread // stride`) appears as input only to the LSTM layers. One may wonder how it can feature in the middle of the network without being used in the previous layers. The answer lies in the `TimeDistributed` layer of Tensorflow.
Consider the input sample with shape:

(`batch_size, time_spread // stride, conv_window, no_feat_input`)

since until the LSTM layers the time sequence is not modeled, one needs a way to pass the input without the second dimension to the convolutional and dense layers. `TimeDistributed` allows to do exactly this.
The input flows in the network for each time step in the time sequence, up until the first LSTM layer. This means that the layers are applied to every time step `time spread // stride`. Once all these single timestep inputs have been processed, the whole time sequence output is fed to the LSTM layers. The same operation is carried out until the end of the network.
Notice that no update of the weights is performed between the processing of different time slices of the same sample.

Reproducibility of the results is ensured by fixing the random seed. Both the global seed and the operation seed are fixed, for each operation that involves randomness.
More in detail, the global seed is set as:

```
tf.random.set_seed(2022461)
```

Moreover, the weights matrix of convolutional and dense layers and the matrix used for linear trasnformations of the input in the LSTM is initialized

through:

```
tf.keras.initializers.GlorotUniform(2022461)
```

while the matrix used for the linear transformation of the recurrent state in the LSTM layer as:

```
tf.keras.initializers.Orthogonal(2022461)
```

Finally, a seed for the dropout operation is also set, equal to 12345.

## 6.2 Training and optimization

Training is performed on a machine with 12 6-core CPUs, model Intel® core™ i7-8700 CPU @ 3.20GHz, and a 32GB memory.
As a first step, a gridsearch on the parameter space is carried out, to identify the best combination of hyperparameters and architecture structure.
Potentially, a high number of combinations is possible: considering the number of layers, if one wants to optimize the number of units for each layer, the size of the convolutional kernel, the dropout rate, the optimizers, the regularizers and the learning rate is easy to reach $\approx 10^6$ different networks.
The ideal approach to explore such a high dimensional parameter space would be to perform a random hyperparameter search to identify the correct region of space in which to focus. Subsequently, a more refined gridsearch on such restricted space of parameters should lend the more appropriate configuration.
Nonetheless, for the AEConvLSTM network, given the high dimensionality of the input and output, training for 20 epochs (which is enough to obtain a substantial decrease in the loss function) takes at least 6h.
This implies the obvious impossibility of performing extensive random/grid search.
Thus, focusing on what are considered to be the most relevant parameters we optimize:

- number of neurons of the latent space embedding $\in \{8, 16, 32\}$
- dropout rate $\in \{0.2, 0, 6\}$
- learning rate $\in \{10^{-5}, 10^{-4}, 10^{-3}\}$

Obtaining a total of 18 configurations, which are studied with Optuna[38], an automatic hyperparameter optimization software framework.
The optimizer is chosen as Adam with the tensorflow implementation standard parameters and the convolutional kernel size is fixed to 3.
The optimization procedure is performed with a training set of 4096 samples

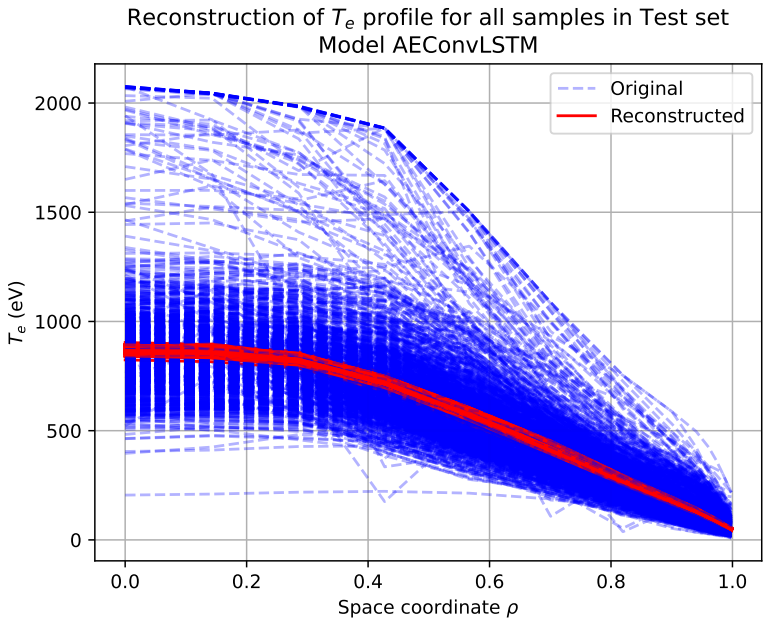


Figure 6.3: Original and reconstructed profiles in the test set, model AEConvLSTM

and a validation set of 1024 samples, running each training for 20 epochs. The employed loss function is the mean squared error.
The best configuration is the one with lowest validation loss, which corresponds to the one with 32 latent units, dropout rate equal to 0.2 and learning rate equal to $10^{-3}$.

The network with such parameters is then trained for 50 epochs, reaching a training loss of 0.0554 and a validation loss of 0.057.

## 6.3 Evaluation

When performing evaluation on the test set, the loss reaches a value of 0.062, which is in good agreement with the training loss, indicating no relevant presence of overfitting.
Nonetheless, the simple numeric value of the loss is not revealing of the actual performance of the model. If one plots, for all samples in the test set, the original profiles and the reconstructed ones in the same figure (Fig. 6.3) it is easy to recognize that the model is actually always producing the same output for all inputs.
This behaviour can be explained by noticing that the output is the *average* profile seen by the network, in the sense that for each channel the network

is giving as output approximately the average values of the electron temperature in the dataset.
The network is not capable of learning the reconstruction task, so it resolves to giving the average as the answer, which is a safer choice than making a random guess. Furthermore, it is likely that the variability in the distribution of each feature gets lost in the convolutional layers because of the model overparametrization with respect to the training data.
Even after explaining the origin of this incorrect output, the question still remains as to why the network fails in completing the reconstruction task.
This can be answered by focusing on the number of samples which are employed in the training set, which is 4096. Such number is too low to ensure a proper training of this kind of model. Indeed, the output samples are arrays with a notable number of elements ($\approx 8 \cdot 10^5$), meaning that the network needs to learn a lot of information from few examples.
Furthermore, up scaling the size of the dataset is not feasible: even if the samples are few, the dataset has considerable dimensions. Considering a training set of size 4096, and a validation and test set both of size 1024, around 67 GB of disk space are needed. Clearly, this also results in a long training time, which is $\approx$ 10h for 40 epochs.
To conclude, the downfall of this network architecture is linked to the too high complexity of the samples, which requires a much larger dataset.
It is expected that by reducing the complexity of the samples and making the dataset larger, one should be able to obtain acceptable results.

# Chapter 7

# AE_LSTM

In this section the details of the AE_LSTM architecture are presented.
The model is implemented in Tensorflow 2.3.0 [36]. For further details on the Anaconda environment used for developing this work, see [37].

## 7.1 Architecture

As previously mentioned the model architecture is that of an autoencoder with LSTM layers. In particular, a detailed graphical representation can be seen in Fig. 7.2.
Given the difficulty encountered in solving the task in Chapter 6, for this architecture, it is decided to develop two distinct models. The first one will be dedicated to reconstructing only the pedestal region, which is the outer region of the plasma ($\approx \rho > 0.8$): monitoring the pedestal is of great importance in tokamak operation. The second model will focus, on the other hand, on reconstructing the whole profile.
Thus, throughout this chapter, for each setup of the network two models will be developed: one for pedestal and one for whole profile reconstruction.
The input/output array is schematically represented in Fig. 7.1. Input and output have shape, respectively:

(`time_spread // stride, no_feat_input`)
(`time_spread // stride, no_target_ch`)

Where:

- `time_spread` is the length in time steps of the time sequence considered to build the sample. `stride` is the number of points shifted over for subsampling of the time sequence: take one point every `stride` points. The operation `//` indicates the integer result of the division between

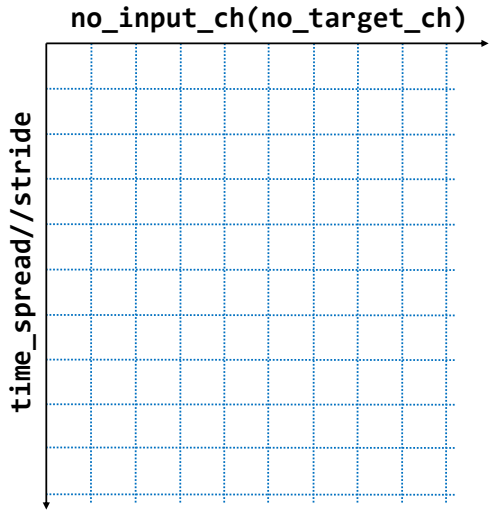


Figure 7.1: Graphical representation of the input (output) to (of) the AE_LSTM architecture.

the two numbers. This is the length of the sequence in input to the LSTM layer.

- `no_feat_input` (`no_target_ch`) is the number of features (channels) in input (output).

Referencing Fig. 7.2, starting from the left, the structure of the network is as follows:

- an LSTM layer with 32 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 42 (for the pedestal reconstruction) or 63 (for the whole profile reconstruction). The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 42/63)
  (`batch_size, time_spread // stride,` 32)

- an LSTM layer with 16 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 32 (the output units of the previous LSTM layer).
  The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 32)
  (`batch_size, time_spread // stride,` 16)

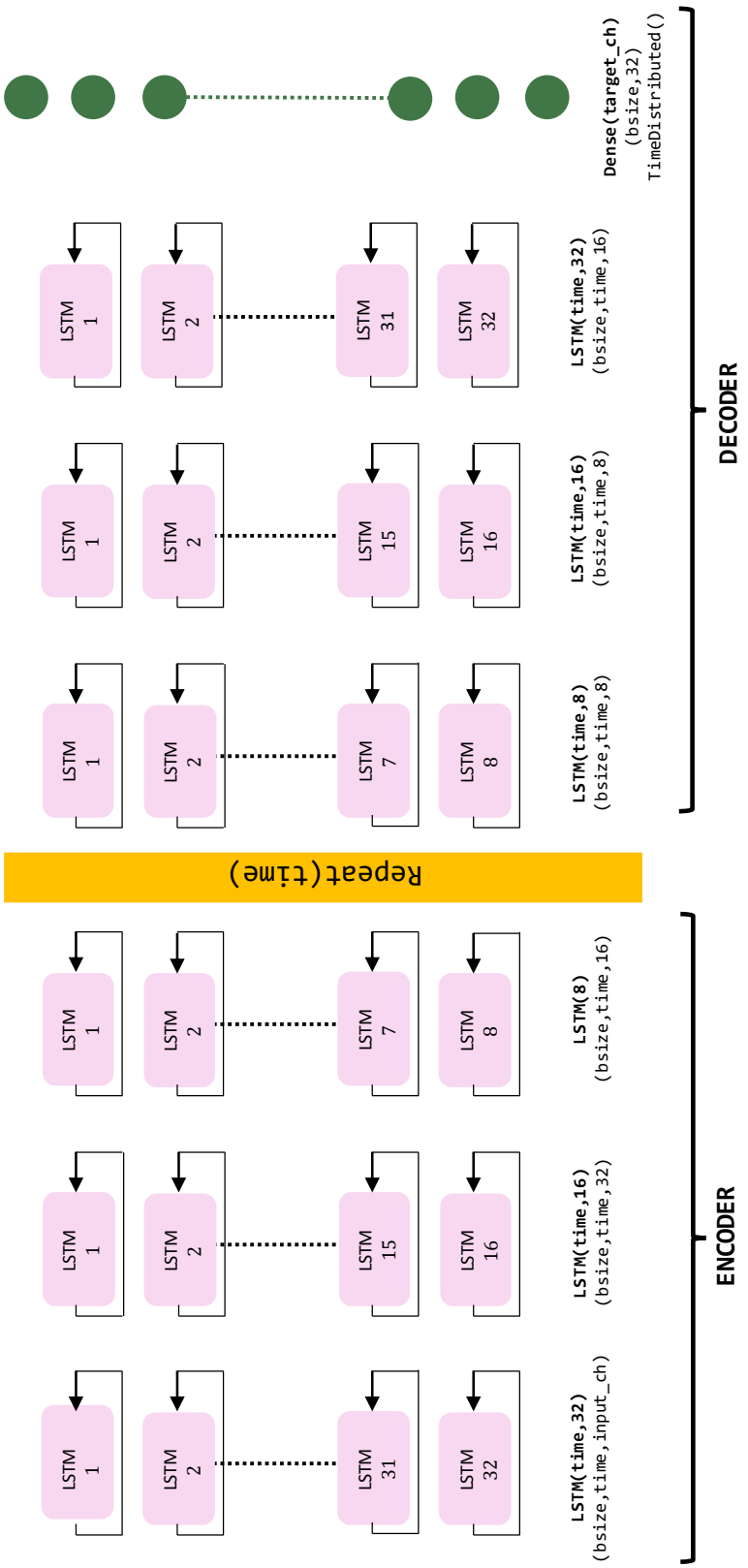


Figure 7.2: Graphical representation of the AE_LSTM architecture.

- an LSTM layer with 8 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 16 (the output units of the previous LSTM layer).
  This is the embedding layer, the one which learns a lower dimensional representation of the data; the last layer of the encoder.
  The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 16)
  (`batch_size,` 8)

- A repeat layer, which takes care of repeating the previous output `time_spread` times so to retrieve the time sequence dimension. The array in input and output to this layer have shape, respectively:

  (`batch_size,` 8)
  (`batch_size, time_spread // stride,` 8)

- an LSTM layer with 8 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 8 (the output units of the previous LSTM layer).
  This is the first layer of the decoder.
  The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 8)
  (`batch_size, time_spread // stride,` 8)

- an LSTM layer with 16 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 8 (the output units of the previous LSTM layer).
  The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 8)
  (`batch_size, time_spread // stride,` 16)

- an LSTM layer with 32 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 16 (the output units of the previous LSTM layer).
  The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 16)
  (`batch_size, time_spread // stride,` 32)

- a fully connected layer with 42/63 units that takes as input a sequence of length `time_spread // stride` and a number of features equal to 32 (the output units of the previous LSTM layer).
  The array in input and output to this layer have shape, respectively:

  (`batch_size, time_spread // stride,` 32)
  (`batch_size, time_spread // stride,` 42/63)

As stated at the beginning of this chapter, two models were developed based on this architecture.
The first, which aims at solving the task of pedestal reconstruction, specifically considers the last 42 channels of the TS diagnostic ($0.8444 \leq \rho \leq 1$). Indeed, the pedestal is an important region to monitor, which holds important information on the confinement state of the plasma and more generally represents the layer interfacing the plasma core to the plasma boundary and the Scrape-Off Layer (SOL).
The second model, which takes care of reconstructing the complete kinetic profile, includes all channels except for the first 4: $0.1432 \leq \rho \leq 1$. Excluding the first 4 channels is a choice made to avoid including data that in the considered dataset was sometimes coming from saturated channels.
Reproducibility of the results is ensured by fixing the random seed. Both the global seed and the operation seed are set, for each operation that involves randomness.
More in detail, the global seed is set as:

```
tf.random.set_seed(2022461)
```

Moreover, the weights matrix of dense layers and the matrix used for linear transformations of the input in the LSTM is initialized through:

```
tf.keras.initializers.GlorotUniform(2022461)
```

while the matrix used for the linear transformation of the recurrent state in the LSTM layer as:

```
tf.keras.initializers.HeUniform(438)
```

Finally, a seed for the dropout operation is also set, equal to 12345.

## 7.2 Training and optimization

Starting from the base structure of the network as described in the previous paragraph, several models can be developed, depending on the number of layers, LSTM units, length of the sequence, region of reconstruction and normalization technique.

For this reason, at first different combinations of these variables on datasets of reduced size (namely 4096 samples in the training set and 1024 for validation and test sets) are tested.

| Model | Sequence Length | Stride | Time resolution | Channels | Normalization |
|---|---|---|---|---|---|
| 1 | 5000 | 100 | 10ms | 5-67 | MinMax |
| 2 | 5000 | 50 | 5ms | 5-67 | MinMax |
| 3 | 5000 | 20 | 2ms | 5-67 | MinMax |
| 4 | 5000 | 100 | 10ms | 5-67 | Tanh |
| 5 | 5000 | 50 | 5ms | 5-67 | Tanh |
| 6 | 5000 | 20 | 2ms | 5-67 | Tanh |
| 7 | 5000 | 100 | 10ms | 26-67 | MinMax |
| 8 | 5000 | 50 | 5ms | 26-67 | MinMax |
| 9 | 5000 | 20 | 2ms | 26-67 | MinMax |
| 10 | 5000 | 100 | 10ms | 26-67 | Tanh |
| 11 | 5000 | 50 | 5ms | 26-67 | Tanh |
| 12 | 5000 | 20 | 2ms | 26-67 | Tanh |

Table 7.1: Trial models for the AE_LSTM.

As can be concluded from Table 7.1, in which are listed all trials, a fixed sequence length of 5000 time slices is considered, which corresponds to 500ms. This is due to the fact that such length appropriately spans the profile evolution, catching the variations over relatively long time scales in the evolution of the discharge. On the other hand, using a shorter sequence, for example 300ms, leads to the risk of not covering major variations in the profile evolution. Even if in principle 300ms is a time scale long enough to consider the profile evolution, the dataset constructed with such choice of length for the time sequence results poor in displaying big changes in the profile, such as those linked to L-H transitions.
Moreover, different strides are considered, as to test how the network performs when seeing the profile evolution at different time resolutions. Different models are developed based on the spatial position of the channels: one pertaining the pedestal (channels 26-67), one pertaining the whole profile (channels 5-67) with the exclusion of the first 4 channels for reasons discussed in Section 7.1.
Lastly, the network performance is tested when trained with samples normalized with different techniques, namely a MinMax normalization in the range $[0, 1]$ or a Tanh normalization (more details in Section 4.3).
All models have been trained for 30 epochs using the MSE as loss function, with the same hyperparameters, namely:

- Dropout set to a rate of 0.4, both for the recurrent and linear connec-

tions;

- Regularizer set to L1_L2 with coefficient of regularization $\alpha = 10^{-8}$, both for recurrent and linear kernels;
- Adam optimizer with the Tensorflow implementation default settings;
- Adaptive learning rate $lr$, set to a starting value of $10^{-3}$ which is fixed for the first 5 epochs and then decays according to:

$$lr = lr \cdot e^{-0.1}$$

The results obtained by the models in terms of training, validation and test losses are reported in Table 7.2, as well as the training duration in seconds. Notice that in order to obtain the total time needed to converge to a functioning model, one has to add the time for the loading of the dataset to the training time. Such time varies depending on the number of samples in the dataset and the size of the samples themselves (different time resolution and space resolution). Therefore a final column with the dataset loading time is added for improved perspective of the whole process.

| **Model** | **Training loss** | **Validation loss** | **Test loss** | **Training time** | **Loading time** |
|---|---|---|---|---|---|
| 1 | 0.0027 | 0.0029 | 0.0028 | 18min 2s | 16s |
| 2 | 0.0032 | 0.0033 | 0.0032 | 34min 27s | 16s |
| 3 | 0.0032 | 0.0032 | 0.0032 | 1h 46min | 25s |
| 4 | 0.031 | 0.025 | 0.025 | 17min 48s | 12s |
| 5 | 0.030 | 0.026 | 0.027 | 34min 54s | 18s |
| 6 | 0.043 | 0.041 | 0.042 | 1h 37min | 23s |
| 7 | 0.0027 | 0.0022 | 0.0021 | 18min 0s | 14s |
| 8 | 0.0030 | 0.0024 | 0.0024 | 34min 29s | 16s |
| 9 | 0.0037 | 0.0037 | 0.0037 | 1h 35min | 25s |
| 10 | 0.028 | 0.024 | 0.025 | 17min 51s | 14s |
| 11 | 0.029 | 0.027 | 0.028 | 34min 48s | 15s |
| 12 | 0.040 | 0.046 | 0.047 | 1h 35min | 25s |

Table 7.2: Training, validation and test loss on normalized data, training time and dataset loading time for the models of Table 7.1.

## 7.3 Evaluation

First of all, for all models, no relevant overfitting is encountered: validation and test losses present values close to the training one, in some cases even lower (Table 7.2).
The training time increases with the samples size (when the stride reduces),

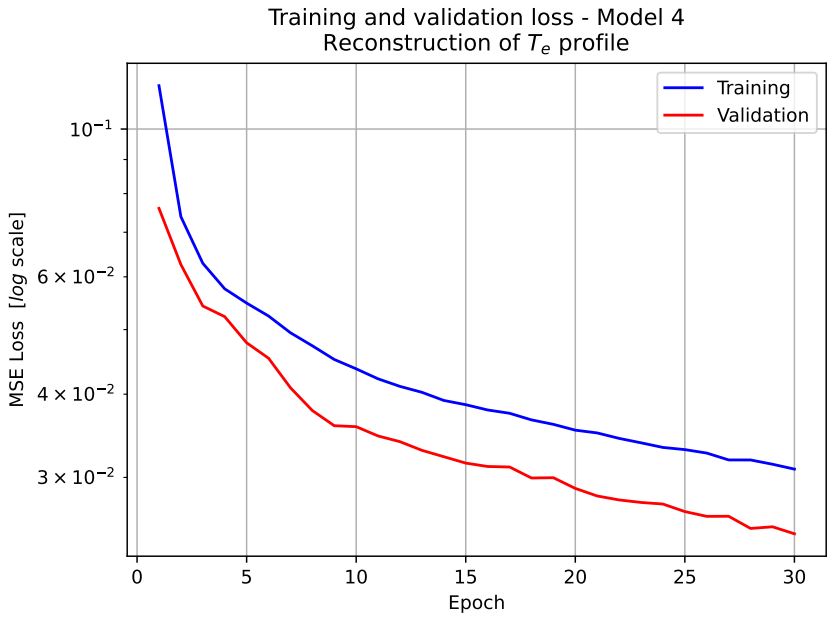


Figure 7.3: Training and validation loss, model 4. Note the logarithmic scale on the $y$ axis.

as it is expected.
Fixing one kind of normalization, the models with the lowest validation loss can be deduced from Table 7.2 and are:

- For the whole profile:
  1. for the MinMax normalization model 1
  2. for the Tanh normalization model 4
- For the pedestal:
  1. for the MinMax normalization model 7
  2. for the Tanh normalization model 10

Following, in Fig. 7.3 and Fig. 7.4 the graphs of the training and validation loss over the training epochs for models 4 and 10 are presented.

In all cases, the model performing better is the one with a stride value of 100. At first, this may seem strange: other models have a lower value of stride, which means the temporal resolution is higher and thus that the network is receiving more information about the profiles evolution.
Nonetheless, the relevant dynamic must happen on a timescale which is higher than 2ms (stride of 20) and 5ms (stride of 50). In the light of this, the models with a shorter stride end up having redundant information in time which simply makes the training heavier, requiring the network to

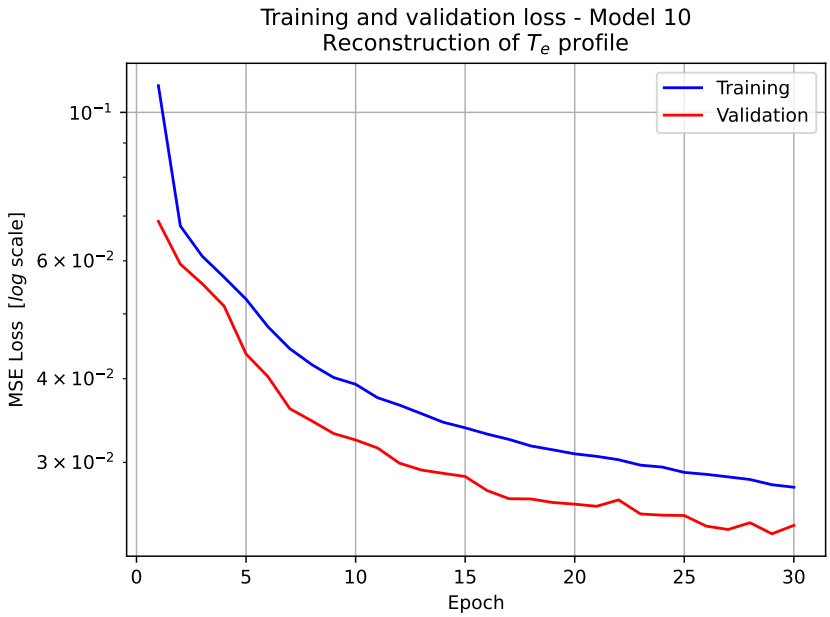


Figure 7.4: Training and validation loss, model 10. Note the logarithmic scale on the $y$ axis.

learn more information, which is not relevant to the overall dynamic. This is consistent with the acquisition frequency of the diagnostics, that is 60Hz, apart from the resampling performed to re-align the Thomson to higher bandwidth diagnostics.

| **Model** | **Test MSE** | **Test sqrt (MSE)** |
|---|---|---|
| 1 | 17151.43 | 130.96 |
| 2 | 17479.75 | 132.21 |
| 3 | 16189.52 | 127.24 |
| 4 | 4666.25 | 68.31 |
| 5 | 4368.15 | 66.09 |
| 6 | 6167.09 | 78.53 |
| 7 | 704.54 | 26.54 |
| 8 | 773.98 | 27.82 |
| 9 | 1117.09 | 33.42 |
| 10 | 587.11 | 24.23 |
| 11 | 674.78 | 25.98 |
| 12 | 1029.73 | 32.01 |

Table 7.3: MSE of de-normalized test data, root square of MSE of de-normalized test data for the models of Table 7.1.

To identify the best kind of normalization it is necessary to compute the

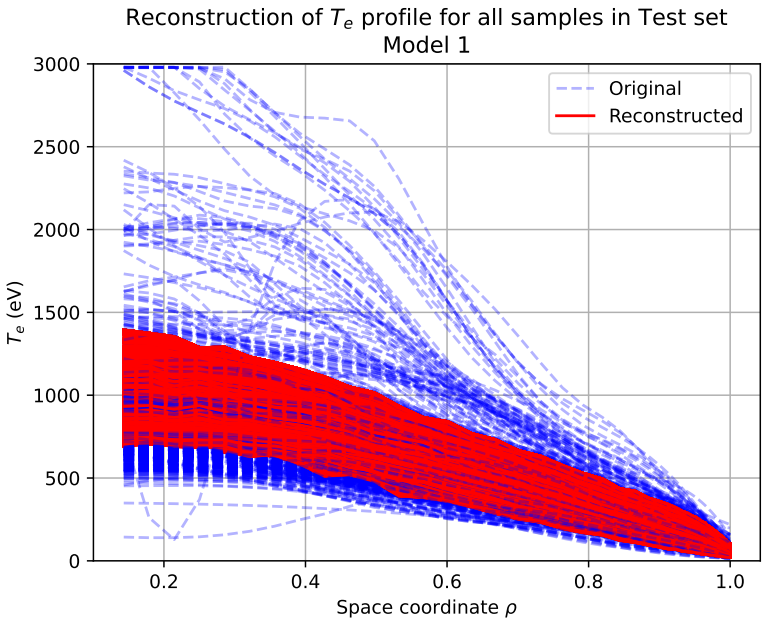


Figure 7.5: Original and reconstructed profiles in the test set, model 1.

MSE between the original and reconstructed profile, after having reversed the normalization process. In Table 7.3 are report the results of such process, also reporting the value for the square root of the MSE.

The value of the MSE square root is to be intended as the average distance (in euclidean measure) between the reconstructed and original profile.

It is clear by looking at such results that the models trained with the Tanh normalized samples consistently perform better than those with MinMax normalization. This is *a posteriori* confirmation of the work carried out in Section 4.3, where the normalization technique was tailored on the feature. Thus, it is concluded that an informed choice in normalization (as it was the case with Tanh for TS electron temperature) leads to better results when confronted with a standard MinMax approach.

The difference in performance is most clearly seen for the model reconstructing the whole profile: the MSE is one order of magnitude smaller for the Tanh normalized model (model 4), achieving an average distance of 68.31eV opposed to the 130.96eV of the MinMax model (model 1).

For what concerns pedestal reconstruction, the difference in performance is not that wide, with the Tanh normalized model (model 10), achieving an average distance of 24.23eV opposed to the 26.54eV of the MinMax model (model 7).

Computing the incidence rate of the average distances in Table 7.3 on the test set range, which changes according to the model, it is obtained that model 1 and 4 are not performing as different as it would appear by looking

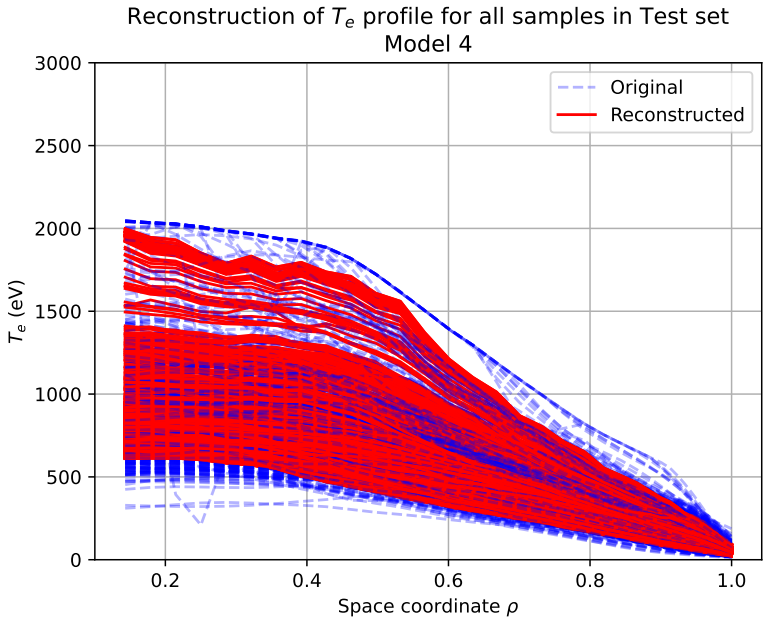


Figure 7.6: Original and reconstructed profiles in the test set, model 4.

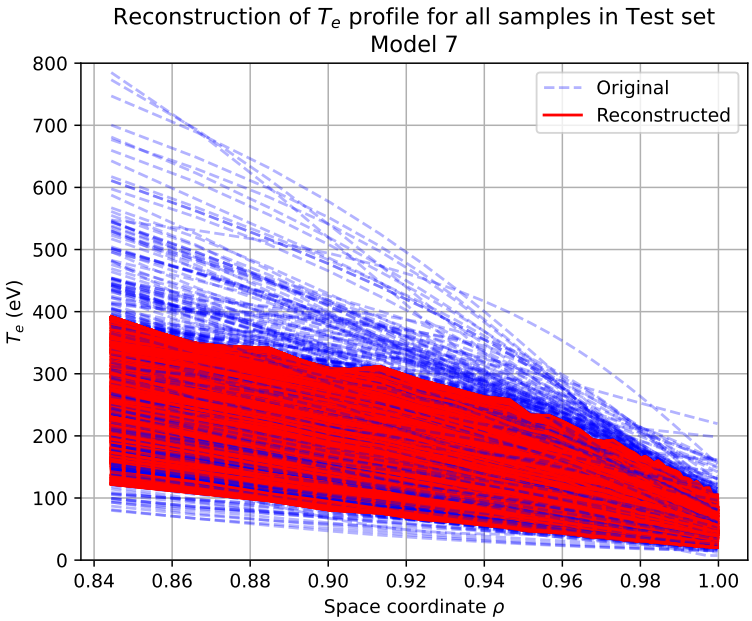


Figure 7.7: Original and reconstructed profiles in the test set, model 7.

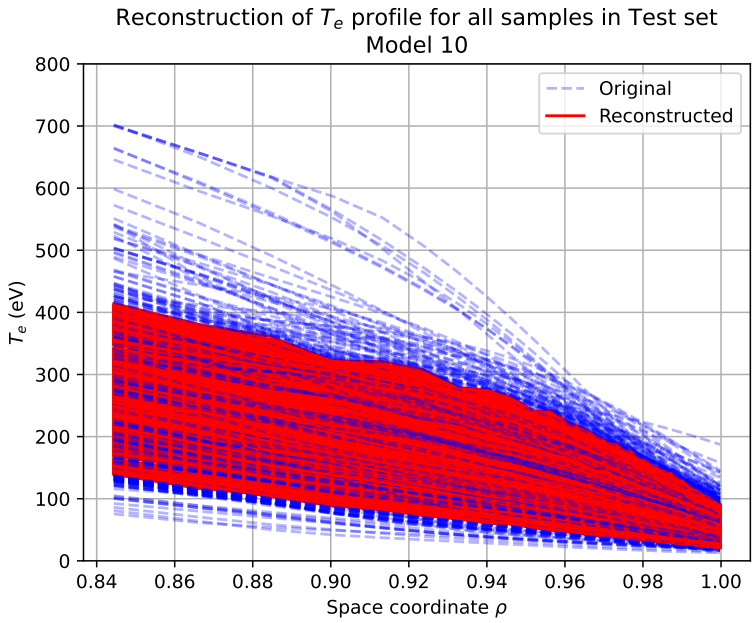


Figure 7.8: Original and reconstructed profiles in the test set, model 10.

at Table 7.3. Indeed, given the fact that the dataset are constructed by a random sampling procedure, different sets can have different ranges of values included. For example, model 1 ranges in $\approx [0, 3000]$ eV while model 4 ranges in $\approx [0, 2000]$ eV.
It is thus not meaningful to simply compare the values of MSE, while it is more meaningful to compare the incidence rate given the dataset range.
Nonetheless, with an incidence rate of 3.4% for model 4 and one of 4.5% for model 1, the Tanh normalized model is still the best performing one.
This conclusion becomes clearer when looking at Fig. 7.5, Fig. 7.6, Fig. 7.7, Fig. 7.8, in which all original and reconstructed profiles in the test set for models 1,4,7,10 are shown.
The time slice is fixed (namely number 20), which is not relevant. Indeed, since each sample has a different starting time, the index of the time slice can be linked to different stages of the discharge for each sample. Nonetheless, since the test MSE does not display specific trends with respect to the time slices indices, one can safely fix a single random time slice for visualization purposes, without losing generalization.
By looking at Fig. 7.5 and Fig. 7.6 it can be seen that the reconstruction of model 4 is indeed better, since it covers the whole range of temperature values. It can also be noticed that both models 1 and 4 struggle in reconstructing the more steeper pedestals.
Moreover, the regularizing effect of the network can be appreciated: whilst some of the original profiles present downward spikes (for example note the

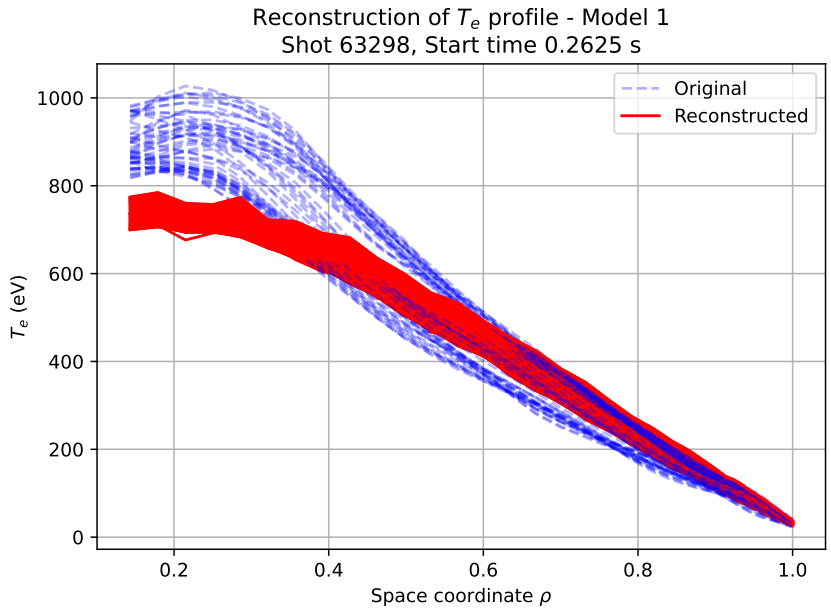


Figure 7.9: Original and reconstructed profile for shot 63298, start time 0.2625s, model 1.

one in Fig. 7.6 at around $0.2 < \rho < 0.3$ and $T_e \approx 250$eV), the reconstructed ones always present the trend that one expects for the temperature profile in a tokamak. By looking at Fig. 7.5 and Fig. 7.6 it can also be noted that for higher values of temperature $T_e > 1500$eV, in the region $0.1 < \rho < 0.6$, the network struggles (model 4, note the *broken* lines in the reconstructed profiles) or fails (model 1) at reconstructing the profiles. This could happen because the model sees less samples at high temperature values and a good portion of these samples presents irregularities which are non physical (downward spikes, saturation). Indeed, the distribution of electron temperature for the inner most channels peaks at $\approx$ 800eV, which is also where most samples in the figures peak.
For what concerns pedestal reconstruction, looking at Fig. 7.7 and Fig. 7.8 it can be seen that the reconstruction of models 7 and 10 is actually comparable in performance. They both fail at properly reconstructing the profiles with higher temperature values, which are the ones with steeper pedestals. Finally, some examples of reconstruction for specific samples in the test set are presented. For these, the graphs display a single sample time evolution.

In Fig. 7.9 we can see the best reconstructed profile in the test set of Model 1, *i.e.* the one with lowest MSE (0.00041). The model can properly reconstruct the pedestal region, but fails when it starts approaching the inner region, where temperature values are higher. This is in accordance with Fig. 7.5, where the model fails at reconstructing higher temperature values.

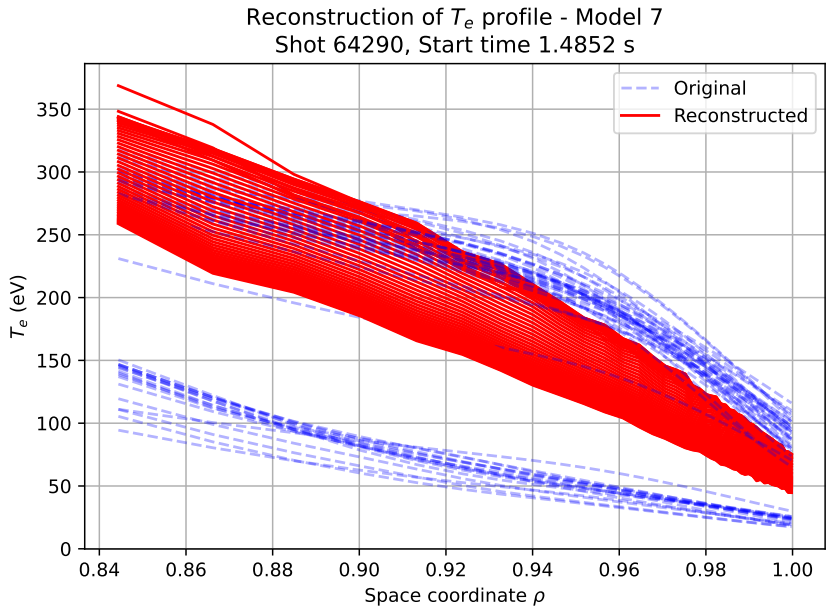


Figure 7.10: Original and reconstructed profile for shot 64290, start time 1.4852s, model 7.

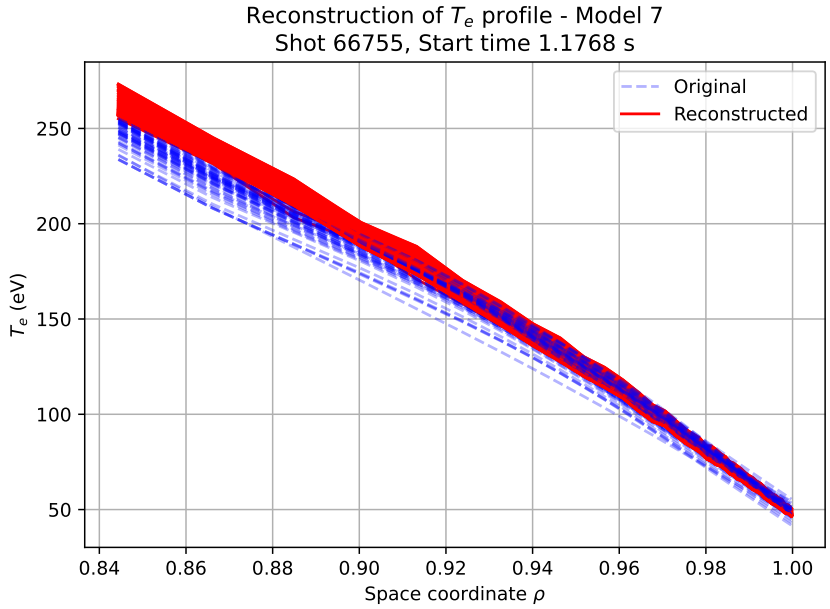


Figure 7.11: Original and reconstructed profile for shot 66755, start time 1.1768s, model 7.

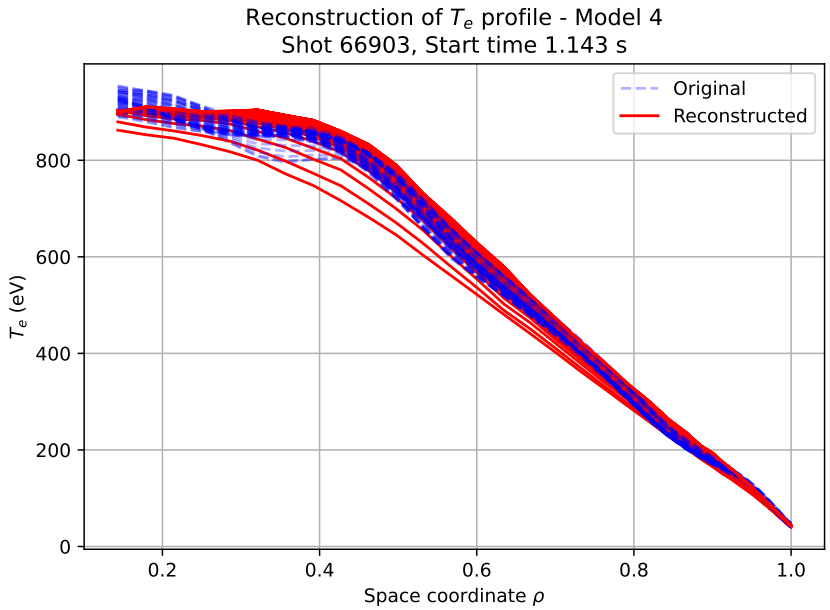


Figure 7.12: Original and reconstructed profile for shot 66903, start time 1.143s, model 4.

On the other hand in Fig. 7.10 it can be seen how model 7 struggles in reconstructing the pedestal region, especially when there's a transition from L to H mode and the appearance of a pedestal, which is what is happening in this sample (MSE= 0.015, the highest in test set). The same model, in absence of a pedestal, can accurately reconstruct the profile, as can be seen in Fig. 7.11 which presents the best reconstructed sample across the test set of Model 7.

Moreover, in Fig. 7.12 a well reconstructed profile from the test set of Model 4 is presented(MSE = 0.0027). This profile displays temperature values which are approximately around the average of the overall dataset for all channels. The type of discharge associated to this electron temperature profile is by far the most present in the dataset and consequently is the best reconstructed.

## 7.4 Final models

The previous subsection presented the results obtained when considering 12 different models, trained on a restricted dataset of 4096 training samples and 1024 validation and test samples.
Having established which are the best performing models, it is now conducted a more substantial training of the winning combinations.
In particular, the dataset is scaled up as follows:

- 12888 samples in training set, 3072 in validation and test set;
- 128880 samples in training set, 30720 in validation and test set;

Two different sizes are tested, to investigate how much the dataset size influences the model performance.
A complete summary of the final models can be found in Table 7.4.

| **Model** | **# Train set** | **# Val/Test set** | **Channels** |
|---|---|---|---|
| 1F | 12888 | 3072 | 5-67 |
| 2F | 128880 | 30720 | 5-67 |
| 3F | 12888 | 3072 | 26-67 |
| 4F | 128880 | 30720 | 26-67 |
| *Parameters common to all models* | | | |
| **Sequence length** | | 5000 | |
| **Stride** | | 100 | |
| **Time resolution** | | 10ms | |
| **Normalization** | | Tanh | |

Table 7.4: Final models for the AE_LSTM

Each model is trained for 40 epochs using the MSE as loss function, with the same hyperparameters, namely:

- Dropout set to a rate of 0.4, both for the recurrent and linear connections;
- Regularizer set to L1_L2 with coefficient of regularization $\alpha = 10^{-8}$, both for recurrent and linear kernels;
- Adam optimizer with the Tensorflow implementation default settings;
- Adaptive learning rate $lr$, set to a starting value of $10^{-3}$ which is fixed for the first 5 epochs and then decays according to:

$$lr = lr \cdot e^{-0.1}$$

Following, in Fig. 7.13, Fig. 7.14, Fig. 7.15 and Fig. 7.16 the graphs of the training and validation loss over the training epochs for the models are presented.

No relevant overfitting is encountered.
In Table 7.5 the results of the trainings are presented, namely the training and validation losses, as well as training time and dataset loading time. Notice how, when scaling up the dataset size, the loading time becomes the most impacting action in terms of required time.
For the models trained with 12288 samples, it is found that there's a slight

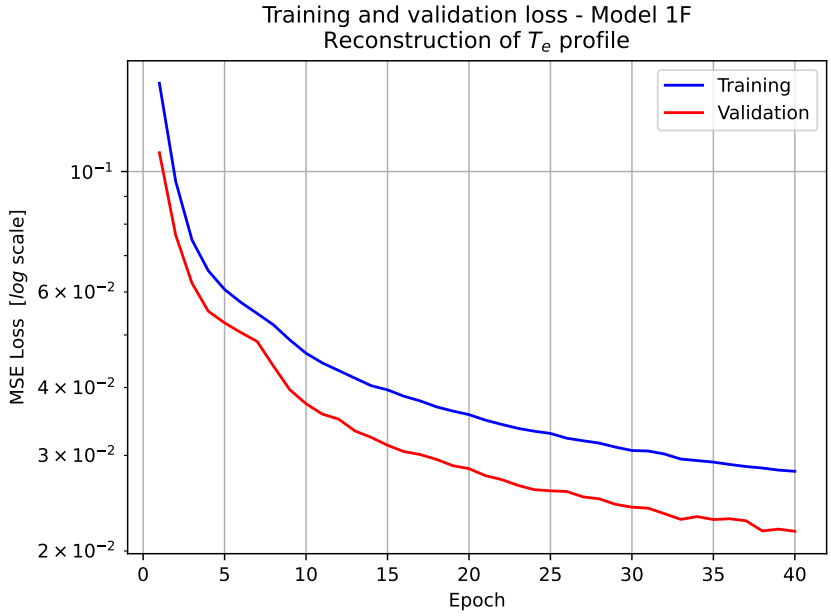


Figure 7.13: Training and validation loss, model 1F. Note the logarithmic scale on the $y$ axis.

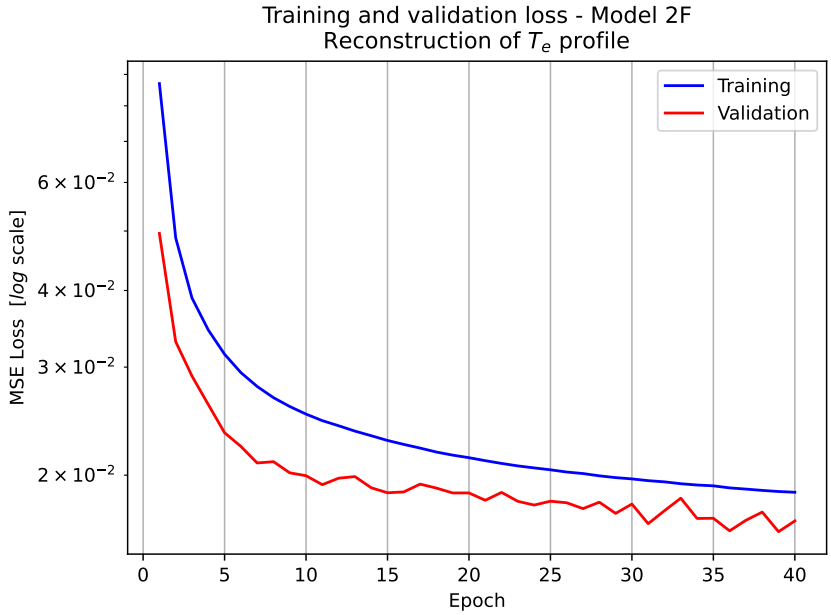


Figure 7.14: Training and validation loss, model 2F. Note the logarithmic scale on the $y$ axis.

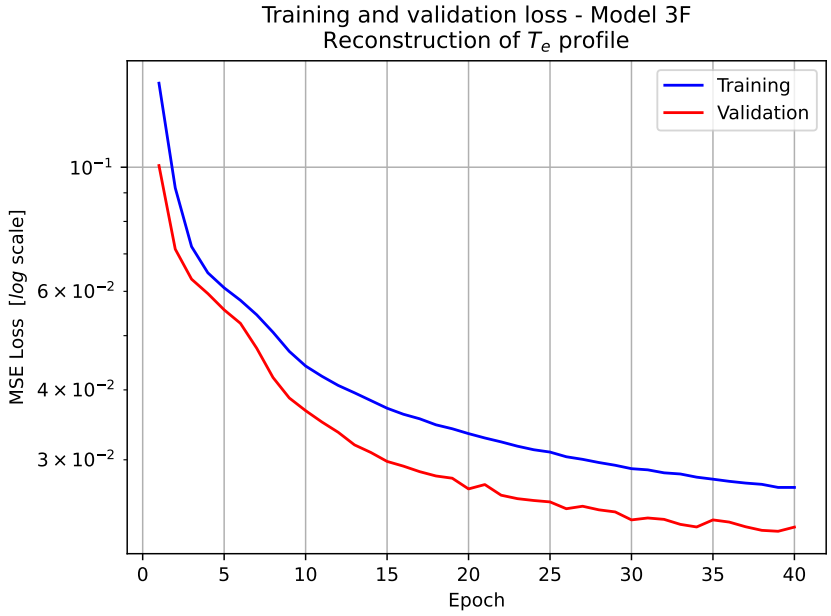


Figure 7.15: Training and validation loss, model 3F. Note the logarithmic scale on the $y$ axis.

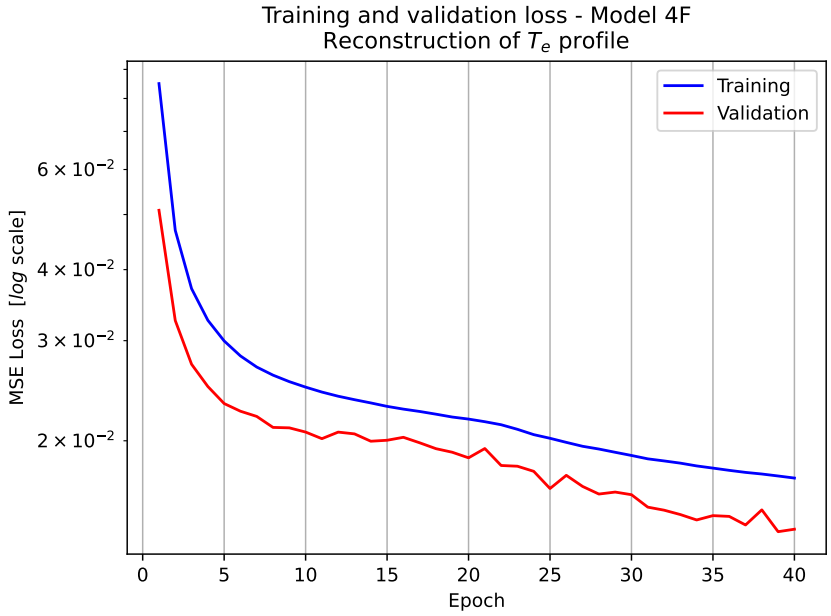


Figure 7.16: Training and validation loss, model 4F. Note the logarithmic scale on the $y$ axis.

decrease in the losses values with respect to the models of the previous section, but not significantly. On the other hand, when training with 122880 samples, it is observed a more relevant decrease in the losses values.
This behaviour is in agreement with what is expected: scaling up significantly the dataset size leads to better performances overall. The dataset is primarily made up of sequences extracted from discharges either of ohmic type or with low auxiliary power, which are the ones the network learns to reconstruct better. The loss starts to significantly decrease only when the network is showed enough samples so that it is able to see a sufficient amount of the other type of discharges.
From the losses values, it is concluded that it is necessary to increase the dataset of at least an order of magnitude to experience an appreciable increase in reconstruction performance.

| **Model** | **Training loss** | **Validation loss** | **Training time** | **Loading time** |
|---|---|---|---|---|
| 1F | 0.028 | 0.022 | 18min 45s | 6h |
| 2F | 0.018 | 0.016 | 5h 20min | 14h |
| 3F | 0.027 | 0.023 | 18min 10s | 5h |
| 4F | 0.017 | 0.014 | 4h 21min | 12h |

Table 7.5: Training and validation loss (normalized data), training time and dataset loading time for the models of Table 7.4.

This trend can also be seen in Table 7.6, where as before the MSE between the original and reconstructed profiles (de-normalized) and its square root are presented.
For model 4F, which reconstructs the pedestal, one finds an average distance between original and reconstructed of 17.74eV, which is $\approx$ 7eV less than for model 10 of the previous section. Nonetheless, model 3F displays an average distance on the test set which is practically the same as the one for model 10 (24.23 the latter, 24.33 the former).

| **Model** | **Test MSE** | **Test sqrt (MSE)** |
|---|---|---|
| 1F | 3858.69 | 62.12 |
| 2F | 2586.20 | 50.85 |
| 3F | 591.95 | 24.33 |
| 4F | 314.63 | 17.74 |

Table 7.6: MSE of de-normalized test data, root square of MSE of de-normalized test data for the models of Table 7.4.

In Fig. 7.17 andFig. 7.18 are shown all original and reconstructed profiles in the test set for models 1F and 3F.
A time slice is fixed (namely number 20), which is not relevant since each

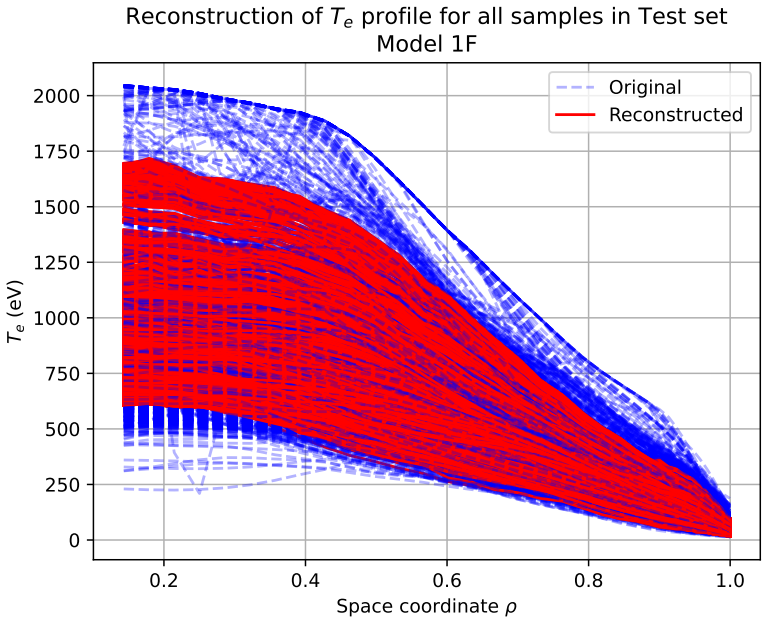


Figure 7.17: Original and reconstructed profiles in the test set, model 1F.

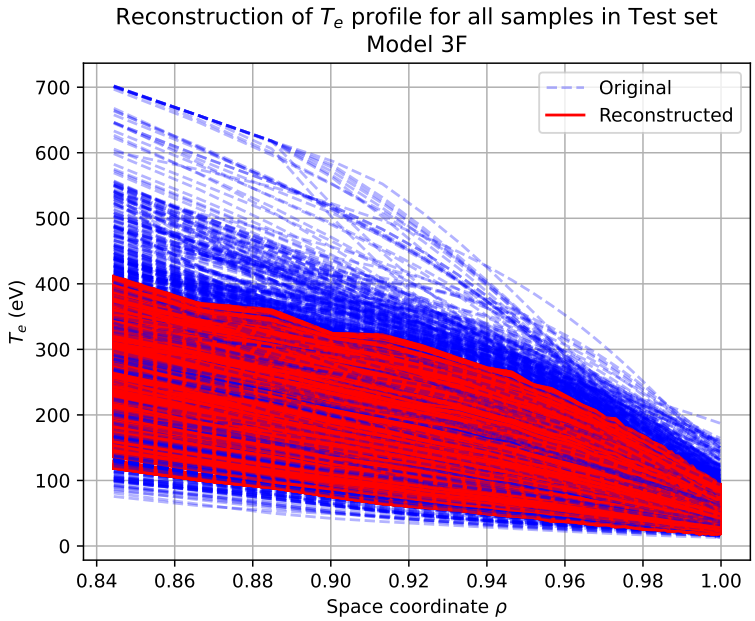


Figure 7.18: Original and reconstructed profiles in the test set, model 3F.

sample has a different starting time (it holds the same reasoning as in the previous section).
As can be seen, overall reconstruction hasn't improved much for these models trained on 12k samples. This is further confirmation that with this dataset, which is not specific to a single discharge type, it is necessary to scale up the dataset to 120k to obtain an improvement in performance. Nonetheless, the reconstruction displayed in Fig. 7.17 and Fig. 7.18 is good.
Finally, are also presented some examples of reconstruction for specific samples in the test set. For these, the graphs display a single sample time evolution.
In Fig. 7.19 the reconstruction of a test set shot is shown, which is performed by the so called Model P, a preliminary model for whole profile reconstruction, trained with MinMax normalized samples on a training dataset of size 12288. In this discharge, it can be seen that in the original profile a MHD mode is present, of which the effect can be seen in the inward bulge at $0.3 < \rho < 0.6$. This model is not capable of recognizing the local flattening of the profile due to the presence of such mode, registers it as something to regularize and produces the known shape of the electron temperature, which does not display bumps or flattenings.
Nonetheless, in Fig. 7.20 the reconstruction performed by Model 2F is presented. In this case, the network recognizes the presence of the MHD mode and properly reconstructs it. This is a clear consequence of the effect of increasing the dataset size, as well as the effect of using a tailored normalization.
Moreover, another comparison case is presented in Fig. 7.21 and Fig. 7.22. The former displays the reconstruction of model P, while tha latter the one of Model 2F. Even in this case, we can see how the reconstruction performance has clearly improved.

## 7.5 Disruption avoidance discharges

As previously stated throughout this work, the dataset is primarily made up of ohmic discharges.
Also, it was noted that in order for the network to improve its performance significantly, it is needed to scale up the dataset size as to have enough samples of the lesser represented discharges.
In the light of this, a dataset made up only of a specific discharge type, namely disruption avoidance configurations, is constructed.
The objective is to observe how the performance increases when the model is developed for a single discharge type, which in the previous dataset was only present in minority and was not properly reconstructed.
In Table 7.7 are presented the main characteristics of the dataset employed for training and evaluating such model.

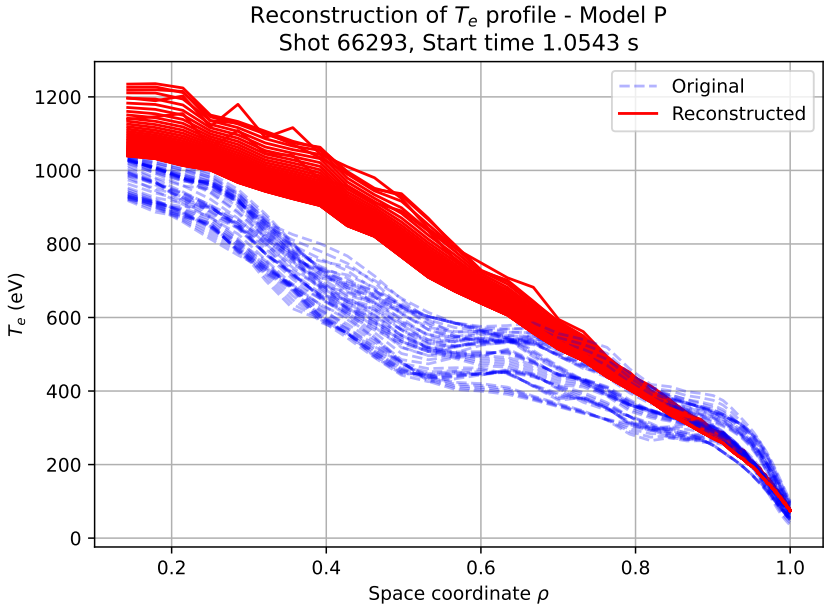


Figure 7.19: Original and reconstructed profile for shot 66293, start time 1.0543s, model P.

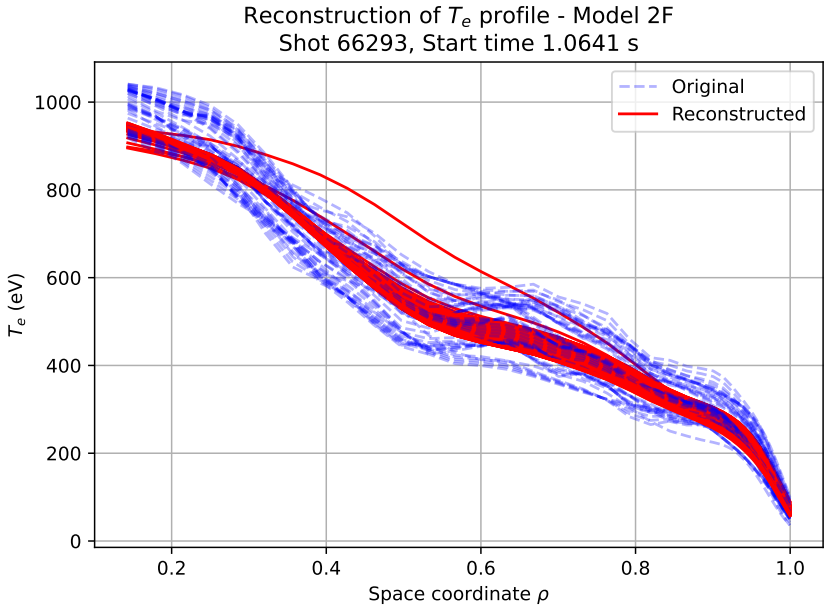


Figure 7.20: Original and reconstructed profile for shot 66293, start time 1.0641s, model 2F.

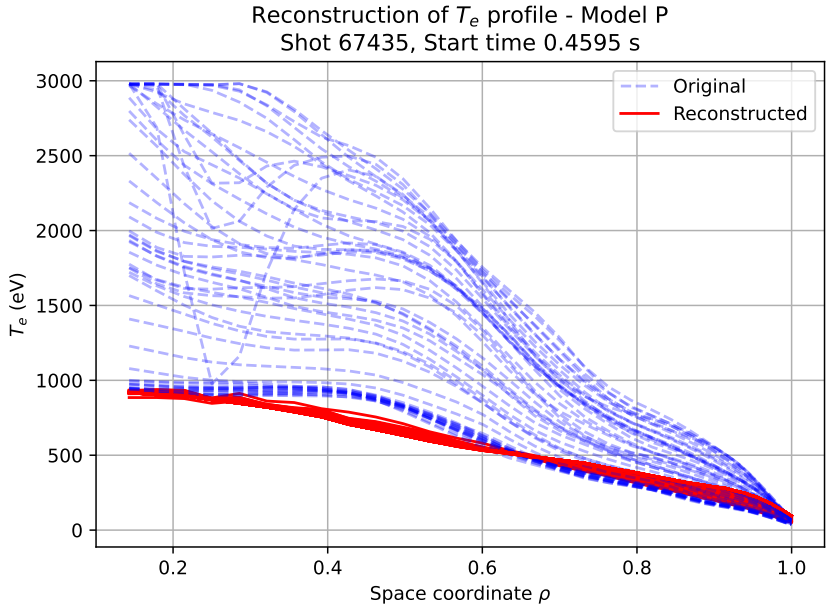


Figure 7.21: Original and reconstructed profile for shot 66293, start time 1.0543s, model P.

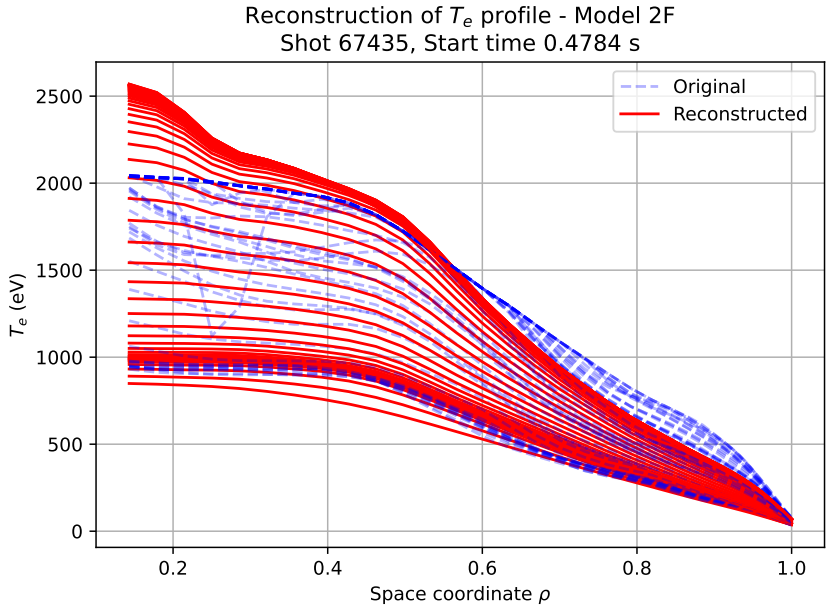


Figure 7.22: Original and reconstructed profile for shot 66293, start time 1.0641s, model 2F.

| Model | # Train set | # Val/Test set | Channels |
|---|---|---|---|
| 1da | 12888 | 3072 | 5-67 |
| 2da | 12888 | 30720 | 26-67 |
| *Parameters common to all models* | | | |
| **Sequence length** | 5000 | | |
| **Stride** | 100 | | |
| **Time resolution** | 10ms | | |
| **Normalization** | Tanh | | |

Table 7.7: Disruption avoidance discharges models for the AE_LSTM.

Each model is trained for 40 epochs using the MSE as loss function, with the same hyperparameters, namely:

- Dropout set to a rate of 0.4, both for the recurrent and linear connections;
- Regularizer set to L1_L2 with coefficient of regularization $\alpha = 10^{-8}$, both for recurrent and linear kernels;
- Adam optimizer with the Tensorflow implementation default settings;
- Adaptive learning rate $lr$, set to a starting value of $10^{-3}$ which is fixed for the first 5 epochs and then decays according to:

$$lr = lr \cdot e^{-0.1}$$

In Table 7.8 are presented the results of the trainings, namely the training and validation losses, as well as training time and dataset loading time.

| Model | Training loss | Validation loss | Training time | Loading time |
|---|---|---|---|---|
| 1da | 0.024 | 0.016 | 20min 1s | 6h |
| 2da | 0.021 | 0.016 | 34min 2s | 6h |

Table 7.8: Training, validation and test loss (normalized data), training time and dataset loading time for the models of Table 7.7.

An evaluation of these models performance compared to the ones of previous sections can be directly made by looking at Table 7.9, where as before are displayed the MSE between the original and reconstructed profiles (denormalized) and its square root.
For model 1da, an average distance between original and reconstructed of 22.08eV is found, which is less than half of what was obtained for model

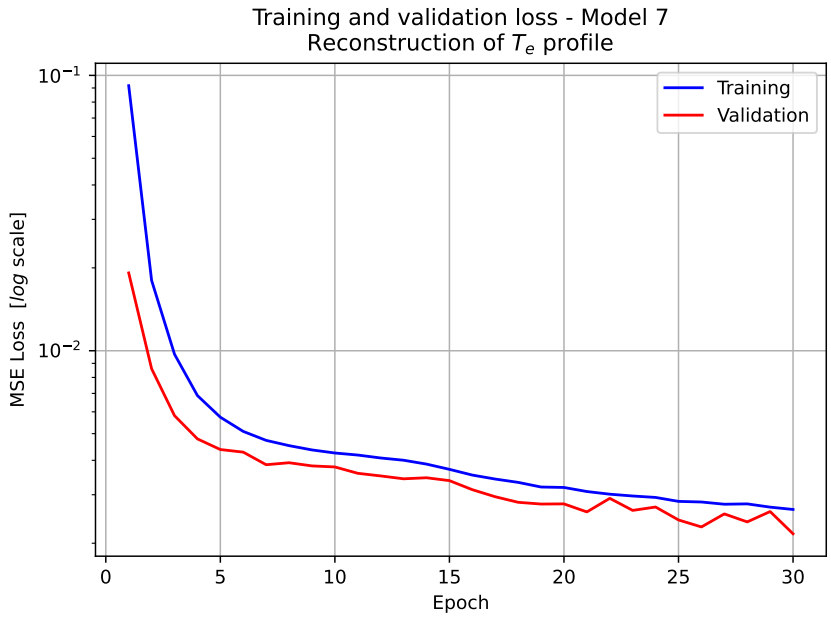


Figure 7.23: Training and validation loss, model 1da. Note the logarithmic scale on the $y$ axis.

2F of the previous section. Nonetheless, model 2F was trained with 120k samples, as opposed to the 12k employed for model 1da. When comparing model 1da with model 1F, which was also trained with 12k samples, one obtains that the average distance for the former is one third of the one for the latter, which is 62.12eV.
For what concerns model 2da, for pedestal reconstruction, there is an improvement with respect to model 3F, which was trained on 12k samples as 2da, but not as dramatic as for the whole profile: 17.50eV the former, 24.33eV the latter. Actually, model 4F, trained for pedestal reconstruction on 120k samples performs quite similarly to model 2da: 17.74eV.

| **Model** | **Test MSE** | **Test sqrt (MSE)** |
|---|---|---|
| 1da | 487.31 | 22.08 |
| 2da | 306.31 | 17.50 |

Table 7.9: MSE of de-normalized test data, root square of MSE of de-normalized test data for the models of Table 7.7.

Following, in Fig. 7.23 and Fig. 7.24 are presented the graphs of the training and validation loss over the training epochs for the models.
No relevant overfitting is present.
In Fig. 7.25 and Fig. 7.26, are shown all original and reconstructed profiles

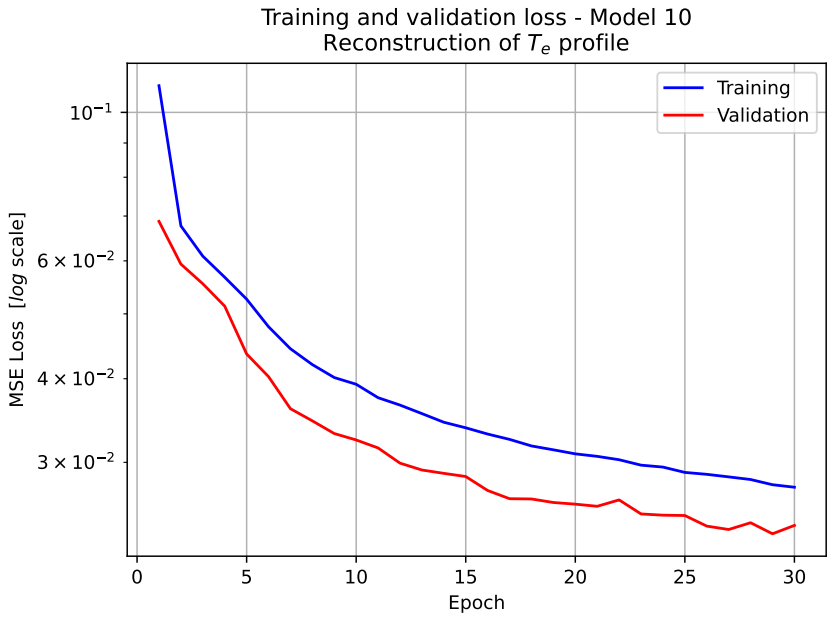


Figure 7.24: Training and validation loss, model 2da. Note the logarithmic scale on the $y$ axis.

in the test set for models 1da, 2da. A time slice is fixed (namely number 20), which is not relevant since each sample has a different starting time (it holds the same reasoning as in the previous subsection).

It can clearly be seen from Fig. 7.25 and Fig. 7.26 how performance increases when restricting the dataset to a single discharge type. Both pedestal and whole profile reconstruction are capable of spanning the whole range of temperature values.

Finally, some examples of reconstruction for specific samples in the test set are also presented. For these, the graphs display a single sample time evolution.

In Fig. 7.27 the worst reconstructed profile in the test set for model 1da is presented (distance from the original profile of 36.87eV). Notice first how this is the worst performance of the network on the test set, but nonetheless reconstruction is overall good. The struggle is in the last time slices, where the model fails to follow the rise of the original profile.

On the other hand in Fig. 7.28 the best reconstructed sample in the test set for model 1da is reported. The distance from the original profile is of only 8.14eV, the lowest value across all models presented in this work. This sample shows little change in temperature values over time: the network is best capable of following a sample which does not modify its temperature range significantly.

Finally, in Fig. 7.30 and Fig. 7.29 are presented, respectively, the best and

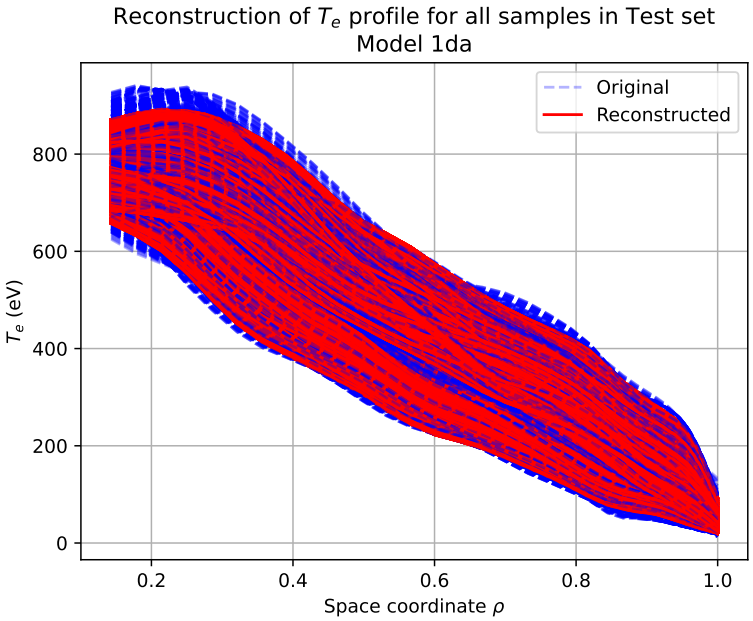


Figure 7.25: Original and reconstructed profiles in the test set, model 1da.

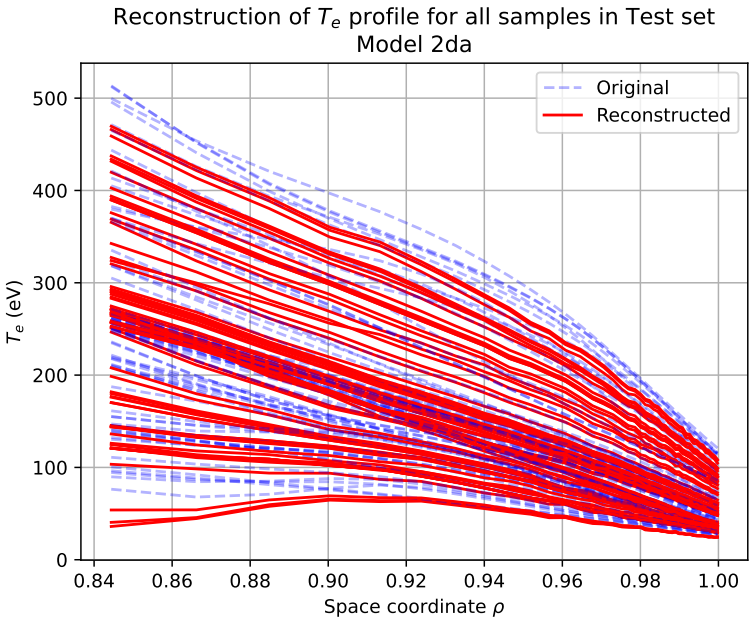


Figure 7.26: Original and reconstructed profiles in the test set, model 2da.

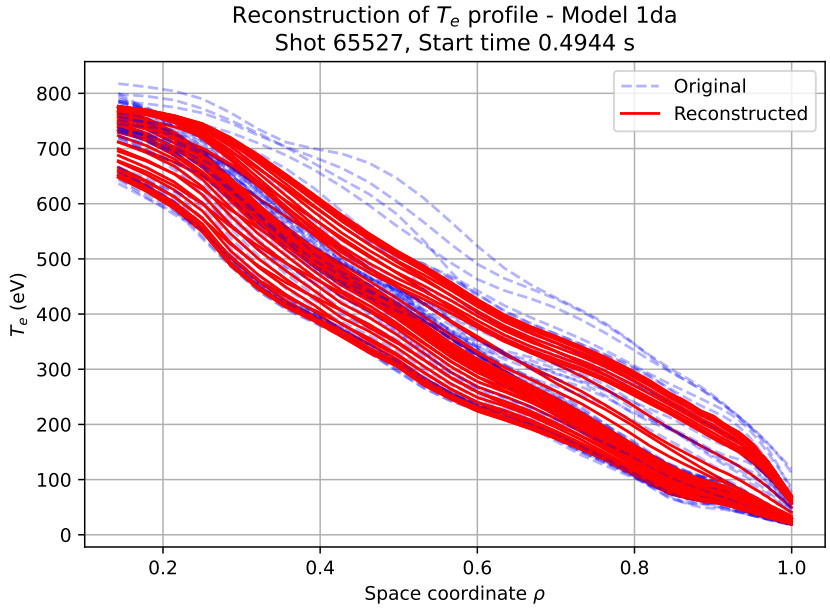


Figure 7.27: Original and reconstructed profile for shot 65527, start time 0.4944s, model P.

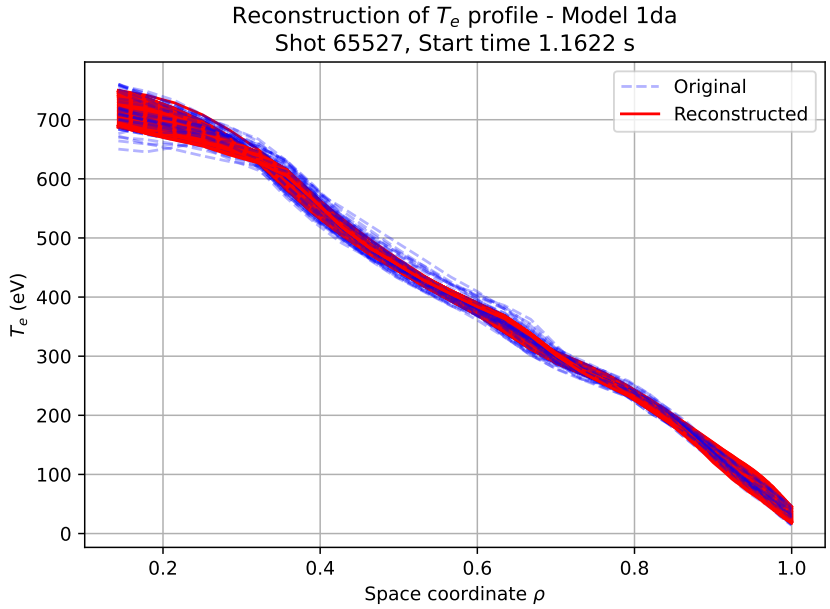


Figure 7.28: Original and reconstructed profile for shot 65527, start time 1.1622s, model 2F.

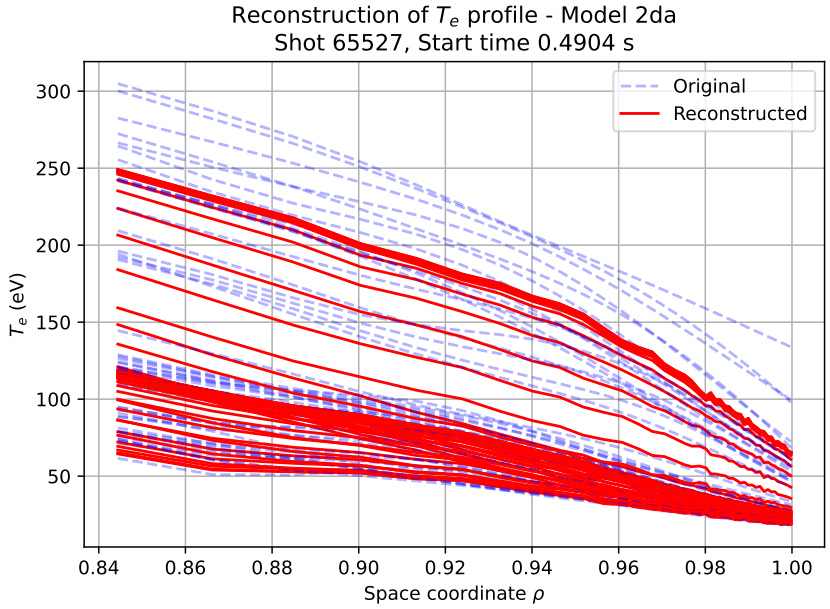


Figure 7.29: Original and reconstructed profile for shot 65527, start time 0.4904s, model P.

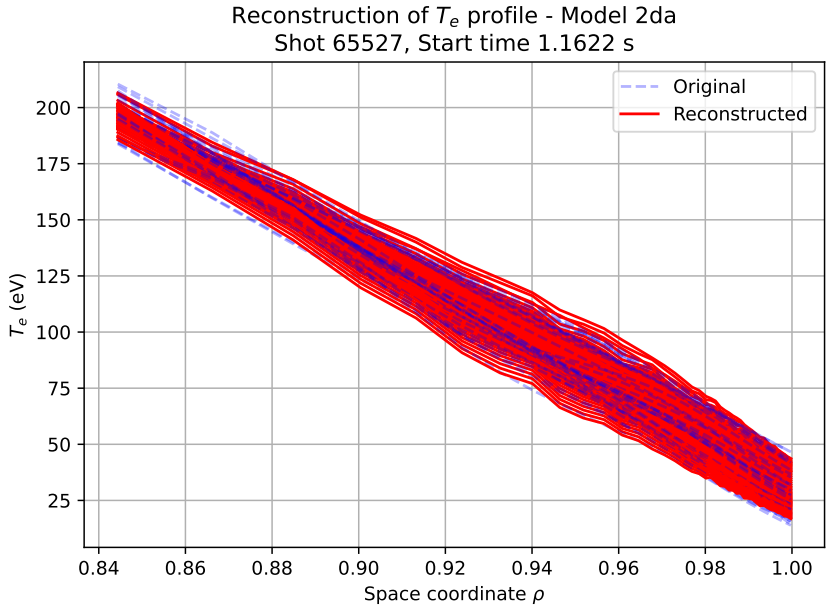


Figure 7.30: Original and reconstructed profile for shot 65527, start time 1.1622s, model 2F.

worst reconstructed profiles in the test set for model 2da. Similar considerations as for model 1da hold, with the worst reconstructed sample displaying the network struggle in following higher temperature values in the last time slices and the best reconstructed one which does not modify much its range of temperature values in time.
Overall, it is concluded that these two models developed for disruption avoidance discharges only are the best performing models in this work. Furthermore, considering model 1da and 2da were trained on a training sample of size 12288, it is expected an additional performance improvement when training with a dataset of bigger size, as the one for models 2F and 4F.

# Conclusions and Outlook

In this work several neural network models to reconstruct the 1D plasma profile of the electron temperature in the TCV tokamak have been explore. First, an autoencoder architecture, with convolutional and recurrent LSTM layers was developed; this model proved ineffective because of the complexity of the data, which would have required a much bigger dataset size. On the other hand, such a scale up of the dataset was unfeasible given the size of the samples and consequently long training time for the model.
Thus, a simpler autoencoder architecture was constructed, composed only of recurrent LSTM layers. This network was tested with several combinations of parameters and finally used to develop two distinct models for pedestal and whole profile reconstruction.
These models achieved a satisfying performance, especially when trained on a dataset of size $\approx$ 120k. They allowed to solve several pathologies of the models trained on smaller datasets, confirming that a bigger training set leads to an improvement in performance.
Furthermore, given that the task complexity also lies in the nature of the dataset, which comprises several different kinds of discharges, an additional model for the same reconstruction task but on a specific discharge type (disruption avoidance) was developed. The intuition was to investigate if a model tailored to a specific type of discharge would perform better.
Indeed, such model surpassed in performance the *generic* ones, even if it was trained on only $\approx$ 12k samples. This leads to the conclusion that focusing on a specific type of discharge could be a more precise approach to solving the task. Alternatively, oversampling the under represented discharge types could also be a solution to the non homogeneity of the dataset with respect to the discharge type.
Overall, it can thus be affirmed that a neural network model(s) for the reconstruction of the electron temperature profile was successfully developed, which can be a confirmation of the usefulness of NN models in the context of nuclear fusion data.

As for what concerns future prospects, more models should be developed, in

the direction of tailoring each model to a specific discharge type. One could argue that training multiple models is expensive, both in terms of time and computational resources. Nonetheless, the models developed in this work have short training times ($\approx$ 30min for 12k samples, $\approx$ 4h for 120k samples) which make them suitable for this proposal.
The bottleneck of this approach actually lies in the input pipeline of the dataset, which is performed by means of the `tf.data.Dataset` object of Tensorflow [36]. This is also one of the points which are intended to be improved: more efficient ways for loading the data are available, among which are, for example, additional parallelization and hardware acceleration.
Moreover, as previously mentioned, over sampling options for underrepresented discharge types in the complete dataset should be explore, as to investigate if it is possible to develop a single model which effectively reconstructs several discharge types.
Furthermore, training the models developed in this work with raw data coming from the Thomson Scattering diagnostic would be a significant line of research: indeed, if one hopes to deploy the model in real time, functioning with raw data is an essential requirement. This direction is encouraged by the fact that being raw data only fitted and giving the regularization properties exhibited by these AE models, no relevant degradation in the performance is expected.
Additionally, a possibility that needs to be explored is to feed the model with data coming from a source which is not the one to reconstruct, but just a proxy signal: specifically, data coming from the Soft X-rays (SSX) diagnostic would be given as input to reconstruct the TS electron temperature profile in output. This would prove useful since the SSX signal is currently available in real time in TCV.
In conclusion, the model developed in this thesis is a first step towards a subsequent model that is capable to perform forecasting of the electron temperature plasma profile. An autoencoder trained for reconstruction can be used as an efficient embedder of information about the profile, which, once encoded, can be used, together with additional variables (such as plasma confinement state, equilibrium parameters, shape parameters, information on MHD modes) as input to a different network with the task of forecasting some timesteps of the profile.

# Bibliography


[1] IEA. *World Energy Outlook 2021*. 2021 (cit. on p. viii).

[2] J. Ongena. "Fusion: a true challenge for an enormous reward". In: *EPJ Web of Conferences* 189 (2018). Ed. by L. Cifarelli and F. Wagner, p. 00015. DOI: 10.1051/epjconf/201818900015 (cit. on p. 2).

[3] J. Wesson and D. Campbell. *Tokamaks*. International series of monographs on physics. Clarendon Press, 2004 (cit. on pp. 3, 8–13).

[4] https://news.mit.edu/2010/fusion-ignition-0510, visited on 3/10/2022 (cit. on p. 11).

[5] J. B. J. Degrave F. Felici et al. "Magnetic control of tokamak plasmas through deep reinforcement learning". In: *Nature* 602.7897 (Feb. 2022), pp. 414–419. DOI: 10.1038/s41586-021-04301-9 (cit. on pp. 19, 28).

[6] F. Hofmann et al. "Creation and control of variably shaped plasmas in TCV". In: *Plasma Physics and Controlled Fusion* 36.12B (Dec. 1994), B277–B287. DOI: 10.1088/0741-3335/36/12b/023 (cit. on pp. 19, 20).

[7] S. Coda et al. "Overview of the TCV tokamak program: scientific progress and facility upgrades". In: *Nuclear Fusion* 57.10 (June 2017), p. 102011. DOI: 10.1088/1741-4326/aa6412 (cit. on p. 20).

[8] C. Nieswand et al. "The Interferometer of TCV". In: *FIX IT* 57.10 (Jan. 1994), p. 102011 (cit. on p. 20).

[9] R.Behn et al. "The Thomson Scattering Diagnostic on TCV". In: *Proceedings of the 7th International Symposium on Laser-aided Plasma Diagnostics (LAPD-7)* (Dec. 1995), pp. 392–397 (cit. on p. 23).

[10] G. Bekefi. "Radiation Processes in Plasmas". In: 1967 (cit. on p. 24).

[11] I. H. Hutchinson. *Principles of Plasma Diagnostics*. 2nd ed. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511613630 (cit. on p. 24).

[12] R. Saravanan and P. Sujatha. "A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification". In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE. 2018, pp. 945–949 (cit. on p. 27).

[13] M. Caron et al. *Deep Clustering for Unsupervised Learning of Visual Features*. 2018. DOI: 10.48550/ARXIV.1807.05520 (cit. on p. 27).

[14] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961 (cit. on p. 28).

[15] O. I. Abiodun et al. "Comprehensive review of artificial neural network applications to pattern recognition". In: *IEEE Access* 7 (2019), pp. 158820–158846 (cit. on p. 28).

[16] D. N. Fente and D. K. Singh. "Weather forecasting using artificial neural network". In: *2018 second international conference on inventive communication and computational technologies (ICICCT)*. IEEE. 2018, pp. 1757–1761 (cit. on p. 28).

[17] J. Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (July 2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2 (cit. on p. 28).

[18] A. Pau et al. "A First Analysis of JET Plasma Profile-Based Indicators for Disruption Prediction and Avoidance". In: *IEEE Transactions on Plasma Science* 46.7 (July 2018), pp. 2691–2698. DOI: 10.1109/tps.2018.2841394 (cit. on p. 28).

[19] A. Pau et al. "A machine learning approach based on generative topographic mapping for disruption prevention and avoidance at JET". In: *Nuclear Fusion* 59.10 (Aug. 2019), p. 106017. DOI: 10.1088/1741-4326/ab2ea9 (cit. on p. 28).

[20] J. Abbate, R. Conlin, and E. Kolemen. "Data-driven profile prediction for DIII-D". In: *Nuclear Fusion* 61.4 (Mar. 2021), p. 046027. DOI: 10.1088/1741-4326/abe08d (cit. on p. 28).

[21] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang. "Predicting disruptive instabilities in controlled fusion plasmas through deep learning". In: *Nature* 568.7753 (Apr. 2019), pp. 526–531. DOI: 10.1038/s41586-019-1116-4 (cit. on p. 28).

[22] J. X. Zhu et al. "Corrigendum: Hybrid deep learning architecture for general disruption prediction across tokamaks (2021 Nucl. Fusion 61 026007)". In: *Nuclear Fusion* 61.4 (Mar. 2021), p. 049501. DOI: 10.1088/1741-4326/abe2e3 (cit. on p. 28).

[23] F. Matos et al. "Classification of tokamak plasma confinement states with convolutional recurrent neural networks". In: *Nuclear Fusion* 60.3 (Feb. 2020), p. 036022. DOI: 10.1088/1741-4326/ab6c7a (cit. on pp. 28, 51, 52).

[24] F. Matos et al. "Plasma confinement mode classification using a sequence-to-sequence neural network with attention". In: *Nuclear Fusion* 61.4 (Mar. 2021), p. 046019. DOI: 10.1088/1741-4326/abe370 (cit. on p. 28).

[25] G. Marceca. "Detection of plasma confinement states in the TCV tokamak". In: 2020 (cit. on p. 28).

[26] A. Jalalvand et al. "Real-Time and Adaptive Reservoir Computing With Application to Profile Prediction in Fusion Plasma". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–12. DOI: 10.1109/tnnls.2021.3085504 (cit. on pp. 28, 35).

[27] C. Rea et al. "Progress Toward Interpretable Machine Learning–Based Disruption Predictors Across Tokamaks". In: *Fusion Science and Technology* 76.8 (Sept. 2020), pp. 912–924. DOI: 10.1080/15361055.2020.1798589 (cit. on p. 28).

[28] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 33).

[29] https://www.mdsplus.org/index.php/Introduction, visited on 3/10/2022 (cit. on p. 41).

[30] A. Pau et al. "A tool to support the construction of reliable disruption databases". In: *Fusion Engineering and Design* 125 (Dec. 2017), pp. 139–153. DOI: 10.1016/j.fusengdes.2017.10.003 (cit. on p. 41).

[31] D. Singh and B. Singh. "Investigating the impact of data normalization on classification performance". In: *Applied Soft Computing* 97 (Dec. 2020), p. 105524. DOI: 10.1016/j.asoc.2019.105524 (cit. on pp. 47, 48).

[32] H. Zohm. "Edge localized modes (ELMs)". In: *Plasma Physics and Controlled Fusion* 38.2 (1996), p. 105 (cit. on p. 51).

[33] L. Zhu and N. Laptev. "Deep and Confident Prediction for Time Series at Uber". In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, Nov. 2017. DOI: 10.1109/icdmw.2017.19 (cit. on p. 52).

[34] https://www.tensorflow.org/tutorials/load_data/tfrecord#read_the_tfrecord_file, visited on 3/10/2022 (cit. on p. 55).

[35] https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle, visited on 3/10/2022 (cit. on p. 56).

[36] A. M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on pp. 57, 65, 96).

[37] https://github.com/cristinaventurini/master_thesis_SPC/blob/main/ml.yml, visited on 3/10/2022. Repository maintained by this thesis author, for informations: cristinaventurini.98@gmail.com. (cit. on pp. 57, 65).

[38] T. Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019 (cit. on p. 62).