UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of data

Final Dissertation

Deep Learning Models for real-time Fusion Device Data Compression Algorithms

Thesis supervisor
Prof. Jacopo Pazzini

Thesis co-supervisor
Dr. Andrea Rigoni Garola

Candidate
Marija Mojsovska

Academic Year 2023/2024

Abstract

The growing demand for sustainable energy solutions has driven significant interest in the exploration of nuclear fusion, a clean and virtually limitless energy source. Among the prominent fusion experiments, the RFX-mod2 device at Consorzio RFX in Padua stands out for its very high spatial resolution magnetic diagnostic with more than 1700 sensors, along with more than 200 actuator coils independently controlled. However, the immense volume of data generated by this diagnostic system presents a significant challenge for real-time data transfer and storage, necessitating the development of an efficient data compression solution.

This thesis proposes a deep learning-based compression algorithm built on an autoencoder architecture and trained using historical data from the RFX database. The algorithm employs attention mechanisms to capture long-range dependencies and extract meaningful patterns while ensuring computational efficiency through parallelization. Strategically placing the attention blocks in the decoder, reduces the dimensionality of the input to the attention function, significantly decreasing the number of model parameters, and lowering computation complexity. In addition, a distortion constraint is incorporated to limit the reconstruction error in a desired range.

The proposed algorithm outperforms widely used general purpose compression methods, such as LFZip and SZ3, on datasets with multiple variables. Furthermore, the performance advantage grows as the error constraints are relaxed. These results demonstrate the algorithm's potential for real-time implementation, paving the way for more efficient data handling in fusion research.

Contents

Introduction

The growing global demand for energy, coupled with the environmental concerns and resource limitations of current energy production methods, highlights the urgent need for alternatives. Nuclear fusion holds promise as a sustainable and clean energy source. It does not depend on external factors (like solar or wind) and it does not involve production of greenhouse gases or long-lived radioactive waste.

The leading approach for obtaining controlled nuclear fusion relies on containing a hot plasma confined within a magnetic field that prevents it from reaching the reactor walls and losing its energy. Two prominent fusion device configurations are the Tokamak and the Reversed Field Pinch (RFP). The RFX experiment, located at Consorzio RFX in Padova can generate plasmas in both magnetic configurations.

One of the standout features of the recently enhanced RFX-mod2 experiment is its exceptionally high spatial resolution magnetic diagnostic system, equipped with over 1700 sensors, coupled with more than 200 independently controlled actuator coils. These numbers surpass those of any existing fusion plasma device by nearly tenfold, giving unprecedented full 3D plasma shape reconstruction and control capability, with the flexibility to develop and test a wide variety of control schemes. As a result the RFX-mod2 experiment offers unique opportunities for the development and validation of advanced machine learning and deep learning algorithms aimed at plasma control.

The volume of data generated by the diagnostic system of the experiment presents a significant challenge. Transferring all sensor information to the central control system in real time is infeasible due to bandwidth limitations. To address this, a dual-channel acquisition strategy has been adopted: one channel for low-latency, low-bandwidth data acquisition, specifically designed for the control system, and a second channel for full-resolution data. The second channel takes advantage of the transient nature of the experimental setup by buffering the data locally and storing all the acquired raw data on the central acquisition server after the pulse.

This thesis focuses on the development of a deep learning based compression algorithm trained on historical data from the RFX database. The proposed approach leverages the attention mechanism to capture long-range dependencies and meaningful patterns in the data while keeping low computational complexity, making it well-suited for application in fusion device data compression.

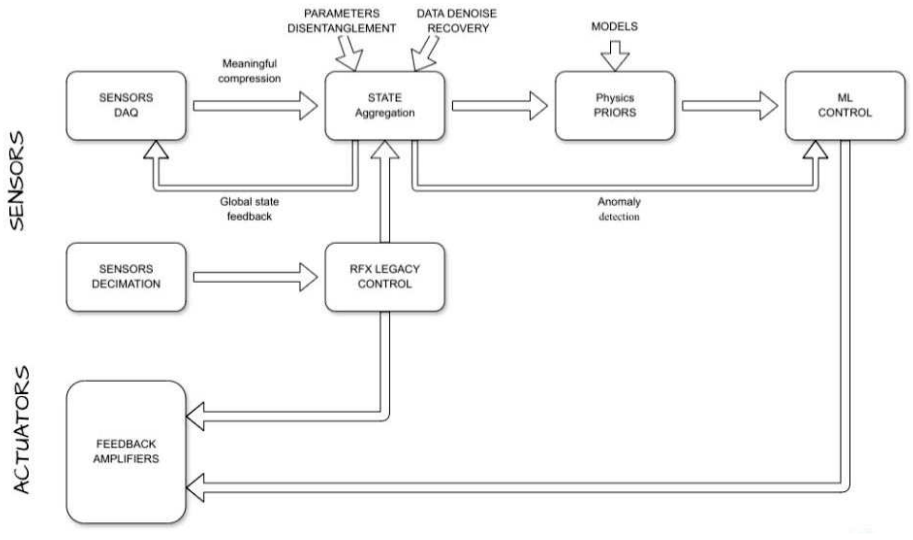

Figure 1: Schematic diagram of the proposed approach for the acquisition pipeline from sensor to the control system feedback.

The ultimate goal is to integrate the compression model into the experiment's acquisition pipeline. A proposed approach for the acquisition pipeline from sensor to the control system feedback is shown in Fig 1. It all starts with the compression of the sensor data into latent states. After the compression, state information from multiple diagnostics is aggregated along with relevant information from the RFX legacy control system. This stage also includes data denoising and recovery to ensure signal integrity. Subsequently, physics models are utilized for identifying plasma characteristics in the data. This information is then used by the control system to make informed decisions. The proposed system is designed as an additional channel to complement the existing feedback control system, leveraging compact and efficient data representations to potentially improve overall performance and responsiveness.

The thesis is structured as follows: Chapter 1 introduces the fundamentals of nuclear fusion and plasma physics, with a focus on magnetic confinement and magnetohydrodynamic (MHD) stability, followed by a description of the RFX experiment. Chapter 2 covers information theory and data compression with an overview of standard time-series compression algorithms. Chapter 3 focuses on machine learning and introduces some of the techniques used in this work. In chapter 4, a detailed description of the proposed compression algorithm. Results are presented in chapter 5 including a comparison with other compression algorithms. To end with conclusions and some remarks on future prospects.

Chapter 1

Nuclear Fusion and the RFX experiment

Current energy production methods face significant challenges, including the well-documented issue of finite energy resources and serious environmental concerns such as greenhouse gas emissions and long-lived radioactive waste. The urgent need for alternative solutions is clear, and nuclear fusion holds promise as a sustainable and clean energy source.

1.1 Introduction to Nuclear Fusion

Nuclear fusion is the process in which two light nuclei combine to form a heavier nucleus releasing energy in the process. This energy is carried by the products of the reaction, the resulting nucleus, and other particles created in the process (neutrons, photons, etc.), in form of kinetic energy.

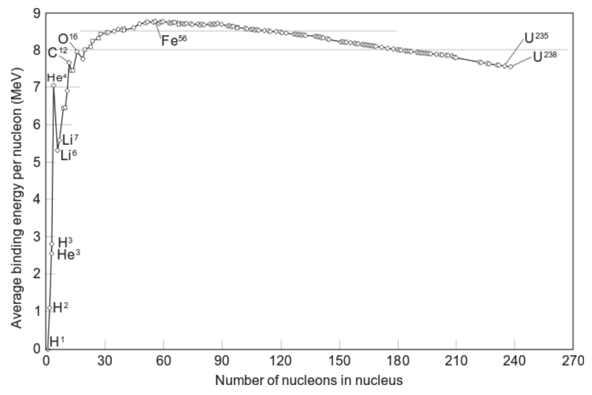

Figure 1.1: Average Binding energy per nucleon

The energy released in nuclear reactions is equal to the difference in binding energy between the products and the reactants. Figure 1.1 shows the binding energy per nucleon as a function of the number of nucleons. Since increase in binding energy corresponds to energy production, we have *fusion* reactions dominate the left side of the curve for light nuclei, where energy is released by combining nuclei while *fission* reactions dominate the right side of the curve, where energy is released by splitting heavy nuclei into lighter ones.

An important factor to consider for achieving nuclear fusion is the Coulomb scattering. The cross-section of Coulomb scattering is an order of magnitude larger than the fusion cross-section. This is why simple methods like colliding beams or beam hitting a target are not effective. Instead, it is crucial to confine the reactants in a limited region of space for a sufficiently long time for them to undergo fusion reaction without being lost due to elastic collisions. To achieve this, a *plasma* containing both ionic species is created and confined in space usually by a magnetic field.

Thermonuclear fusion as an energy source is a common phenomenon in nature, powering the Sun and other stars. Fusion reactions in stars follow the *proton-proton chain*, converting the main source, protons, into helium nuclei, releasing energy in the process.

$$\begin{aligned} \mathrm{p} + \mathrm{p} &\longrightarrow {}^2_1\mathrm{D} + \mathrm{e}^+ + \nu_e \\ {}^2_1\mathrm{D} + \mathrm{p} &\longrightarrow {}^3\mathrm{He} + \gamma \\ {}^3\mathrm{He} + {}^3\mathrm{He} &\longrightarrow {}^4\mathrm{He} + \mathrm{p} + \mathrm{p} \end{aligned}$$

Fusion research does not follow the *p-p* reaction we see in stars because of its very small interaction cross-section, instead it focuses mainly on reactions involving Deuterium ${}^2_1H$ and Tritium ${}^3_1H$.

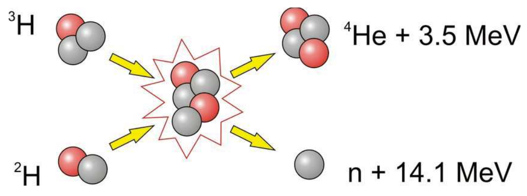

Figure 1.2: Deuterium-Tritium fusion reaction with production of α particle, neutron and 17.6MeV of energy.

Deuterium is a stable hydrogen isotope, abundant in nature. On the other hand, tritium is radioactive and decays with a half-life of 12.3 years. That being the case, a fusion reactor will need a method to produce it. This can be done through a process called *tritium breeding*. The neutrons produced by the fusion reactions escape the magnetic confinement of the plasma and are absorbed by the blanket that is covering the surrounding walls. The role of the blanket is to screen the superconducting magnets and other external components from the neutron radiation, and convert the neutron energy into heat for power generation. In addition, it contains lithium that by absorbing an incoming neutron recombines into tritium and helium through the following reactions:

$$^{6}\mathrm{Li} + \mathrm{n} \longrightarrow {}^{4}\mathrm{He} + \mathrm{T} + 4.8\mathrm{MeV}$$
$$^{7}\mathrm{Li} + \mathrm{n} \longrightarrow {}^{4}\mathrm{He} + \mathrm{T} + \mathrm{n} - 2.5\mathrm{MeV}$$

The tritium can then be removed from the blanket and recycled into the plasma as fuel, eliminating the need for external transportation and lowering significantly the safety concerns.

1.1.1 Safety concerns

One of the advantages of fusion technology is the safety. Compared to fission reactors there is no risk of chain reactions. In the event of an accident, energy production is interrupted with the stop of the reactor operation, and the stored energy in the reactor components is released slowly, allowing sufficient time for operator intervention.

The main concerns are related to tritium and neutron-induced activation of materials. The decay time for the radioactive waste produced with material activation is a century, which is not very long compared with fission-produced radioactive waste. Tritium is produced on-site but in the case of contamination, it is not retained in the human body for a long time. Tritium decays with emission of a low energy beta particle, which range in organic tissue is less than the thickness of the epidermis. Tritiated water is much more dangerous, while it follows the water cycle of the human body when ingested, some of the tritium (around 10%) can bind with organic tissues and be retained for up to 300 days.

1.2 Power balance of fusion reactor

In order for energy production to occur within a fusion reactor, the energy balance must be positive, which implies that the fusion process must generate more energy than the total energy required to initiate and sustain the reactions, including any external energy inputs used for heating or maintaining the plasma.

The evolution of the thermal energy per unit volume contained in the plasma is given by the following equation:

$$\frac{dw}{dt} = p_H + p_\alpha - p_L - p_R$$

- p_H: external heating
- $p_\alpha = \frac{1}{4}n^2\langle\sigma v\rangle E_\alpha$: heating power from α particles

  $\langle\sigma v\rangle$ = reactivity of the α particles

In Deuterium-Tritium fusion reactions, the energy carried by the neutrons exits the plasma confinement and becomes available for electricity production in the form of thermal energy of a cooling fluid. On the other hand, the charged α particles remain confined and release their energy to the plasma, by collisions.

- $p_L = \frac{w}{\tau_E}$: power lost due to transport processes, including convection and conduction

  $\tau_E =$ energy confinement time, a measure of how quickly energy is lost from the plasma

- $p_R = p_b = \alpha_b n^2 T^{1/2}$: power lost through electromagnetic radiation, primarily bremsstrahlung

  $\alpha_b = 5.35 \cdot 10^{-37} W m^3 keV^{-1/2}$

With adequate confinement of the plasma a point can be reached when the plasma temperature can be maintained solely by the α particles without applying any external heating, this is called the **ignition condition**.

$$n\tau_E > \frac{12T}{\langle \sigma v \rangle E_\alpha - 4\alpha_b T^{1/2}} \tag{1.1}$$

Considering operation temperature for a fusion reactor in the range $10 - 20keV$, radiation losses are low enough to be neglected. This results in a condition on the triple product of density, temperature and confinement time.

$$nT\tau_E > 3 \cdot 10^{21} m^{-3} keV s \quad ^{1} \tag{1.2}$$

1.3 Magnetic confinement

Among the various configurations of magnetic confinement proposed throughout the years, the torus configuration appeared to be the most topologically advantageous. Two magnetic field components toroidal and poloidal are superimposed to give rise to field lines that twist helically around the torus confining the ionized particles.

In cylindrical approximation, the toroidal geometry can be described in a $(r, \theta, z = R_0\phi)$ reference system, where r is the radial coordinate, θ is the poloidal angle and ϕ is the toroidal angle. The major radius of the torus is R_0 and the minor radius is a.

There are three principal categories of toroidal magnetic configurations:

- Tokamak

[1] Result obtained considering flat plasma profiles

- Reversed field pinch (RFP)
- Stellarator

In Tokamak and Reversed field pinch the toroidal field is produced by external coils which approximate a toroidal solenoid. The poloidal field is generated by toroidal current inside the plasma, called *plasma current*. The plasma current is induced by a second system of coils with time-dependent current which by Faraday's law induces a toroidal electric field that drives the plasma current. The main difference between the Tokamak and the RFP is the strength of the magnetic field components. The Tokamak has a toroidal magnetic field much stronger than the poloidal. In RFP the strength of the magnetic field is of the same order of magnitude in both directions.

On the other hand, the Stellarator is designed to be currentless. The helical structure of the magnetic field lines is generated entirely by external coils.

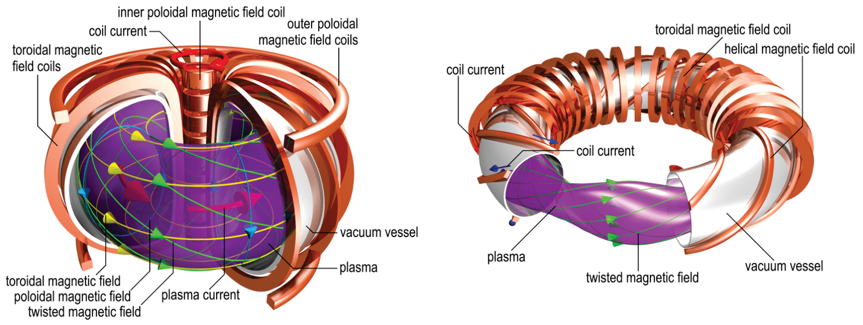

Figure 1.3: Tokamak on the left and Stellarator on the right

1.3.1 Magnetohydrodynamics (MHD)

The fully ionized plasma, found in fusion devices, can be considered a conducting fluid, hence a magnetohydrodynamic (MHD) approach can be used to describe it.

The movement of the plasma can be expressed by a single fluid model based on the fluid continuity and movement equations. To simplify the description it is reasonable to consider the *charge quasi-neutrality* approximation, considering $n_i = n_e$.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \tag{1.3a}$$

$$\rho \left(\frac{\partial u}{\partial t} + (u \cdot \nabla)\, u \right) = \sigma + j \times B - \nabla p \tag{1.3b}$$

$\rho = \rho_i + \rho_e$	total mass density
$p = p_i + p_e$	total kinetic pressure
$\mathbf{u} = (\rho_i \mathbf{u}_i + \rho_e \mathbf{u}_e)/\rho$	mass fluid velocity
$\sigma = e(n_i + n_e)$	charge density

The **resistive MHD** model is obtained by incorporating Maxwell's equations and Ohm's law into the continuity(1.3a) and plasma motion equations (1.3b) to account for electromagnetic interactions.

$$\begin{aligned}
\nabla \cdot \mathbf{B} &= 0 \\
\nabla \times \mathbf{B} &= \mu_0 \mathbf{J} \\
\nabla \cdot \mathbf{E} &= 0 \\
\nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\
\mathbf{u} \times \mathbf{B} + \mathbf{E} &= \eta \mathbf{J}.
\end{aligned} \tag{1.4}$$

In this framework the dynamics of magnetic and fluid velocity fields are expressed in terms of convection of the magnetic field with the plasma and resistive diffusion of the field across the plasma (1.5).

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{u} \times \boldsymbol{B}) + \frac{\eta}{\mu_0} \nabla^2 \boldsymbol{B} \tag{1.5}$$

1.3.2 MHD stability

The study of the plasma MHD instability mainly concerns the perturbations of the ideal system starting from a magneto-static equilibrium point. When both stationary and static conditions are satisfied the **Force balance equation** applies.

$$j \times B = \nabla p \tag{1.6}$$

It describes the balance between the pressure gradient which tends to move the plasma from the inner, hotter and more dense regions, to the outer ones, and the Lorentz force, that opposes this tendency, resulting in a magnetic confinement.

In the Force balance equation (1.6) it is evident that the current density j and the magnetic field B are orthogonal to the pressure gradient. This leads to the following conclusions:

- $B \cdot \nabla p = 0 \rightarrow$ There is no pressure gradient along magnetic field lines, meaning that magnetic surfaces are also surfaces of constant pressure.
- $j \cdot \nabla p = 0 \rightarrow$ Pressure remains constant along current density lines, indicating that current lines lie on the magnetic surfaces.

In axisymmetric toroidal equilibria, the magnetic field lines lie on nested magnetic surfaces, wound around the magnetic axis. This surfaces are described by the Grad-Shafranov equation:

$$\begin{gathered} \Delta^* \psi = -\mu_0 R^2 p' - \mu_0^2 f f' \\ \Delta^* = R \frac{\partial}{\partial R} \left(\frac{1}{R} \frac{\partial}{\partial R} \right) + \frac{\partial^2}{\partial Z^2} \end{gathered} \tag{1.7}$$

ψ and f are the magnetic and current flux functions $(B \cdot \nabla \psi = 0, j \cdot \nabla f = 0)$

The constant value contours of the solution $\psi(R, Z)$ give the shape of the magnetic surfaces on the poloidal plane.

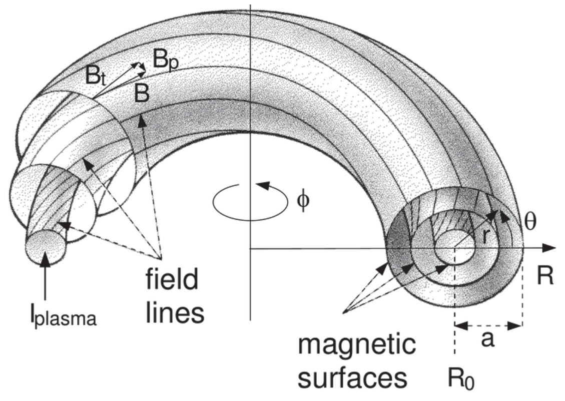

Figure 1.4: Geometry of the magnetic field lines and nested magnetic surfaces

Under certain conditions, small disturbances in the equilibrium state can cause the development of significant instabilities in the plasma. The effect of generic perturbations $\tilde{\psi}$ can be analysed using the Fourier transform

$$\tilde{\psi}(\mathbf{r}, t) = \sum_k \tilde{\psi}_k(r) e^{i(\mathbf{k} \cdot \mathbf{r} - \omega t)} = \sum_k \tilde{\psi}_k e^{i(m\vartheta + n\varphi - \omega t)} \tag{1.8}$$

where $\mathbf{r} = (r, \vartheta, \varphi)$ is the displacement vector in toroidal coordinates, and $\mathbf{k} = (k_r, k_\theta = \frac{m}{r}, k_\phi = \frac{n}{R})$ is the wave vector with m and n the poloidal and toroidal mode respectively.

Each (m, n) couple describes a helical perturbation mode. A mode can become unstable if the resonance condition $\mathbf{k} \cdot \mathbf{B} = 0$, between its wave vector and the equilibrium magnetic field, is satisfied. If the resonance condition is not satisfied the perturbation will be suppressed by the *magnetic tension*, a restoring force that opposes deformation of the magnetic field lines, like bending or squeezing. When the resonance condition occurs, the perturbation shifts all the points of a magnetic field line by the same quantity, in this way the field line is not curved by the perturbation, and therefore a stabilizing effect is not present.

An important parameter in determining the plasma stability is the **safety factor**. The safety factor is the toroidal angle that a magnetic field line needs to travel in order to make a full poloidal turn. On large aspect ratio tokamaks (major radius much larger than the minor radius), it can be approximated by:

$$q(r) = \frac{rB_\phi(r)}{RB_\theta(r)} \tag{1.9}$$

In a resonant surface, the maximum of the magnetic field perturbation has the same periodicity as the magnetic field line.

$$q = -\frac{m}{n} \tag{1.10}$$

MHD instabilities can be classified based on the triggering mechanism: pressure driven, current driven or particle driven. The pressure driven instabilities arise from the combined effect of pressure gradient and magnetic field curvature. The current driven modes originate from gradients in the current density profile, kink modes are an example of current driven instabilities.

Kink instabilities are characterized by a poloidal mode number $m = 1$. They lead to sharp twists and curves of the magnetic field lines and a gradual increase in the magnetic pressure at the lower surface concave curve. The modes with poloidal number $m = 0$ are called *sausage* instabilities. They are characterized by a uniform kinetic pressure and magnetic pressure that tends to be stronger at the saddle, resulting in bulges that rapidly grow in time. These unstable modes, however dangerous, can be stabilized by a strong toroidal field.

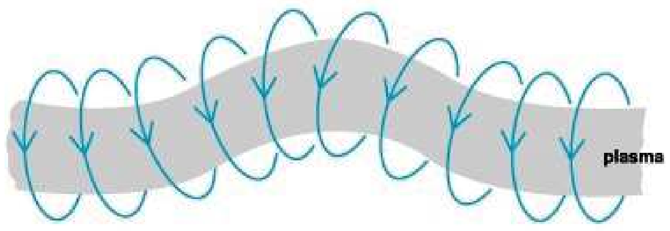

Figure 1.5: Kink instability

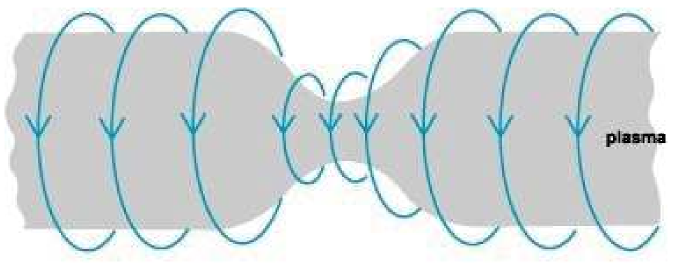

Figure 1.6: Sausage instability

Tearing modes are another significant category of instabilities that critically impact the efficiency of fusion devices. The term “tearing mode” refers to an instability in which the resistivity of the plasma allows the magnetic field topology, which would be “frozen” in an ideal plasma, to relax to a more energetically favorable state. This relaxation induce a magnetic reconnection within the plasma, leading to a redistribution of magnetic field lines and potentially resulting in a general loss of confinement. For this reason, understanding and controlling tearing modes are essential for maintaining the stability of plasma in magnetic confinement fusion devices. Among these, the *Neoclassical Tearing Mode (NTM)* is particularly noteworthy. It is a current-driven resistive instability, distinguished by variations in the current density that align parallel to the magnetic field lines. This instability leads to a fundamental alteration in the topology of the magnetic surfaces, which facilitates the formation of rotating magnetic islands. Within these islands, the confinement of plasma is severely compromised, as evidenced by noticeable reductions in the pressure and temperature profiles, which directly correspond to the dimensions of the islands. A critical observation is that accurately modeling the evolution of tearing modes within a plasma configuration cannot be achieved using an ergodic description. Instead, it requires the assumption of time-varying chaotic behavior.

To mitigate these effects, various strategies have been developed. A prevalent approach involves the use of resonant magnetic perturbations, which can alter the magnetic topology in a controlled manner to suppress tearing modes. Additionally, real-time control systems employing feedback mechanisms have been implemented to dynamically adjust the magnetic field and plasma parameters based on the detection of tearing modes. Moreover, predictive modeling plays a crucial role in understanding the behavior of these modes under different operational conditions. Advanced simulation tools that incorporate comprehensive physics models are used to predict the onset and evolution of these modes, allowing for preemptive measures to be taken before they can adversely affect the plasma confinement. Consequently, the objective of this project is to maximize the extraction of information from the edge magnetic sensor. This will enhance the quality of the signal pertaining to the current plasma configuration, which will be supplied to a real-time control system. The control system will then generate an appropriate feedback signal, crucial for effective plasma management.

1.3.3 Tokamak

The tokamak configuration is characterized by a strong toroidal magnetic field component and a weak poloidal component which results in a concave safety factor profile with a minimum on the vacuum chamber axis. The edge q value (usually $q_{edge} = 3 - 4$) depends on two parameters, the toroidal field and plasma current, giving full control of the quantity. On the other hand, the central q is determined by the on-axis temperature, which defines the resistivity and is partially out of the operator's control. In order to maintain plasma stability, the safety factor needs to respect the Kruskal-Shafranov limit ($q > 1$) [2]. This is because of the onset of Sawtooth oscillations, due to the ($m = 1, n = 1$) mode, causing confinement losses and perfect conditions for the growth of a strong kink instability. The Kruskal-Shafranov limit establishes a maximum plasma current achievable solely through ohmic heating, therefore additional heating methods are necessary to achieve fusion conditions.

1.3.4 Reverse field pinch (RFP)

Unlike in the tokamak, the magnetic field components in RFP are of comparable strength resulting in a toroidal magnetic field reversal near the plasma edge, and easier plasma control. Following the behavior of the toroidal magnetic field, the safety factor changes sign at the plasma edge. The RFP is a configuration with a low safety factor value ($q \ll 1$ at the edge). This allows for the presence of multiple resonant modes. Having a low safety factor does not negatively impact the RFP configuration. The resilience of the RFP arises primarily from two factors: a high level of magnetic shear, which stabilizes magnetic modes, and the presence of a conducting shell, which suppresses rapidly growing magnetic perturbations. RFP is not subject to the same issues as the tokamak, particularly the Kruskal-Shafranov limit and consequently the plasma current restrictions. However, increasing the current leads to high MHD activity resulting in a chaotic plasma and poor confinement. Despite that resonant instabilities are crucial for sustaining the RFP configuration through the **dynamo process** [3]. If the RFP equilibrium is governed solely by resistive diffusion, the characteristic reversal of the toroidal magnetic field would be lost shortly after being established.

The steep spatial gradients of the current density drive unstable a wide range of tearing modes, placing the plasma in what is known as the *multiple-helicity (MH)* state. In this state, the magnetic islands associated with each tearing mode overlap, creating a chaotic magnetic field that drives anomalous heat and particle transport. This spectrum of tearing modes is important because their interaction generates the mean dynamo electric field necessary to sustain the magnetic configuration over time.

We can represent the magnetic and velocity fields as the sum of a mean field and a fluctuating component. When we do this, the dynamo electric field, given by the product of the velocity and magnetic field fluctuations associated with the dynamo modes, naturally emerges in the parallel Ohm's law.

$$\begin{gathered} E_{0\parallel} + E_{d\parallel} = E_{0\parallel} + \langle \tilde{\mathbf{v}} \times \tilde{\mathbf{b}} \rangle_{\parallel} = \eta j_{0\parallel} \\ \mathbf{B} = \mathbf{B_0} + \tilde{b}, \quad \mathbf{v} = \mathbf{v_0} + \tilde{v} \end{gathered} \tag{1.11}$$

MHD simulations suggest that the dynamo mechanism needed to sustain the RFP equilibrium can be achieved through the nonlinear saturation of a single tearing mode, resulting in the so-called *single helicity (SH)* states. Unlike the multiple helicity states, SH states exhibit significantly lower levels of magnetic turbulence, making them advantageous for plasma confinement.

Quasi-SH (QSH) states are the closest to the theoretical SH state that has been experimentally observed. In these states, a single $m = 1$ mode dominates the magnetic spectrum, though other modes with small, finite amplitudes still exist, causing some residual magnetic chaos. As the plasma current increases, QSH states tend to become more persistent and exhibit greater purity, leading to enhanced energy confinement. The increase of the plasma current is limited by the increase in temperature at the plasma edge and the plasma-wall interactions.

1.4 The RFX experiment

The roots of RFP technology can be traced back to the 1960s with early devices such as the ZETA experiment. ZETA, short for Zero Energy Thermonuclear Assembly was a major experiment build at the Atomic Energy Research Establishment in the United Kingdom that operated from 1954 to 1958. It was part of the early history of fusion power research and despite its failure to achieve fusion it contributed to numerous important advances in the field. The machine presented highly unstable regimes, sometime followed by sporadic quiescent phases difficult to interpret. In 1974 C. B. Taylor proposed a theoretical model to explain these observations [4]. He proved that the quiescent condition in ZETA were the result of a spontaneous reversal of the toroidal field that took place when the toroidal field coils were short circuited. The phenomenon was driven by the system's desire to preserve magnetic helicity. These observations lead to the idea of a reactor that would work using extended confinement time rather than increased density, similar to the stellarator but with lower field strength and reduced energy request for maintaining the confinement. This approach is now known as the *Reversed field pinch.* In the late 70s a research on this phenomena started in Padua that lead the way for the making of *RFX (Reversed Field experiment).*

RFX is a reversed field fusion machine with a major radius $R_0 = 2m$ and a minor radius $a = 0.459m$, able to sustain a plasma current of $2MA$, highest operation value of any RFP experiment [5].

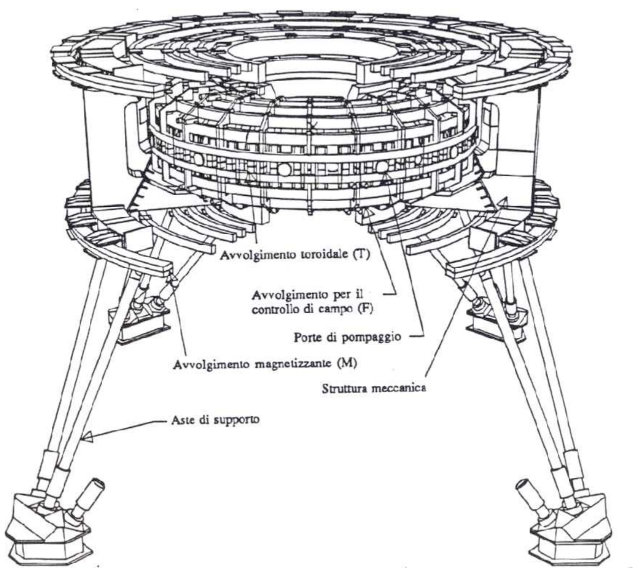
Avvolgimento toroidale (T)
Avvolgimento per il controllo di campo (F)
Porte di pompaggio
Avvolgimento magnetizzante (M)
Struttura meccanica
Aste di supporto

Figure 1.7: Perspective view of the RFX experiment

The main components of the RFX device are the following:

- **Vacuum vessel**
  The vacuum vessel is a plasma containment chamber made of Inconel 625, a nickel-chromium-molybdenum alloy with high temperature tolerance and robust mechanical properties. Its structure consists of 72 elements, each covering a 5 degrees arc, creating a complete toroidal assembly. The inside wall which in direct contact with the plasma is covered with 2016 graphite tiles, which shield the vessel from sudden heat impulses generated by the plasma. For the plasma control and diagnostic requirements the vessel contains 96 portholes with access to the plasma.

- **Stabilizing shell**
  The vacum vessel is wrapped with a 3 mm copper shell, that serves as a passive stabilizer with a time constant of 50 ms, and a stainless steel mechanical support structure. The shell is made of 4 main components, it presents a cut running along the entire equatorial region and two gaps left unwelded in the poloidal direction that allow the toroidal magnetic and electric fields to penetrate inside the shell.

- **Active control system**
  Active control system is the main feature of the RFX experiment. The shell is able to contribute to the horizontal equilibrium over the initial pulse by stabilizing rapidly growing perturbations,

but on longer time scales the active feedback system is needed to control the various MHD instabilities. The system contains 4x48 saddle coils, 4 toroidal and 48 poloidal arrays, organized in a bi-dimensional mesh that covers the whole outer surface shell. A pair of sensors is associated with each radial field coil. Pick up coil, detecting both toroidal and poloidal components, and a saddle probe measuring the average radial field. Each saddle coil is fed independently through its own power supply and controlled by a set of dedicated CPUs, capable of processing all of the inputs and generating all of the outputs at a rate of up to 5 kHz [6][7].

1.4.1 Diagnostics

One of the peculiarities of the RFX-mod machine is the rich number of diagnostics: 8336 signals nodes are acquired by sensors and probes, producing a high amount of data in short time ($\approx$ 40 GB/s) [8]. Signals can be classified by the nature of the diagnostics: signals from magnetic probes, Soft X-Ray detectors, light sensors such as photodiodes and photomultipliers, from Langmuir probes, etc.

The diagnostics can be divided into 4 main categories:

- Magnetic diagnostics, plasma current, loop potentials, etc.
- Spectroscopic diagnostics, which measure the spontaneous e.m. radiaton emitted by the plasma, ranging from microwave (electron cyclotron emission, ECE) to X-Ray.
- Reflectometers and interferometers, which are based on the modification on the phase and polarization of a microwave coherent beam or optical laser, once they pass through the plasma.
- Thomson scattering for the measurement of the electron temperature from the diffusion of an intense visible or IR pulsed laser beam.

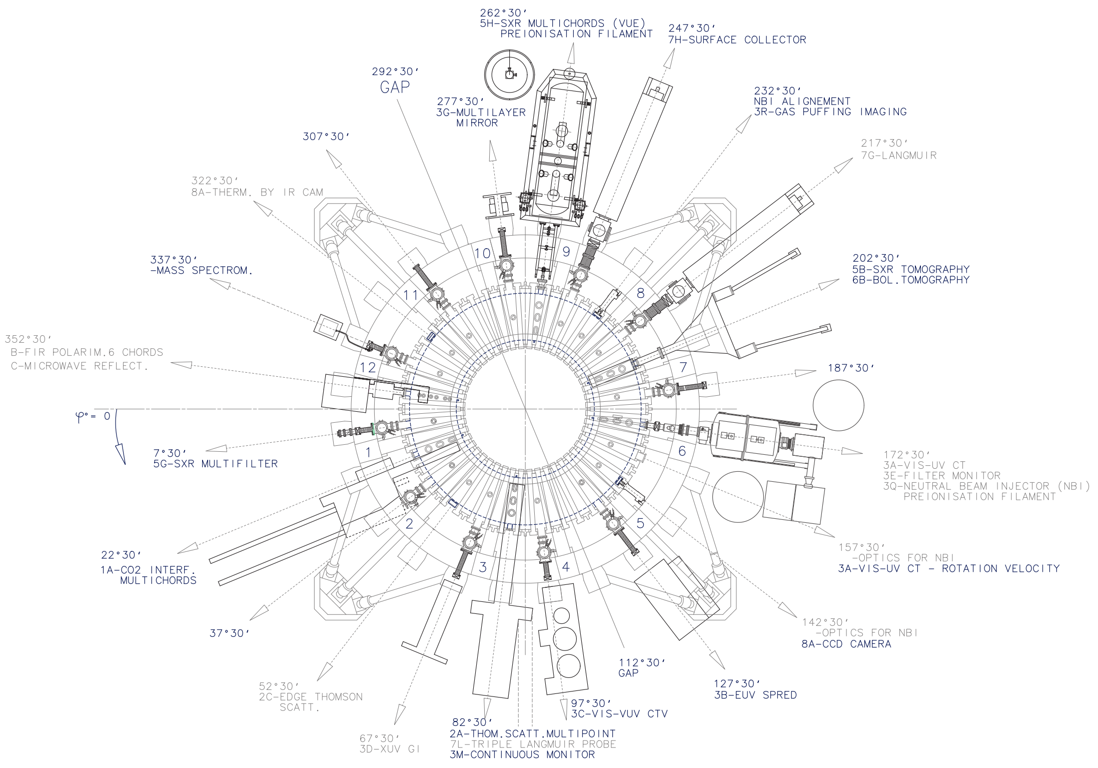

Figure 1.8: RFX map of diagnostics

Chapter 2

Data compression

The RFX experiment, located at Consorzio RFX in Padua, provides a very high spatial resolution magnetic diagnostic with more than 1700 sensors, along with more than 200 actuator coils independently controlled. However the downside of having a remarkable full 3D plasma shape reconstruction and control capability is the amount of data produced by the sensors. A solution is needed for efficient data transfer and storage, and for that, we take a look at time series compression techniques.

2.1 Entropy

The outcome of a random variable X is inherently uncertain. This uncertainty can be viewed in two complementary ways: as the quantity of uncertainty we have before the outcome is revealed, or as the knowledge we gain once the outcome becomes known.

To formally describe information in probabilistic systems, we start from the concept of *probability space* $S = (\Omega, \mathcal{F}, P)$.

- Ω: sample space containing all possible outcomes;
- $\mathcal{F}$: σ-algebra of events, meaning $\mathcal{F}$ is a family of subsets of Ω closed under the operations of union, intersection and complement (en event $E \in \mathcal{F}$ is a collection of outcomes);
- $P : \mathcal{F} \to [0, 1]$ probability measure.

The degree of information each event provides about the outcome can be quantified by a function $I : \mathcal{F} \to \mathbb{R}^+$, that satisfies the following considerations:

1. Events with the same number of outcomes provide the same amount of information.

$\rightarrow$ I is a function of the probability of the event $I(E \in \mathcal{F}) = f(P(E))$

2. Events with more outcomes provide less information than events with fewer outcomes.

   $\rightarrow$ f is a monotonically decreasing function of P

3. For independent events E_1 and E_2, $I(E) = I(E_1) + I(E_2)$

   $\rightarrow$ f must have the property $f(x \cdot y) = f(x) + f(y)$

The logarithmic function satisfies these constraints. It measures the information we obtain when we get to know that the outcome of a random process is among a given subset of all possible outcomes, specified by the event E

$$I(E) = -\alpha \log(P(E)) \tag{2.1}$$

where α corresponds to the choice of the logarithm's base which determines the unit of measurement of information. In information theory, the base is often chosen as 2, defining the unit **bit** (binary digit). It is the information revealed when we specify the outcome of a random process with two equiprobable outcomes.

If the events used to specify the outcome are not independent, the information provided by their joint occurrence is less than the sum of their individual contributions. This reflects the redundancy inherent in dependent events.

Let's consider a random variable $X : \Omega \rightarrow \{x_1, ..., x_N\} \in \mathbb{R}$ in a discrete probability space Ω. For each outcome $a \in \Omega$ there is a corresponding event $E = \{a\} \in \mathcal{F}$. If $X : a \mapsto x_i$, Then X has the probability mass function $p_X(x_i) = P(a)$. The **Shannon entropy** of X is given by

$$H[X] = -\sum_{i=1}^{N} p_X(x_i) \log p_X(x_i) \tag{2.2}$$

which is the *average information needed to specify the outcome of the random variable.*

Some properties of the Shannon entropy are:

- $H[X]$ is zero only if one outcome has probability 1, otherwise it is positive
- $H[X]$ is maximized for uniform probability function $p_X(i) = 1/N \; \forall i$
- $H[X]$ is concave with respect to p, implying that mixing two probability distributions increase the entropy

Let X, Y be two random variables with values $\{x_i\}, \ \{y_j\}$

- The *joint entropy* of X and Y, is given by the Shannon entropy of the joint distribution $p_{X,Y}(x,y)$.
$$H[X,Y] = H\left(p_{X,Y}\right) = -\sum_{i,j} p_{i,j} \log p_{i,j} \tag{2.3}$$
where $p_{i,j} \equiv p_{X,Y}(x_i, y_j)$
- *Conditional entropy* of Y given X is defined as
$$H[Y \mid X] = \sum_i p_i H\left[Y \mid x_i\right] = -\sum_i p_{\mathrm{i}} \sum_j \frac{p_{i,j}}{p_i} \log \frac{p_{i,j}}{p_i} \tag{2.4}$$
- *Mutual information* between X and Y is the information that X provides about Y, and vice versa.
$$.I[X:Y] = H[X] + H[Y] - H[X,Y] = H[Y] - H[Y \mid X] = H[X] - H[X \mid Y] \tag{2.5}$$

Kullback-Leibler divergence, or relative entropy, is a measure of discriminability between two distributions. Given two random variables X_1, X_2 with values in the same space $\{x_i\}_{\ i=1,\ldots,N}$ the KL divergence is defined as:

$$D_{KL}\left[X_1 \| X_2\right] = D_{KL}(p\|q) = \sum_i p_i \log \frac{p_i}{q_i} \tag{2.6}$$

where $p_i \equiv p_{X_1}(x_i)$ and $q_i \equiv p_{X_2}(x_i)$.

$$\begin{gathered} D_{KL}\left[p\|q\right] \geq 0 \\ D_{KL}\left[p\|q\right] = 0 \quad \text{if and only if} \quad p = q \end{gathered} \tag{2.7}$$

The mutual information can be written as the KL divergence between the joint distribution and the product of the marginals.

$$\begin{gathered} I[X:Y] = D_{KL}\left[p_{i,j} \| p_i q_j\right] \geq 0 \\ I[X:Y] = D_{KL}\left[p_{i,j} \| p_i q_j\right] = 0 \quad \text{if and only if } X \text{ and } Y \text{ are statistically independent} \end{gathered} \tag{2.8}$$

2.2 Information compression

The principels of information theory revolve around the efficient and reliable transfer of messages between a sender and a receiver through a communication channel. To send a message the sender must first convert the message into a suitable form that can be sent through the channel, this step is the *encoding*. The receiver then interprets this encoded message to reconstruct the original content, a process called *decoding*. The encoding process needs to be executed in a way that guarantees accurate reconstruction of the original message, with minimal information loss, while compressing the message as much as possible.

Let's assume that the message to be transmitted x is selected with a certain probability from a finite set of messages $\mathcal{A}_X = x_i, ..., x_n$. X is a random variable with values in $\mathcal{A}_X$ and probability mass function $p_X(x)$. The encoding of $X \in \mathcal{A}_X$ into a binary string of length K is given by

$$E : \mathcal{A}_X \to B^K, \quad E : x \mapsto y \tag{2.9}$$

where $B^K = \{0, 1\}^K$ is the set of all possible binary strings of length K.

The decoding, as might be expected, is the inverse of the encoding

$$D = E^{-1} : B^K \to \mathcal{A}_X, \quad D : y \mapsto x \tag{2.10}$$

Consequently E needs to be invertible, mapping each different $x \in \mathcal{A}_X$ into a different $y \in B^K$. Therefore the number of digits K needed to encode a message must satisfy the following condition:

$$K \geq H_0 \equiv \log |\mathcal{A}_X| \tag{2.11}$$

H_0 is called *raw bit content* of X.

There are two fundamental approaches to achieving further information compression:

- **Lossy Compression**: This method accepts some loss of information by allowing two distinct inputs to be mapped to the same output, or by discarding some inputs entirely. However, this must occur with low probability to maintain acceptable reconstruction.

- **Lossless Compression**: This approach ensures the original data can be perfectly reconstructed by using variable-length codes. Inputs are encoded into binary strings of variable length l_x, with more probable inputs mapped to shorter strings and less probable inputs to longer ones.

$$\langle l_x \rangle = \sum_{x \in \mathcal{A}_X} p_X(x) l_x < H_0 \tag{2.12}$$

2.2.1 Lossy coding

Let's consider the smallest subset of outcomes such that its overall probability is $> 1 - \delta$, δ sufficient subset.

$$S_\delta[X] = \text{argmin}_{|S|} \left(S \subset A_X \mid P(X \in S) > 1 - \delta\right) \tag{2.13}$$

This is the set of relevant messages. If we torlerate a probability error δ, the number of digits required to encode X is given by the raw bit content $H_\delta[X]$

$$H_\delta[X] = \log |\mathcal{S}_\delta[X]| \tag{2.14}$$

For a long sequences $X^{(n)}$, $n \to \infty$

$$H_\delta(X^{(n)}) \to nH(X) \quad \forall\delta \tag{2.15}$$

meaning we can describe the outcome with $nH(X)$ binary digits with vanishingly small error probability.

Shannon's **source coding theorem**: For any arbitrarily small β, and for any arbitrarily small δ, it is possible to achieve lossy compression of $X(n)$ with only $n(H+\beta)$ bits with an error probability δ for large enough n.
For large enough n, it is impossible to achieve lossy compression of $X(n)$ with only $n(H-\beta)$ bits with an error probability δ, for any δ, even arbitrarily close to 1.

$$\forall\delta > 0, \forall\beta > 0, \exists n_0 \quad \text{such that} \quad \forall n > n_0, \left| H_\delta\left(X^{(n)}\right) - nH \right| < n\beta \tag{2.16}$$

Shannon's theorem result is based on the existence of *typical sequences*.

$$T_{n\beta} = \left\{ x^{(n)} \in A_X^n | \, | -\frac{1}{n} \log P\left(x^{(n)}\right) - H[X] \right| < \beta \right\} \tag{2.17}$$

The probability that a random string is typical grows with n.

$$P(X^{(n)} \in T_{n\beta}) \xrightarrow[n\to\infty]{} 1 \tag{2.18}$$

2.2.2 Lossless coding

Lossless coding ensures perfect reconstruction of the original code. This is done by *symbol coding*, encoding messages into binary strings of variable length.

The encoding is defined as:

$$\begin{gathered} E : \mathcal{A}_X \to B^+ \\ B^+ = B \cup B^2 \cup B^3 \cup ... \end{gathered} \tag{2.19}$$

B^+ is the set of all possible binary strings with variable length.

To guaranty lossless coding the symbol codes need to be easy to decode, uniquely decodable, and as short as possible.

The *average length* of a code is:

$$L(E,X) = \sum_i p_i l_i \tag{2.20}$$

Kraft inequality expresses a constraint on the codeword length posed by the request of unique decodability.

$$\sum_i 2^{-l_i} \leq 1 \tag{2.21}$$

Source coding theorem for symbol codes: For a random variable $X \in \mathcal{A}_X$ there exists a symbol code E such that:

$$H[X] \leq L(E,X) \leq H[X] + 1 \tag{2.22}$$

The *Huffmann* algorithm can be used to find optimal symbol codes. It is a recursive procedure that constructs an optimal binary tree starting from the leaves. At each iteration, it creates a parent node having the two least probable nodes as children.

2.2.3 Rate-Distortion theory

A **distortion** measure $d(x,y)$ quantifies the information loss introduced by a lossy compression process. It is defined as a measure of the difference between the original data x and the reconstruction y. An example of a distortion measure is the mean squared error function $d(x,y) = (x-y)^2$.

The average distortion D is give by:

$$D = \mathbb{E}[d(X,Y)] = \sum_{x \in X} \sum_{y \in Y} P_{X,Y}(x,y) d(x,y) \tag{2.23}$$

where $P_{X,Y}(x,y)$ is the joint probability distribution of X and Y.

The **Rate-Distortion theory** describes the trade-off between the number of bits per symbol required to encode the data, called **rate**, and the distortion introduced by the compression process. Small distortion requires higher rate, lower distortion allows for lower rate.

For a given maximum average distortion D^*, the rate distortion function $R(D^*)$ is the lower bound for the transmission bit-rate.

$$R(D^*) = \min_{D \leq D^*} \{I(X;Y)\} \tag{2.24}$$

$I(X;Y)$ is the mutual information between the source X and the reconstruction Y. It measures how much information about X is retained in the reconstruction Y.

2.3 Time series compression algorithms

Time series is a sequence of data points collected over an interval of time, sorted in ascending order according to the timestamp associated with each element.

There are two categories of time series:

- **Univariate Time Series (UTS)**: sequence of observations of a single variable recorded over time.
- **Multivariate Time Series (MTS)**: sequence of observations of multiple variables recorded simultaneously over time. In MTS an array is associated with each timestemp, containing the values of the variables at that timestemp.

The notation used for time series is:

$$TS = [(t_1, x_1),\ (t_2, x_2), ...,\ (t_n, x_n)], \quad x_i \in \mathbb{R}^m \tag{2.25}$$

where n is the number of elements in the time series, and m is the number of variables in the multivariate time series, corresponding to its dimensionality. For univariate time series $m = 1$.

Time series can be divided into segments, called *time windows* while preserving the order and all the elements.

$$\begin{gathered} TS_{[i,j]} = [(t_i, x_i),\ (t_{i+1}, x_{i+1}), ...,\ (t_j, x_j)] \\ TS[k] = TS_{[i,j]}[k - i + 1] \quad \forall k \in [i, j] \end{gathered} \tag{2.26}$$

A good compression algorithm should take advantage of some of the peculiarities of time series data produced by sensors, such as data redundancy, predictability, and approximability of sensor response by functions.

Compression algorithms can be described with a combination of the following classes:

- *Non-adaptive - adaptive*: A non-adaptive algorithm, contrary to adaptive, has fixed parameters and doesn't need a training phase to adjust to different types of data.
- *Lossy-Lossless*: Lossless algorithm ensures perfect reconstruction of the original data, while lossy allows for small reconstruction errors.
- *Symmetric - non-symmetric*: An algorithm in which the decoding process is the inverse of the encoding operations is called symmetric. A non-symmetric algorithm uses different operations in the encoding and decoding.

2.3.1 Dictionary-Based algorithms

Dictionary-based compression algorithms rely on the observation that time series data often contain recurring patterns or segments [15]. These segments can be extracted from the data and collected into a dictionary that has a univocal key for each entry. The choice of the key length is a trade off between low decompression error and high compression factor. During the compression phase, each segment of the time series is replaced by its key. If some segments are not present in the dictionary, they are either added as a new entry or kept uncompressed. Dictionary based compression has both lossy and lossless implementation.

The main challenges of this approach are:

- *Efficient Search in the Dictionary*: The algorithm needs to minimize the search time for matching segments in the dictionary.
- *Generalizing Dictionary Entries*: The entries in the dictionary should be as general as possible to reduce the number of unmatched segments. This implies optimizing the dictionary entries selection process to minimize the distance between the original segment and their dictionary representation.

2.3.2 Function Approximation

The function approximation compression technique is based on the principle that time series segments can be represented as time dependent functions. Since exploring all possible functions is computationally not feasible, implementations typically focus on a specific family of functions. The process involves dividing the time series into segments of fixed or variable length and finding parameters that better approximate each segment. This approach is inherently lossy.

A significant advantage of this method is the absence of a training phase since the regression algorithm processes each segment independently.

Some techniques used in Function Approximation are:

- Piecewise Polynomial Approximation
- Chebyshev Polynomial Transform
- Discrete Wavelet Transform
- Discrete Fourier Transform

2.3.3 Sequential algorithms

Sequential compression algorithms operate by combining sequentially multiple simple compression techniques. Each one further compressing the output of the previous one.

Some commonly used techniques are:

- **Huffman coding**: A lossless data compression algorithm that assigns shorter binary codes to more frequent symbols and longer codes to less frequent ones.
- **Delta encoding**: Each element x_t at time t is encoded as the difference $\Delta(x_t - x_{t-1})$.
- **Run-length encoding**: Replaces consecutive repetitions of the same value with the pair (value at time t, number of consecutive occurrences).
- **Fibonacci binary encoding**:

$$F_E(N) = \sum_{i=0}^{j} a_i \cdot F(i) \tag{2.27}$$

where $a_i \in \{0, 1\}$ is the i-th bit of the binary representation of $F(N) = a_1...a_n$

$$F(N) = \begin{cases} F(N-1) + F(N-2) & \text{if } N \geq 1 \\ N & \text{if } N = -1 \text{ or } N = 0 \end{cases} \tag{2.28}$$

2.3.4 Autoencoders

Autoencoders are neural networks designed to learn compressed latent representations of the input data. They are composed of two symmetric parts, encoder and decoder. The encoder learns to identify and extract the most relevant features of the input and represent them with codes in the latent space, while the decoder reconstructs the original input starting from the latent representation. The algorithm is trained to minimize the difference between the original input and its reconstruction using an adequate loss function.

Conceptually, an autoencoder can be thought of as a compression algorithm that reduces the dimensionality of data by learning an encoded representation [16]. The encoder compresses the input and the decoder attempts to recreate the input from this compressed representation.
Autoencoders are distinguished from traditional compression techniques in several ways:

- **Data-specific:** Autoencoders are trained specifically on a given set of data. Unlike generic compression algorithms like JPEG or MP3, the parameters of an autoencoder are optimized to best compress and decompress data similar to what they have been trained on.

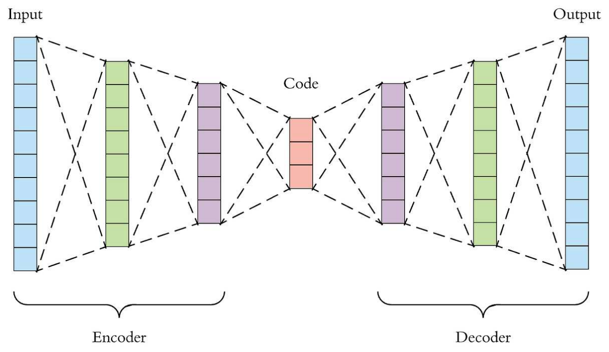

Figure 2.1: Autoencoder architecture, the topological symmetry is characterized by an internal notch where the reduced latent space dimension of the code (in red) implement the actual compression of the input feature. On the opposite end the decoder network, simultaneously trained, guarantees the best lossy recovery of the signal from the encoded feature.

- **Lossy compression:** The process of encoding and decoding generally leads to a loss of information, making autoencoders a form of lossy compression. The extent of data loss depends on the capacity of the encoder and decoder and the complexity of the data being compressed.
- **Non-linear transformations:** Autoencoders can learn non-linear transformations, which can potentially lead to more efficient compression compared to linear methods such as PCA.
- **Flexibility in architecture:** The architecture of an autoencoder can be easily adapted to fit with specific types of data (e.g., convolutional layers for image data), which can enhance the compression efficiency.

Autoencoders can also be the best candidate to be applied in real-time data compression pipeline. By training on a dataset representative of the real-time data, an autoencoder can quickly encode incoming data into a lower-dimensional space and decode it when needed. This real-time capability is beneficial for applications requiring immediate processing, such as providing the input for the reactor feedback.
Moreover, the latent space of an autoencoder can be aligned with meaningful physical quantities or parameters. This alignment can be achieved through techniques such as:

- **Supervised dimensionality reduction:** By incorporating label information during training, the latent dimensions can be encouraged to align with known attributes or classes.
- **Regularization:** Adding regularization terms (e.g., sparsity of the representation) can help in learning more interpretable features.
- **Domain-specific architectures:** Designing the encoder and decoder to reflect physical or

functional constraints of the data can lead to latent representations that are more aligned with physical quantities.

By leveraging these techniques, autoencoders not only serve as effective compression tools but also facilitate the extraction of meaningful insights from large and complex datasets, making them particularly useful in our application.

2.3.5 Transformers

A transformer is another deep learning architecture that can be used effectively for the implementation of a compression algorithm. The first idea came from a Google research on large language models mainly focused on text translation and based on the multi-head attention mechanism, proposed in 2017 [17]. Transformers are the main methodology on which our compression mechanism is based and a more detailed description of the foundation principle will be presented in the following chapters.

Transformers and autoencoders share the concept of learning representations from input data. Both architectures are designed to encode input data into a compressed representation and potentially decode it back or make predictions based on it. However transformers can be particularly advantageous over autoencoders for time series analysis because the self-attention mechanism allows them to weigh the importance of different points in the time series regardless of their position. This is particularly useful in time series where the relevance of historical data can vary significantly and is not necessarily dependent on proximity. In addition, unlike recurrent models, Transformers process all data points simultaneously, which allows for faster training times as computations can be parallelized effectively. This is beneficial for time series analysis, especially when dealing with large datasets.

Both Transformers and autoencoders can learn data representations, however, Transformers offer advantages in terms of handling long-range dependencies, parallel computation, and adaptability to different tasks, making them often a better choice for complex time series analysis and the method applied in this study.

Chapter 3

Machine Learning

Machine learning transformed the traditional rule-based approach in computational modeling by allowing machines to learn patterns directly from data. This practice diverges from the classical scientific methods of hypothesis formulation and construction of a model around it. It uses data-driven techniques to build representations without predefined models. It is a powerful concept for handling complex systems with nonlinear interactions and intricate dependencies. Machine learning models can generalize from training data to unseen scenarios, adapting to diverse domains without requiring extensive prior knowledge.

3.1 Neural Networks

Neural networks (NN) are a class of machine learning models inspired by biological neural systems. The first milestone in this field was the introduction of the *Perceptron* by Frank Rosenblatt in 1958, a simple mathematical model for binary classification $f : \mathbb{R}^d \to \{-1, +1\}$. It learns a linear decision boundary that separates the input space into two halfspaces.

Decision function:

$$y = f(x) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + \mathbf{b}) \tag{3.1}$$

where $\mathbf{w}$ is a vector of learnable parameters (**weights**), and $\mathbf{b}$ is scalar offset (**bias**).

Given a dataset of labeled samples $\textsc{d} = \{(x_1, t_1), (x_2, t_2), ..., (x_N, t_N)\}$, the weights of the model are adjusted at each iteration in order to improve the classification on $\textsc{d}$. For a correct classification the following is true

$$\langle \mathbf{w}^{(i)}, \mathbf{x}_n \rangle t_n > 0 \tag{3.2}$$

The training process can be summarized with the updated rule:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{x}_n t_n \tag{3.3}$$

It involves maximizing the quantity $\langle \mathbf{w}, \mathbf{x}_n \rangle t_n$ for each misclassified sample. The step is repeated until all samples are correctly classified.

The drawback of the Perceptron is that real data is rarely linearly separable. This can be mediated by using feature mapping to transform the input into a higher dimensional space where the data is linearly separable. However, finding a suitable mapping is not an easy task. The solution to this problem can be found in **Neural networks** that will *learn* the feature mapping.

Neural networks are composed of **neurons** that build up on the idea of the Perceptron by switching the sign function with a non-linear activation f.

$$y_j = f(\sum_{i=1}^{d} w_{i,j} x_i + b_j) \tag{3.4}$$

A layer is constructed by stacking many neurons together. Interconnecting the neurons of multiple layers, in a way that each neuron is connected to all the neurons in the previous layer and all the neurons in the next layer, creates a **feed-forward neural network**. The first layer of a network is called *input layer*, it receives the input data that is then fed to the first in a series of *hidden layers*. The last layer is the *output layer*, which produces the output of the network. When a network has many hidden layers, we talk about **deep learning**.

The procedure for generating an output $\mathbf{y}$ for a certain input $\mathbf{x}$ is called **forward pass** and it works as follows:

1. Feed $\mathbf{x}$ to the first layer

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{o}(1) = f(\mathbf{a}^{(1)}) \tag{3.5}$$

2. Use the output of the first layer $\mathbf{o}(1)$ as input for the second layer

3. Repeat for all hidden layers

4. Get output vector $\mathbf{y}$

$$\mathbf{a}^{(L)} = \mathbf{W}^{(L)}\mathbf{o}^{(L-1)} + \mathbf{b}^{(L)}, \quad \mathbf{y} = g(\mathbf{a}^{(L)}) \tag{3.6}$$

$g()$ is the output function that is chosen in according to the learning problem.

3.1.1 Activation functions

To learn complex patterns in data, it is crucial to introduce non-linearity in the network. That is why activation functions are s key component of a network.

The selection of an activation function is based on the architecture of the network, but there aren't any definitive rules. For the inner layers, the *Rectified linear unit (ReLU)* and its variants (*Leaky ReLU, Exponential-linear unit (ELU)*, etc.) are the most commonly used activation fucntions.

$$ReLU(z) = max(0, z) \tag{3.7}$$

Contrary to the hidden layers, the activation of the neurons in the output layer must take into consideration the task of the model. For example in binary classification, the most suitable activation function is the sigmoid. When using the sigmoid function, the output of the network can be interpreted as the probability of the sample belonging to each class. In multi-class classification, the softmax function is used to obtain a probability distribution over the classes. For regression problems the output can be any real number, therefore the identity function is the most appropriate.

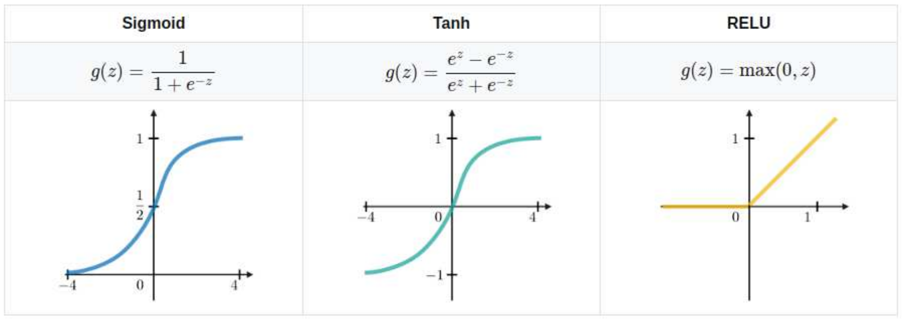

Figure 3.1: Some of the most commonly used activation functions

3.1.2 Network training

Training a neural network involves optimizing its parameters to learn the task in hand. There are two main paradigms in network training:

- **Supervised learning**: The network is trained using labeled data, with the goal of minimizing the discrepancy between the labels predicted by the network and the true labels of the data.
- **Unsupervised learning** involves training unlabeled data to learn underlying patterns and hidden insights within the data. Examples of unsupervised learning are clustering, dimensionality reduction, anomaly detection, etc.

The optimization objective of the network training is expressed by the **loss function**. The loss function provides a measure of how well the network performs on a given task, hence it should reflect the learning problem. Some of the most commonly used loss functions are the *Mean squared error* $\mathcal{L}(\mathcal{D}, \mathcal{W}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{t}_n\|^2$, which measures the squared distance between predicted and true labels; *Binary cross entropy* used for binary classification $\mathcal{L}(\mathcal{D}, \mathcal{W}) = -\sum_{n=1}^{N} [t_n \log y_n + (1 - t_n) \log(1 - y_n)]$, etc.

The training of a neural network consists in finding a set of weights $\mathcal{W}$ that minimize the loss function $\mathcal{L}(\mathcal{D}, \mathcal{W})$ on a given dataset $\mathcal{D}$. This is an optimization problem and the general approach is to iterate over the data and at each step update the weight in according to the **gradient descent (GD)** algorithm.

$$\left(w_{i,j}^{(\ell)}\right)^{n+1} = \left(w_{i,j}^{(\ell)}\right)^{n} - \eta \nabla_{w_{i,j}^{(\ell)}} \mathcal{L}(\mathcal{W}, \mathbf{x}) \tag{3.8}$$

η is the *learning rate*, a hyperparameter that specifies the size of the step taken in the direction of the gradient.

Backpropagation is an efficient way to compute the gradients of the loss function with respect to the network parameters. The idea is to repeatedly use the *chain rule* of derivative, which makes computation efficient and recursively implementable.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{i,j}^{(\ell)}} &= \frac{\partial \mathcal{L}}{\partial a_j^{(\ell)}} \frac{\partial a_j^{(\ell)}}{\partial w_{i,j}^{(\ell)}} = \frac{\partial \mathcal{L}}{a_j^{(\ell)}} o_i^{(\ell-1)} \\ \frac{\partial \mathcal{L}}{\partial b_j^{(\ell)}} &= \frac{\partial \mathcal{L}}{\partial a_j^{(\ell)}} \frac{\partial a_j^{(\ell)}}{\partial b_j^{(\ell)}} = \frac{\partial \mathcal{L}}{a_j^{(\ell)}} \cdot 1 \end{aligned} \tag{3.9}$$

The first necessary step for backpropagation is to perform a **forward pass** to obtain $\mathbf{o}^{(l)}$, $\mathbf{a}^{(l)}$ $\forall l$ and $\mathbf{y}$. The results of the forward pass are than used to backpropagated the gradients from the last layer to the first.

The gradient of the loss with respect to a weight is given by the product of a *error message* δ at that weight and the output of the parent neuron.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{i,j}^{(\ell)}} &= \delta_j^{(\ell)} o_i^{(\ell-1)} \\ \delta_j^{(\ell)} \triangleq \frac{\partial \mathcal{L}}{\partial a_j^{(\ell)}} &= \sum_k \frac{\partial \mathcal{L}}{\partial a_k^{(\ell+1)}} \frac{\partial a_k^{(\ell+1)}}{\partial a_j^{(\ell)}} \end{aligned} \tag{3.10}$$

By applying again the chain rule,

$$\frac{\partial a_k^{\ell+1}}{\partial a_j^\ell} = \frac{\partial a_k^{\ell+1}}{\partial o_j^\ell}\frac{\partial o_j^\ell}{\partial a_j^\ell} = w_{j,k}^{(\ell+1)}\frac{\partial f}{\partial a_j^\ell} = f'\left(a_j^\ell\right) \tag{3.11}$$

we arrive to the final recursive rule

$$\delta_j^{(\ell)} = f'\left(a_j^\ell\right)\sum_k \delta_k^{(\ell+1)} w_{j,k}^{(\ell+1)} \tag{3.12}$$

The error message is updated in the backward direction starting from the output layer for which the error message is given by:

$$\delta_m^{(L)} = \frac{\partial \mathcal{L}}{\partial a_m^{(L)}} = \frac{\partial \mathcal{L}}{\partial y_m}\frac{\partial y_m}{\partial a_m^{(L)}} = \frac{\partial \mathcal{L}}{\partial y_m} g'\left(a_m^{(L)}\right) \tag{3.13}$$

3.1.3 Advanced optimization methods

Iterating over large datasets can be slow, especially if the dataset does not fit in memory. To address this issue, **stochastic gradient descent (SGD)** can be applied. Instead of using the exact gradient of the loss to perform the update step, SGD uses a quantity that on average has the same direction. The gradient is computed on a single data point picked at random. This results in a much faster algorithm but with the consequence of having unstable trajectory towards the minimum. Having noisy updates could be useful for escaping local minima and saddle points.

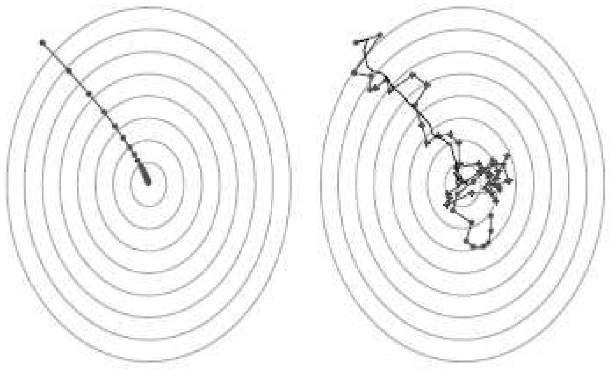

Figure 3.2: Comparison of the GD algorithm on the left and SGD on the right

Minibatch implementation is the compromise between the stability of GD and the speed of SGD. It computes the gradient over a minibatch of training data samples. The size of the batches is an important hyperparameter that can influence the optimization process.

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \frac{\eta^{(i)}}{K} \sum_{k=1}^{K} \nabla \mathcal{L} \left(\mathbf{x}_k, t_k; \mathbf{w}^{(i)} \right) \tag{3.14}$$

Momentum $\mathbf{v}$ can be incorporated into the optimization algorithm to give it robustness to brief changes in the update direction. The concept involves smoothing out the noisy trajectory by prioritizing directions that consistently appeared in previous updates.

$$\begin{aligned} \mathbf{v}^{(i+1)} &= \alpha \mathbf{v}^{(i)} - \frac{\eta^{(i)}}{K} \sum_{k=1}^{K} \nabla \mathcal{L} \left(\mathbf{x}_k, t_k; \mathbf{w}^{(i)} \right) \\ \mathbf{w}^{(i+1)} &= \mathbf{w}^{(i)} + \mathbf{v}^{(i+1)} \end{aligned} \tag{3.15}$$

The α parameter controls the strength of the momentum. Larger α, with respect to the learning rate η, puts more weight on the previous gradients. Long-term trends in the gradient direction lead to acceleration of the optimization process. Having momentum is also useful for easy escape from local minima.

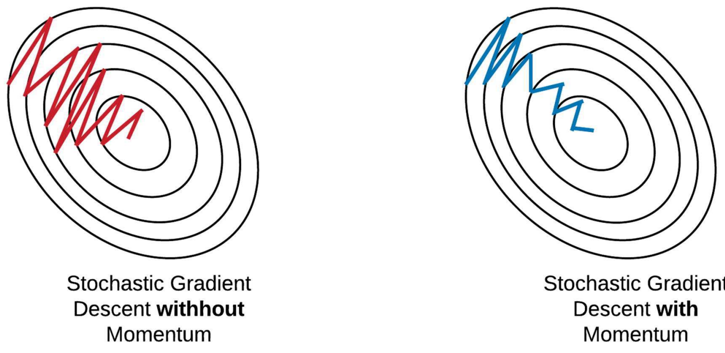
Stochastic Gradient Descent **without** Momentum
Stochastic Gradient Descent **with** Momentum

Figure 3.3: The effect of momentum on the stochastic gradient descent algorithm

Optimization algorithms are a well researched area in the machine learning field. Throughout the years numerous advanced optimizers have been developed. Some of the most widely used are:

- **Adaptive Gradient (AdaGrad)** [21] which introduced the idea of *adaptive learning rate*, as some weights may require slower or faster updates than others. In AdaGrad individual learning rate η_m is assigned to each weight. The update rule for the m-th weight is:

$$w_m^{(i+1)} = w_m^{(i)} + \eta_m^{(i)} \frac{1}{K} \sum_{k=1}^{K} \nabla \mathcal{L} \left(x_k, t_k; w_m^{(i)} \right) \tag{3.16}$$

with weight specific learning rate computed starting from the initial learning rate η

$$\eta_m^{(i)} = \frac{\eta}{\sqrt{\sum_{j=1}^{i}[\frac{1}{K}\sum_{k=1}^{K}\nabla\mathcal{L}\left(x_k, t_k; w_m^{(j)}\right)]^2 + \epsilon}} \tag{3.17}$$

where ϵ is a small constant used for avoiding division by zero.

AdaGrad enables exploration of dimensions that are rarely updated. However, its drawback is that after several updates, the learning rate eventually approaches zero, causing learning to stop.

- **Adaptive moment estimation (Adam)** [22] outperforms most of the other optimizers on a huge variety of tasks and for that reason is usually the default choice for an optimizer. One iteration of Adam can be summarized in 3 steps:

  1. Estimation of first and second moment of the gradients

  $$\begin{aligned} \xi_m^{(i)} &= \beta_1 \xi_m^{(i-1)} + (1-\beta_1)\, g_m^{(i)} \\ \psi_m^{(i)} &= \beta_2 \psi_m^{(i-1)} + (1-\beta_2)\left(g_m^{(i)}\right)^2 \end{aligned} \tag{3.18}$$

  where $\mathbf{g}^{(i)}$ is the gradient on a mini-batch at iteration i

  $$\mathbf{g}^{(i)} = \frac{1}{K}\sum_{k=1}^{K}\nabla\mathcal{L}\left(x_k, t_k; \mathbf{w}^{(i)}\right) \tag{3.19}$$

  2. Bias correction

  $$\begin{aligned} \hat{\xi}_m^{(i)} &= \xi_m^{(i)} / \left(1-\beta_1^i\right) \\ \hat{\psi}_m^{(i)} &= \psi_m^{(i)} / \left(1-\beta_2^i\right) \end{aligned} \tag{3.20}$$

  3. Weights update

  $$w_m^{(i)} = w_m^{(i-1)} + \eta\frac{\hat{\xi}_m^{(i)}}{\sqrt{\hat{\psi}_m^{(i)} + \varepsilon}} \tag{3.21}$$

  β_1 and β_2 are the moving average weights, hyperparameters of the Adam algorithm with default values $\beta_1 = 0.9,\ \beta_2$=0.999.

3.1.4 Overfitting

Overfitting is a fundamental concept in network training. It is defined as "learning a model that corresponds too closely (or exactly) to a specific dataset, and does not generalize well when faced with new data". What happens is that the model memorizes noise patterns present in the training data, rather than learning meaningful features that can be generalized to new data.

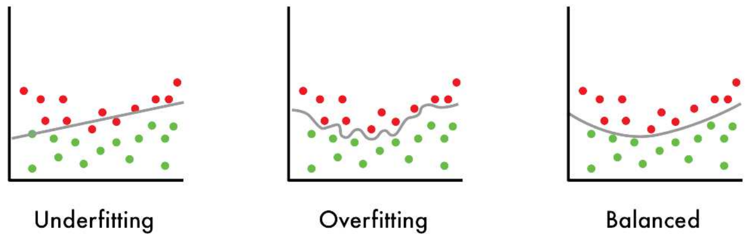

Figure 3.4: Example of overfitting and underfitting in a classification task

Avoiding overfitting is crucial for effective learning, which is why there are several methods that can control overfitting.

- **Early stopping**

  It is important to properly monitor the training process for signs of overfitting. A common way to do this is by using validation data that is completely disjoint from the training data.

  Early stopping consists in interrupting the training before the model enters the overfitting region, meaning as soon as the validation loss stops decreasing.

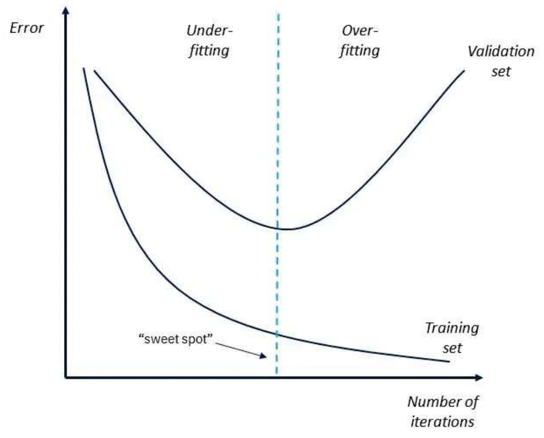

Figure 3.5: Monitoring the training process for overfitting

- **Regularization**

  Regularization is used to stabilize the model. Overfitted models are highly sensitive to small changes in the training data, resulting in large changes in the output function. Good models are rather insensitive to small modifications in the input data. They learn smooth and simple functions that capture the core features of the data, rather than small changes due to noise and outliers.

Model stability can be achieved by adding a regularizer to the loss function. The regularizer is a function of the model's parameters that measures the model's complexity. Its purpose is to impose some constraints on the model parameters so that the it does not become overly unstable or excessively tailored to the training data.

The most important regularization techniques are L1 and L2 regularization.

L1 regularization is used to sparsify the parameters, it encourages the model to use a small subset of the features, by setting some weights to zero.

$$\mathcal{R}(\mathcal{W}) = \lambda \sum_{w_i \in \mathcal{W}} |w_i| \tag{3.22}$$

L2 regularization is also known as weight decay. Weights associated with small gradients, which are the ones that are rarely updated, will progressively decay at each iteration. This results in a reduction of unimportant weights.

$$\mathcal{R}(\mathcal{W}) = \lambda (\sum_{w_i \in \mathcal{W}} w_i^2)^{\frac{1}{2}} \tag{3.23}$$

- **Dropout**

  When using dropout, at every iteration of the training process neurons are dropped from the training, along with all their connections, with probability p, called *dropout rate*. They are then reactivated at the beginning of the next iteration and the process repeats. During inference, all neurons are active and their output is scaled by p which approximates an ensemble average of the outputs of different architectures sharing the same weights. Dropout prevents the network from becoming too reliant on specific neurons, which can lead to instability and overfitting.

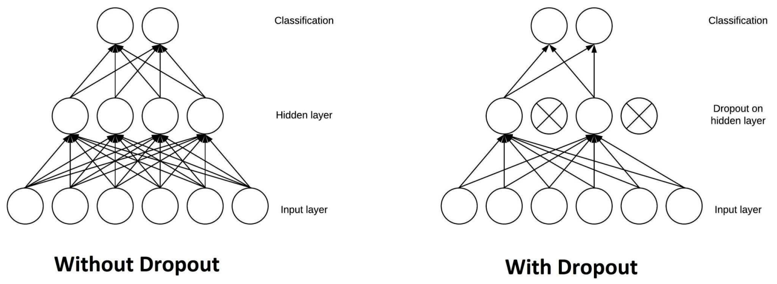

Figure 3.6: Architecture of a neural network classifier with and without dropout

3.2 Autoencoders

Autoencoders were first introduced in the 1980s as unsupervised learning method for dimensionality reduction and feature extraction. Early autoencoders were shallow networks with a single hidden layer and were closely related to Principal Component Analysis. With the rise of deep learning in the last twenty years, autoencoders evolved into deeper architectures, enabling more complex representations.

The goal of autoencoders is to learn an informative representation of the data. They work by encoding the input into a latent representation, that is then decoded to reconstruct the original data. The training is done by minimizing the reconstruction error, using a suitable loss like the mean squared error.

A schematic view of an autoencoder:

$$\begin{tikzcd}
\text{Input } \mathbf{x} \arrow[d] \\
\text{Encoder network } \phi(\cdot) \arrow[d] \\
\begin{array}{c}\text{Embedding} \\ \mathbf{c} = \phi(\mathbf{x}, \mathcal{W}_\phi)\end{array} \arrow[d] \\
\text{Decoder Network } \psi(\cdot) \arrow[d] \\
\begin{array}{c}\text{Reconstruction} \\ \mathbf{y} = \psi(\mathbf{c}, \mathcal{W}_\psi)\end{array}
\end{tikzcd}$$

The loss function is computed as

$$\begin{aligned}
\mathcal{L}((\mathbf{x}, \mathcal{W}_\phi, \mathcal{W}_\psi)) &= \| \mathbf{x} - \mathbf{y} \|^2 + \mathcal{R}(\mathbf{c}) \\
&= \| \mathbf{x} - \psi(\phi(\mathbf{x}, \mathcal{W}_\phi), \mathcal{W}_\psi) \|^2 + \mathcal{R}(\mathbf{c})
\end{aligned} \tag{3.24}$$

where $\mathcal{R}(\mathbf{c})$ is the regularization on the embedding vector $\mathbf{c}$ implemented to prevent overfitting by learning the identity function.

3.2.1 Variational autoencoders

Variational autoencoders (VAE) are an extension of classical autoencoders that integrate probabilistic modeling and deep learning. They are generative models that learn a probability distribution

$p(x)$ which is then used to generate new data.

Variational autoencoders follow the *Variational Bayes* idea to approximate the posterior distribution with a function $q(\mathbf{z}|\theta)$ belonging to a family of functions parametrized by θ. The optimal variational distribution is a sum of two contributions, equation (3.25). The first is the Kullback-Leibler divergence between $q(\mathbf{z})$ and the prior $p_{prior}(\mathbf{z})$, that constrains $q(\mathbf{z})$ to closely follow the prior. The second contribution comes from the data $\mathbf{x}$ and it constrains $q(\mathbf{z})$ to be concentrated around values of $\mathbf{z}$ with large likelihood $p(\mathbf{x}|\mathbf{z})$.

The prior represents our initial belief about the distribution of $\mathbf{z}$ before observing any data. The likelihood $p(\mathbf{x}|\mathbf{z})$ is the probability of observing $\mathbf{x}$ given $\mathbf{z}$, while the posterior represents the updated belief about the distribution after observing the data.

$$q^*(\mathbf{z}) = \text{argmax}_{\boldsymbol{\theta}} \left(-D_{KL}\left(q(\mathbf{z}|\boldsymbol{\theta}) \parallel p_{\text{prior}}(\mathbf{z})\right) + E_{z\sim q} \log p(\mathbf{x}|\mathbf{z})\right) \tag{3.25}$$

Assuming that the prior and the data are available to us, and the likelihood of the data belongs to a family of functions parametrized by the parameter ϕ, $p(\mathbf{x}|\mathbf{z}, \phi)$. Then a self-consistent generative model of the data can be found among a class of models.

$$q^*(\mathbf{z}) = \text{argmax}_{\boldsymbol{\theta},\boldsymbol{\phi}} \left(-D_{KL}\left(q(\mathbf{z}|\boldsymbol{\theta}) \parallel p_{\text{prior}}(\mathbf{z})\right) + E_{z\sim q} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\phi})\right) \tag{3.26}$$

The learning framework is the following:

$$\underset{\text{training data}}{x} \xrightarrow{Q(z|x;\mathcal{W_Q})} z \xrightarrow{P(x|z;\mathcal{W_P})} \underset{\text{generated sample}}{\hat{x}} \tag{3.27}$$

where the functions $Q(z|x;\mathcal{W_Q})$ and $P(x|z;\mathcal{W_P})$ represent the encoder and the decoder respectively, with $\mathcal{W_Q}$ and $\mathcal{W_P}$ being their weights.

Q is a stochastic mapping, not a deterministic function. To make it learnable with a neural network it should be parametrized so that the distribution shape can be chosen by learning the parameters. The typical choice is to use the normal distribution:

$$Q(z|x) = \mathcal{N}(z; \mu(x, \mathcal{W}_Q), \Sigma(x, \mathcal{W}_Q)) \tag{3.28}$$

where $\mu(x, \mathcal{W}_Q) = [\mu_1, ..., \mu_k]^T$ and $\Sigma(x, \mathcal{W}_Q) = diag([\sigma_1^2, ...\sigma_k^2]^T)$, the mean and variance of the normal distribution, are the parameters deterministically learned by the encoder.

The variational autoencoder training consists in maximizing $P(x)$

$$\log P(x) = D_{KL}(Q(\mathbf{z}|\mathbf{x}) \parallel P(\mathbf{z}|\mathbf{x})) \; + E_{z\sim q} \log P(\mathbf{x}|\mathbf{z}) - D_{KL}(Q(\mathbf{z}|\mathbf{x}) \parallel P_{\text{prior}}(\mathbf{z})) \tag{3.29}$$

The first term in equation (3.29) measures the distance between the unknown true distribution $P(\mathbf{z}|\mathbf{x})$ and the NN approximation $Q(\mathbf{z}|\mathbf{x})$, but because it is positive we can maximize $P(x)$ without it. This results in a lower bound on the objective function, and the training is done by maximizing the *lower evidence bound ELBO*.

$$\log P(x) \geq E_{z\sim q} \log P(\mathbf{x}|\mathbf{z}) - D_{KL}(Q(\mathbf{z}|\mathbf{x}) \parallel P_{\text{prior}}(\mathbf{z})) \tag{3.30}$$

The Kullback-Leibler divergence $D_{KL}(Q(\mathbf{z}|\mathbf{x}) \parallel P_{\text{prior}}(\mathbf{z}))$ forces the encoder not to depart strongly from the prior. By choosing a prior that follows a normal distribution $p_{prior}(z) = \mathcal{N}(z; 0, \mathbb{1})$, this term can be rewritten as:

$$D_{KL}(Q(\mathbf{z}|\mathbf{x}) \parallel P_{\text{prior}}(\mathbf{z})) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2) \tag{3.31}$$

The other term in equation (3.30), $E_{z\sim q} \log P(\mathbf{x}|\mathbf{z})$, is more problematic. It represents the log-likelihood and it is not solvable because it depends on the decoder function $P(\mathbf{x}|\mathbf{z})$. However, it can be approximated using Monte Carlo by drawing samples of $\mathbf{z}$ and calculating the empirical average.

$$E_{z\sim q} \log P(\mathbf{x}|\mathbf{z}) \approx \frac{1}{L}\sum_{l=1}^{L} \log P(\mathbf{x}|z_l) \tag{3.32}$$

$P(\mathbf{x}|z_l)$ is typically assumed to be gaussian.

The sampling process is not deterministic, it is a discontinuous operation with no gradient, meaning it can't be backpropagated. The answer to this lies in using the *reparametrization trick*:

$$z_l = \mu(\mathbf{x}) + \epsilon_l \sigma(\mathbf{x}) \tag{3.33}$$

and sample ϵ from a normal distribution with zero mean $\epsilon \sim \mathcal{N}(0, 1)$.

Finally, by putting everything together, the ELBO loss function for VAE training can be written as:

$$\begin{aligned} \mathcal{L}(\mathbf{x}) &= \frac{1}{L}\sum_{l=1}^{L} \log P(\mathbf{x}|z_l = \mu(\mathbf{x}) + \epsilon_l \sigma(\mathbf{x})) + \frac{1}{2}\sum_{i}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2) \\ &= \frac{1}{L}\sum_{l=1}^{L} \frac{\parallel \hat{\mathbf{x}}_l - \mathbf{x} \parallel^2}{2\sigma_P} + \frac{1}{2}\sum_{i}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2) \end{aligned} \tag{3.34}$$

where $\hat{\mathbf{x}}_l$ is a sample generated by the decoder starting from z_l, and σ_P is the variance hyperparameter that encapsulates the trade-off between the two terms. The first makes sure the model reconstructs the input samples as closely as possible, the second regularizes the VAE by forcing it to learn latent variables that follow an easy Gaussian distribution.

3.3 Transformers

From Multilayer Perceptrons (MLPs) to Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), deep learning models paved the way for advancements in image processing, time series analyses, and natural language processing(NLP). In recent years a new architecture has overtaken the world of deep learning, the **Transformer** introduced in the famous paper *Attention is all you need* by A.Vaswani et al. [17]. Popularized by groundbreaking models such as ChatGPT, BERT, and Gemini, transformers have revolutionized how sequential data is processed, analyzed, and generated. An integral part of the transformer is the **attention mechanism**. It allows the model to focus on relevant parts of the data dynamically, capturing both short and long-distance dependencies. This selective focus is akin to how humans pay attention to certain aspects of visual scenes or conversation.

Transformers address several limitations of the traditional sequential models like RNNs.

- **Parallel processing**
  While RNNs process data step-by-step, carrying forward a hidden state that contains historical information, Transformers process the entire sequence simultaneously. This allows for processing sequences in parallel, which significantly speeds up the training and inference by leveraging the power of modern hardware, such as GPUs.

- **Long-range dependencies**
  One of the main features of Transformers is their ability to capture long-range dependencies thanks to the attention blocks that weigh the importance of each element in a sequence independent of the distance. Understanding the context in sequences where relevant information may be separated by considerable distances is fundamental. This approach is a significant improvement over RNNs which suffer from vanishing gradients that cause diminishing influence of earlier elements in a sequence. While LSTMs (Long Short-Term Memory networks) addressed to some extent this issue by incorporating gating mechanisms to better retain information over time, they still struggle with modeling very long sequences.

Transformers can be categorized based on their architecture.

- *Encoder-decoder transformers* were the original transformers, in Fig 3.3 is shown the architecture of the first Transformer introduced in 2017 [17]. They are sequence-to-sequence models focusing on tasks that require to transform one type of sequence into another. The input sequence is processed by an encoder while the output sequence is generated by a decoder. They are mostly used in translation, where sequences are mapped from one language to another.

- *Decoder-only transformers* center around generative tasks. The decoder-only models don't require an explicit input sequence and don't focus on the underlying context, instead they use

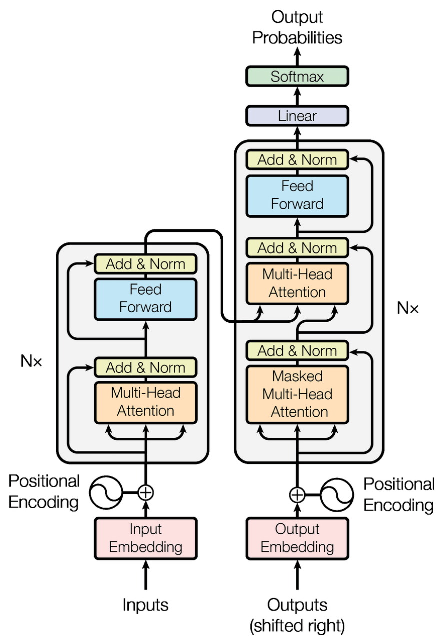

Figure 3.7: Architecture of the first Transformer introduced in [17]

prior tokens, words for text generation task, to iteratively predict the next token.

- *Encoder-only transformers* aim at understanding the input sequence while generating an output sequence is not required. They are mainly used in classification problems as well as in anomaly detection. Due to their nature of capturing meaningful dependencies and patterns in data, encoder-only models can also be used in compression tasks.

While the attention mechanism makes transformers highly powerful algorithms, they are also know to have an increased number of parameters, which demand significant memory and computational resources.

Chapter 4

A Deep Learning compression algorithm

The elegance of the attention mechanism lies in its simplicity and effectiveness. By allowing models to focus on various aspects of data and weigh the relevance of the information that each sequence element brings, transformers excel in tasks requiring a deep understanding of context. They have not only redefined NLP, but their impact has also extended to fields like computer vision, audio processing, time series analyses, etc.

Recently, DeepMind's paper *"Language Modeling is Compression"* [23] drew a connection between large language models and data compression attributing to the role of attention in creating efficient representations. Attention can be used to improve compression models, by adjusting dynamically the level of compression based on the content's complexity and importance, it can identify and retain crucial information while being more aggressive on less important parts. Attention can detect repeating patterns or trends over long distances. During decompression, its selective focus can assist in reconstructing key components with high accuracy.

Motivated by the desire to explore the strengths of the attention mechanism in the context of compression while keeping model complexity and resource demands low, the model proposed in this thesis is a lossy compressor inspired by the paper *Deep Dict: Deep Learning-based Lossy Time Series Compressor for IoT Data* by J.Liu et al.[24].

4.1 Model architecture

The compression algorithm consist of an autoencoder that encodes the input into Bernoulli distributed latent states rather than real-value latent states in order to drastically reduce the size of the latent states and improve compression rate. In addition, a distortion constraint is used to limit the maximum acceptable reconstruction error. Fig 4.1 shows an overview of the model.

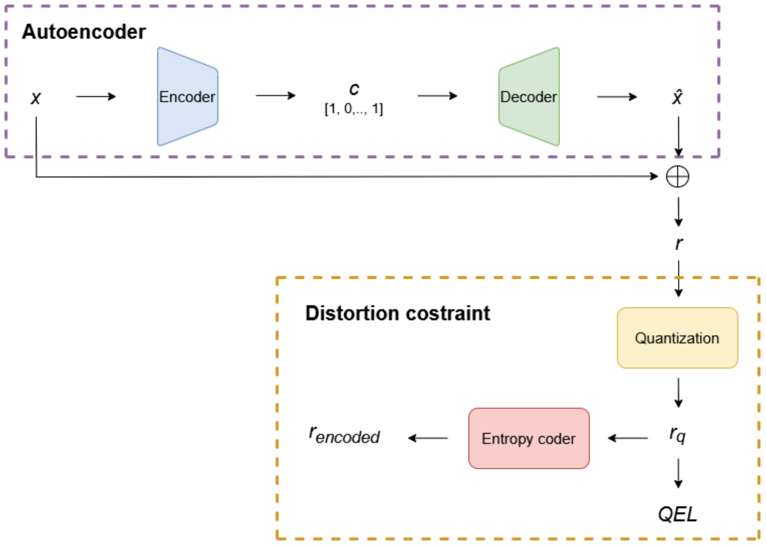

Figure 4.1: An overview of the compression model

4.2 Data preprocessing

Before the data goes through the encoding process it is divided into non-overlapping segments of fixed length, called time windows. This allows the model to use a fixed-size input from which to learn the temporal relationships in sequential data. The size of the window is controlled by the *window size* hyperparameter.

After windowing, the data goes through a normalization process which shifts and scales the inputs into a distribution centered around 0 with standard deviation 1. Normalization is an important prepossessing step that strengthens numerical stability, improves convergence in optimization and balances feature contribution.

For real-time application the regularization of the data through normalization is not actually feasible. However, in a future implementation different methods could be used to handle the full dynamic range of the data.

4.3 Encoder

The Encoder has a very simple structure. It is composed of three fully connected layers with 64 neurons in each of them. The output layer uses a Bernoulli distribution, from the Tensorflow probability library [25] that integrates probabilistic methods with deep learning networks, to encode each time window segment of the data into a bit array of size $|c|$. The size of the latent states is one of the hyperparameters of the model, it is typically set to 30, but it is adjusted based on the data dimensionality and complexity.

4.4 Decoder

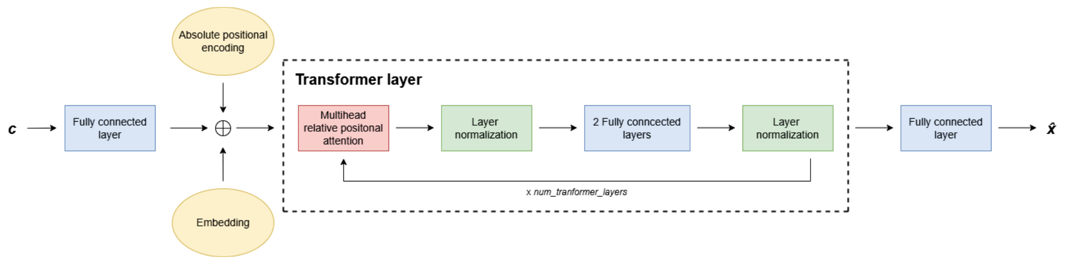

Figure 4.2: Schematic view of the decoder.

The decoding process starts by expanding the encoded states through a fully connected layer with L neurons. L represents the *sequence length*, or the number of tokens in a sequence and it was set to 64.

Then embedding and absolute positional encoding are applied. The *embedding* converts the input tokens to dense vectors in continuous space that capture meaningful relationships and patterns in the data. While the positional encoding provides positional information to input tokens in a sequence. Each encoding vector has the same dimensionality, d_{model}, as an embedding vector, to allow for them to be summed.

The main part of the decoder is the transformer layers containing the attention block. They consist of multi-head relative positional attention followed by layer normalization, 2 fully connected layers, and another layer normalization. The two fully connected layers with ReLU activation in between, allow for the introduction of non-linearity into otherwise linear attention calculations. The number of transformer layers used is a hyperparameter usually set to 2 or 4 based on the complexity of the dataset. The output of the transformer layers is then passed through a fully connected layer to obtain the reconstruction.

4.4.1 Absolute positional encoding

Transformer models, unlike recurrent neural networks (RNNs), process all tokens in a sequence simultaneously. However, this means transformers lack inherent knowledge of the order of tokens. To address this, positional embeddings are introduced to encode the position of each token within the sequence.

There are several different strategies for positional encoding. A common approach is the sinusoidal positional embedding, also used in the original Transformer model introduced by A.Vaswani et al.[17]. Each position in a sequence is represented using sine and cosine functions of varying frequencies.

$$\begin{aligned} PE(pos, 2i) &= \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \\ PE(pos, 2i+1) &= \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \end{aligned} \tag{4.1}$$

where pos is the position of the token in the sequence, i is the specific dimension within the embedding vector and d_{model} is the dimensionality of the embeddings.

The wavelengths of the sinusoidal functions form a geometric progression from 2π to $10000 \cdot 2\pi$. This ensures positional information is encoded across a wide range of frequencies.

The choice of 10,000 as the base for the exponential scaling is deliberate. It ensures that the frequency spectrum is spread over a logarithmic scale, allowing the model to capture positional patterns over both short and long ranges. As i increases, $10000^{\frac{2i}{d_{model}}}$ grows exponentially, resulting in decreased frequency and longer wavelengths. Lower dimensions (small i) have higher frequencies, capturing fine-grained positional patterns, while higher dimensions (large i) have lower frequencies, capturing broader patterns. By setting the denominator to $10000^{\frac{2i}{d_{model}}}$, the rate of change of the sinusoidal functions is moderated, which helps ensure stable gradient propagation across a range of positions and dimensions.

A key advantage of sinusoidal positional embeddings is their ability to generalize well to sequence positions beyond those encountered during training, thanks to the periodic nature of sine and cosine functions. Additionally, these embeddings allow the model to infer relative positional relationships, as the encoding for $PE(pos+k)$ can be expressed as a linear function of $PE(pos)$ for any fixed offset k.

However, sinusoidal embeddings have some limitations. They are fixed and non-adaptive to a specific data or task. Additionally, in certain scenarios, they may struggle to capture complex positional relationships as effectively as learned embeddings.

4.4.2 The Attention mechanism

The Attention mechanism allows the model to weigh the significance of different parts of the input independently of their position in the sequence. It works by first transforming a sequence of L input vectors $X = (x_1, x_2, ...x_L)$ of dimension d_{model} into **queries** $Q = XW^Q$, **keys** $K = XW^K$ and **values** $V = XW^V$ obtained by multiplying the input vectors X with $d_{model} \times d_{model}$ weight matrices W^Q, W^K and W^V. Then a **self attention** function is applied, based on a scaled dot-product between the Q, K, and V matrices as proposed in [17].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right)V \tag{4.2}$$

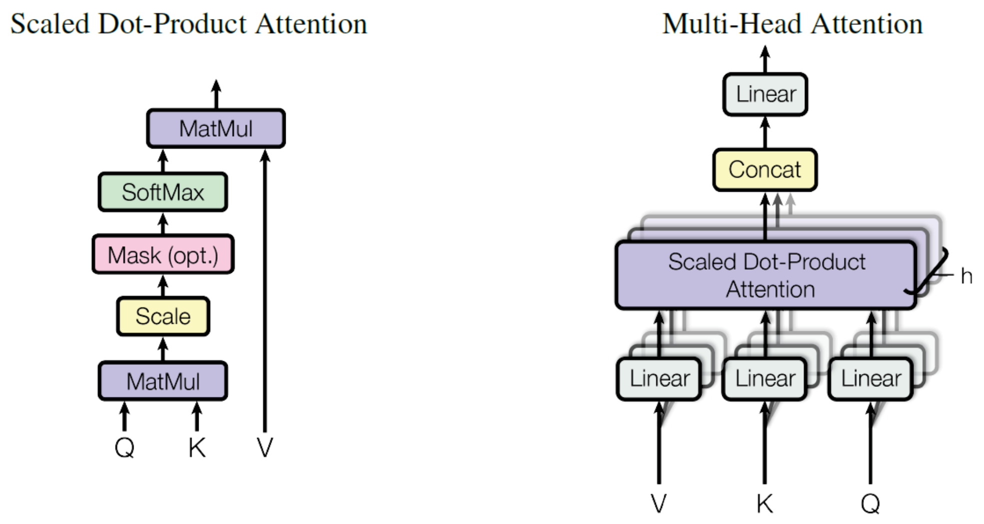

Figure 4.3: Scaled Dot-Product Attention and Multi-head Attention as introduced in [17]

To further improve performance, **multi-head attention** is used. It enables the model to focus on different parts of the sequence simultaneously by using multiple attention heads. Each head learns distinct aspects of the data, enhancing the model's representational potential. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. It works by projecting the queries, keys, and values h times. On each of these projected versions of Q, K, and V the attention function is performed in parallel.

$$\text{Attention}(Q^h, K^h, V^h) = \text{softmax}\left(\frac{Q^h K^{hT}}{\sqrt{d_h}}\right)V^h \tag{4.3}$$

The result is a sequence of vector outputs for each head that are then concatenated together and linearly transformed to obtain the final output, an $L \times d_{model}$ dimensional matrix.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}\left(\text{head}_1, \ldots, \text{head}_\text{h}\right) W^O \\ \text{where head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{4.4}$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{model} \times d_h}$ and $d_h = d_{model}/h$.

Because of the reduced dimension of each head, the computational cost of multi-head attention is comparable to single-head attention with full dimensionality.

The model, as previously mentioned, relies on sinusoidal encoding for absolute positional information. Adding relative position representations will allow the attention to be informed by how far two positions are apart in a sequence. This is achieved by learning a relative positional embedding E^r for each pairwise distance $r = j_k - i_q$ between a query at position i_q and a key at position j_k. Then the S^{rel} matrix is created containing the dot products between each query q and embedding of the relative distance $r = j_k - i_q$. The **relative attention** for each head is given by the function:

$$\text{Relative Attention} = \text{softmax}\left(\frac{QK^T + S^{rel}}{\sqrt{d_h}}\right) V \tag{4.5}$$

To improve memory efficiency, the computation of S^{rel} is done by directly multiplying Q with E^r resulting in a matrix with (i_q, r) entries containing the dot product of the query in position i_q and the embedding of a relative distance r. Then a "skewing" procedure is implemented to move the QE^{rT} elements to their correct positions (i_q, j_k) in order to arrive at the desired S^{rel}. This procedure was introduced in the *Music Transformer* by Huang, Vaswani, et al.[26]. It consists in shifting column indices r to reproduce column indices $j_k = r - (L-1) + i_q$ as depicted in Fig 4.4.2. For a detailed description of the procedure see the paper [26].

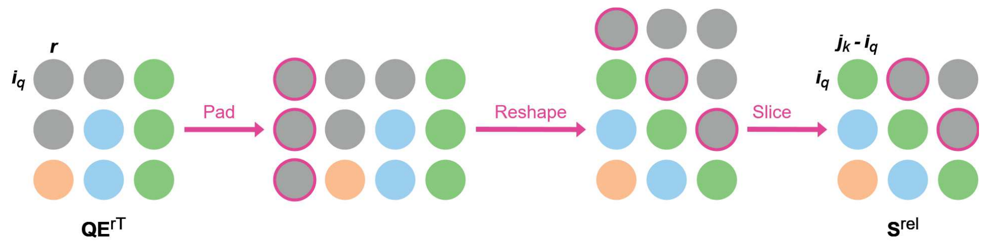

Figure 4.4: The memory efficient "skewing" algorithm for Relative attention proposed in the Music Transformer by Huang, Vaswani, et al. [26]

4.5 Distortion constraint

A distortion constraint is added to the model to reduce the reconstruction error to a desired range. It is done by calculating the residual between the reconstruction and the original data $r = |x - \hat{x}|$. The residual is then uniformly quantized to r_q.

$$r_q \;=\; 2\epsilon \;\text{round}(\frac{r}{2\epsilon}) \tag{4.6}$$

where ϵ is the quantization step and determines the maximum acceptable reconstruction error. Small epsilon leads to small error, but there is a trade-off with the computation complexity and the compression rate. For small ϵ, the number of unique values of r_q will be large.

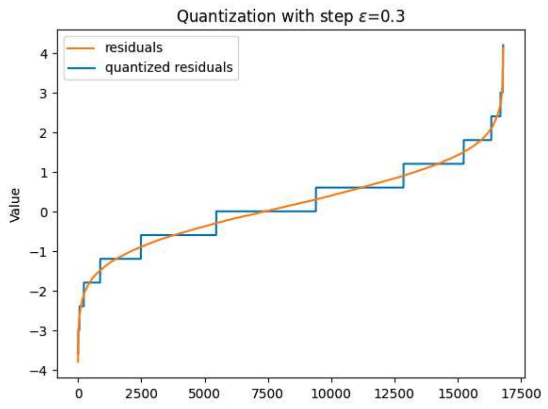

Figure 4.5: Example of residual quantization

The quantized residual r_q is then encoded to $r_{encoded}$ and stored to be used for reconstruction of the compressed data. For the encoding of r_q the lossless entropy coder *bsc* was used form *libbsc* [27] which is an open-source, high-performance data compression library. The compression is done by assigning shorter codes to more frequent symbols. The more frequently a value in r_q appears, the shorter is the code assigned to it.

4.6 Quantized Entropy Loss (QEL)

Considering that a distortion constraint was already imposed on the reconstruction error, the loss function does not focus on reducing the discrepancy between the original data and the reconstruction, instead, it aims to minimize the size of $r_{encoded}$.

$$\min_{r} H(r) = -\sum_{j=0}^{|S|} p\left(s_j\right) \log p\left(s_j\right) \tag{4.7}$$

where s_j are the unique values of r_q and $p(s_j)$ is the probability of s_j appearing in r_q. Uncertain and unpredictable data has higher entropy. By minimizing the entropy of the quantized residuals, the number of bits required to encode the residuals is decreased, all while maintaining reconstruction quality.

During the optimization process, an approximation is necessary to enable the backpropagation of the loss gradient, due to having a discreet quantized variable. The gradient of H is calculated as follows:

$$\frac{\partial H}{\partial r_i} = \lim_{b \to \infty} \sum_{j=0}^{|S|} \left[1 + \ln p\left(s_j\right)\right] \times R\left(r_i - s_j\right) \tag{4.8}$$

where

$$R\left(r_i - s_j\right) = \frac{b}{|r| \epsilon^b} \frac{\left(r_i - s_j\right)^{b-1}}{\left[\frac{\left(r_i - s_j\right)^b}{\epsilon^b} + 1\right]^2} \tag{4.9}$$

and b is a hyperparameter that controls the sharpness of the R function.

4.7 Reconstruction

The reconstruction of the compressed data involves decoding the latent states c with the decoder and decoding the residuals $r_{encoded}$ with *libbsc*. The final reconstruction is obtained by performing the inverse normalization on the sum of the decoded residual $r_{decoded}$ and the output of the decoder $\hat{x}$.

$$x_{reconstructed} = \text{Inverse normalize}\left(\hat{x} + r_{decoded}\right) \tag{4.10}$$

4.8 Implementation

For the implementation of the model into the acquisition and control system of the experiment a dual-channel approach is thought to be effective at processing and managing the data. The first channel will focus on real-time encoding of the input data to latent representations. These latent states, can be leveraged by the control system to make informed decisions. One potential future direction is to use autoencoders or similar networks to learn a mapping between the latent states and the physical characteristics of the signal, allowing the control system to better interpret and respond to real-time data changes. This approach would enable the control system to operate with more compact and efficient representations of the data, potentially improving system performance and responsiveness. In the second channel the data is processed through the transformer-based decoder to generate the residual encodings. These residuals are crucial for achieving reconstruction with errors constrained within a specific range. The residual encodings together with the latent states are stored for future use, ensuring that the data can be accurately reconstructed when required. Through this system, the model not only facilitates efficient signal compression but also integrates directly into the broader control and monitoring infrastructure.

4.8.1 Code

The code was implemented using TensorFlow. The model combines a probabilistic encoder and a transformer-based decoder to achieve efficient signal compression and reconstruction. The encoder compresses input data into a lower-dimensional latent representation using a series of dense layers with ReLU activation. The final output of the encoder is processed through a TensorFlow Probability (TFP) layer, which produces a Bernoulli distribution representation of the latent states.

```python
class Encoder(tf.keras.Model):
    def __init__(self, latent_dim):
        super().__init__()

        self.ffn1 = tfk.layers.Dense(64, activation="relu")
        self.ffn2 = tfk.layers.Dense(64, activation="relu")
        self.ffn3 = tfk.layers.Dense(tfpl.IndependentBernoulli.params_size(latent_dim
    ), activation=None)
        self.output_layer = tfpl.IndependentBernoulli(latent_dim)
        self.flatten = tfk.layers.Flatten()

    def call(self, x, training=False, **kwargs):

        x = self.flatten(x)
        f_out = self.ffn1(x)
        f_out = self.ffn2(f_out)
        f_out = self.ffn3(f_out)
        out = self.output_layer(f_out)

        return out
```

Listing 4.1: Encoder

The decoder utilizes a stack of custom transformer layers to reconstruct the input data from the encoded latent representation. It enhances input embeddings with sinusoidal positional encodings and employs relative positional encoding as introduced in the Music Transformer [26]. The Transformer Layers consist of multi-head attention and feedforward networks with layer normalization in between. These layers allow the model to learn complex relationships within the latent space and ensure that the decoder can faithfully reconstruct the original data.

```python
class TransformerLayer(keras.layers.Layer):
    def __init__(self, d_model, rate=0.1, h=8, additional=False, max_seq=2048):
        super(TransformerLayer, self).__init__()

        self.d_model = d_model
        self.rga = RelativeGlobalAttention(h=h, d=d_model, max_seq=max_seq, add_emb=
    additional)

        self.FFN_pre = keras.layers.Dense(self.d_model // 2, activation=tf.nn.relu)
        self.FFN_suf = keras.layers.Dense(self.d_model)

        self.layernorm1 = keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = keras.layers.Dropout(rate)
        self.dropout2 = keras.layers.Dropout(rate)

    def call(self, x, mask=None, training=False, **kwargs):
        attn_out, w = self.rga([x,x,x])#, mask=None)
        attn_out = self.dropout1(attn_out, training=training)
        out1 = self.layernorm1(attn_out+x)

        ffn_out = self.FFN_pre(out1)
        ffn_out = self.FFN_suf(ffn_out)
        ffn_out = self.dropout2(ffn_out, training=training)
        out2 = self.layernorm2(out1+ffn_out)
        return out2, w


class Transformer(keras.layers.Layer):
    def __init__(self, num_layers, d_model, h, input_vocab_size, rate=0.1, max_len=
    None):
        super(Transformer, self).__init__()

        self.d_model = d_model
        self.num_layers = num_layers

        self.embedding = keras.layers.Embedding(input_vocab_size, d_model)

        if True:
            self.pos_encoding = DynamicPositionEmbedding(self.d_model, max_seq=
    max_len)

```

```
        self.transformer_layers = [TransformerLayer(d_model, rate, h, additional=False, max_seq=max_len)
                           for i in range(num_layers)]
        self.dropout = keras.layers.Dropout(rate)

    def call(self, x, mask=None, training=False):
        weights = []
        # adding embedding and position encoding.
        x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x = self.pos_encoding(x)
        x = self.dropout(x, training=training)

        for i in range(self.num_layers):
            x, w = self.transformer_layers[i](x, mask=None, training=training)
            weights.append(w)

        return x, weights
```

Listing 4.2: Transformer decoder

A custom loss function designed for compression tasks, computes the entropy of previously quantized residuals. To ensure differentiability, a custom gradient approximation is implemented, allowing the loss gradient to be backpropagate during training.

```
def quantized_entropy_loss(epsilon,b=10.0):
  #epsilon is the quantization step, small epsilon -> small reconstruction error, but there is a trade off with the computation complexity (for small epsilon the number of unique values of rq will be large)
  @tf.custom_gradient
  def loss_func(r):

    #Quantization
    rq = tf.cast((2*epsilon) * tf.math.round(r/(2*epsilon)), dtype=tf.float64) # epsilon is the quantization step

    # Calculate entropy
    unique_rq, _, count_rq = tf.unique_with_counts(tf.reshape(rq, [-1]))
    p_rq = count_rq / tf.reduce_sum(count_rq) #probability of each unique residual
    QEL = tf.cast( -tf.reduce_sum(p_rq * tf.math.log(p_rq + 1e-8)), dtype=tf.float32)

    #Gradient approximation
    def R(rq, sj, b=b, epsilon=epsilon):
          return (b / (tf.cast(tf.size(rq), dtype=tf.float64) * epsilon**b)) * ((rq - sj)**(b-1)) / (((rq - sj)**b / epsilon**b) + 1)**2
    def grad(upstream):
      gradient_approx = 0.0 #dH/dr
      for sj,pj in zip(unique_rq, p_rq):
          gradient_approx += (1 + tf.math.log(pj + 1e-8)) * R(rq,sj)   #compute
    gradient sum for each unique value
      return upstream * tf.cast(gradient_approx, dtype=tf.float32)

    return QEL, grad

  return loss_func
```

Listing 4.3: Custom Loss function

The encoder and decoder are integrated into an autoencoder architecture. Input features are normalized to improve training stability and ensure compatibility across varying data scales. The encoder's latent representation is expanded through a dense layer and processed by the transformer-based decoder. The decoder reconstructs the original data shape through a combination of dense and reshaping layers. To prevent overfitting, dropout layers are incorporated into the model.

The machine used for the model training features a 4 core 8 threads Intel(R) Xeon(R) W-2223 CPU @ 3.60GHz and an NVIDIA Geforce RTX 3090 24 GB VRam, with 500 GB of RAM.

Chapter 5

Results

In this chapter the results of a variety of tests of the proposed compression model are shown. The model was first tested on publicly available datasets containing sensor recordings, a thorough exploration of its behavior and sensitivity to hyperparameter adjustments was conducted. These initial experiments provided valuable insights into the model's functionality and a deeper understanding of the importance and impact of its hyperparameters. The model was subsequently applied to a dataset containing historical data from the RFX experiment. Then the model's performance, was benchmarked against two widely recognized error-bounded lossy compression frameworks, SZ3 and LFZip. These comparisons allowed for an in-depth analysis of the model's compression efficiency, robustness across varying error constraints, and its ability to handle datasets with increasing dimensionality.

5.1 Hyperparameter tuning

Several different strategies can be used to find the set of hyperparameters that lead to the best performance of a model.

- *Grid search* works by trying out every possible combination of hyperparameters, it trains the model for each of them and selects the combination with the best results on a test dataset. This approach is useful when the number of hyperparameters is small, otherwise, the number of different combinations to be tried out becomes too large and a different strategy should be considered.

- *Random search* selects a random subset of hyperparameters on which to evaluate the model instead of trying all possible combinations. It is useful for finding a good result in a short search time but there is no guarantee that it is the best result.

- *Bayesian optimization* is a more sophisticated approach that uses a probabilistic model, typically a Gaussian Process, to approximate the objective function and predict promising hyperparameter combinations.

Optuna was used for the hyperparameter tuning of the compression model. Optuna [28] is an open source framework for hyperparameter optimization based on advanced optimization techniques, like Bayesian optimization, that automates the process of searching for optimal hyperparameter values.

The objective of the optimization process is to maximize the **compression ratio** given by the size of the data in bytes before compression divided by the size of the data after the compression.

5.1.1 Exploration of the hyperparameter space

Before applying the compression model to data from the RFX experiment, it was first tested on two distinct datasets containing sensor recordings. This allowed for exploration of the model's functionality and a deeper understanding of the importance and impact of its hyperparameters.

The datasets used are:

- *Heterogeneity Activity Recognition dataset* [29] containing data from Smartphones and Smartwatches sensors devised to benchmark human activity recognition algorithms. Accelerometer recordings from Samsung Galaxy S3 mini were used for the initial testing of the model. This dataset contains 7190 instances of measurements along the three axes (x, y, z).

- *Gas sensors for home activity monitoring* [30]. This dataset is composed of recordings from a gas sensor array consisting of 8 MOX (metal oxide) gas sensors under three different conditions, background home activity, and exposure to two different stimuli: wine and banana. The dataset includes 928,500 instances of measurements from the 8 sensors.

The hyperparameter exploration revealed that the **window size** is the most impactful parameter. Values in the range between 500 and 1000 lead to satisfying results. For small values, the compression time is significantly longer, but it decreases sharply and stabilizes for values larger than 400.

The **number of transformer layers** and $\mathbf{d_{model}}$ are directly linked to the model's complexity, as they determine the total number of parameters. As shown in Fig 5.1, the choice of the number of transformer layers is a trade-off between the compression ratio and the compression time. However, large values can lead to overfitting, which is why this hyperparameter is set to 2 or 4. The value of $\mathbf{d_{model}}$ was fixed to 32.

The **size of the latent states** is adjusted based on the data complexity and dimensionality. For example, for the *Heterogeneity Activity Recognition dataset*, a value of 10 was sufficient. While for the other dataset, *Gas sensors for home activity monitoring*, $|c|$ was set to 30, reflecting the larger number of sensors.

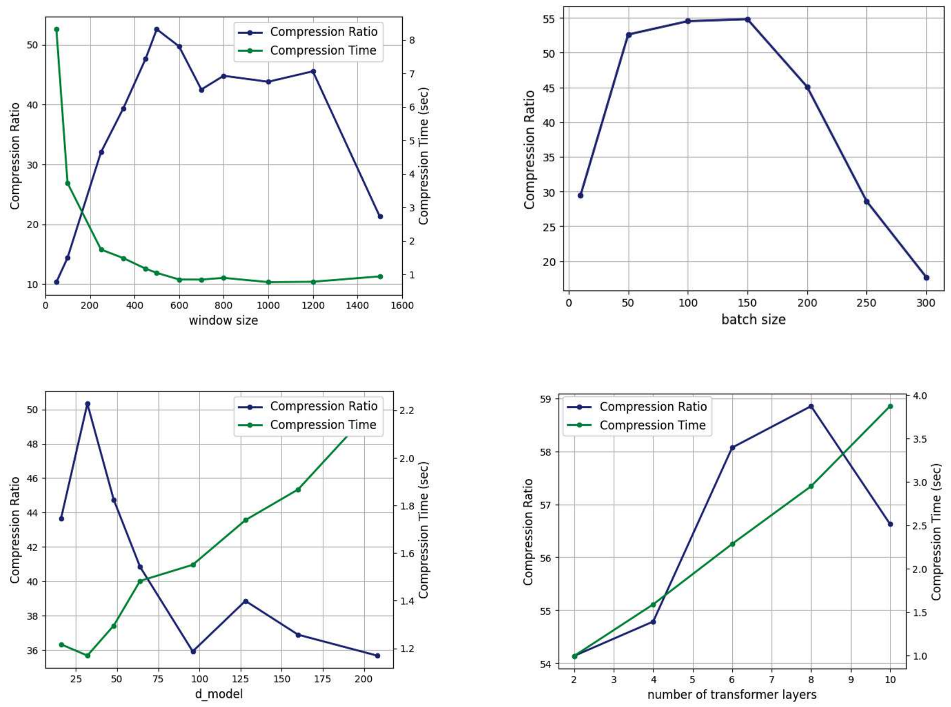

Figure 5.1: The effect of some hyperparameters (top left: *window size*, top right: *batch size*, bottom left: d_{model}, bottom right: *number of transformer layers*) on the compression ratio and compression time for fixed ϵ.

For the training of the model, **Adam** was used as optimizer with a *learning rate* of 0.01 and **weight decay** of 0.001. Additionally, **dropout** with a rate of 0.2 was applied to prevent overfitting. It was found that values of **b**, the hyperparameter of the loss gradient approximation, in the range of 6 to 12 result in a stable optimization process. And for the **batch size** a value between 50 to 150, as seen in Fig 5.1.

5.2 Results on data from the RFX experiment

Finally, the compression model was tested on a dataset created from historical data from the RFX database.

The dataset consists of values of the following quantities for 100 discharges.

- Toroidal magnetic field at the minor radius a
- Poloidal magnetic field at the minor radius a

- Toroidal tension at the minor radius a
- Poloidal tension at the minor radius a
- Pinch parameter at the minor radius a
- Reversal parameter at the minor radius a
- Safety factor at the minor radius a

All the computed values concur to represent the magnetic configuration of the plasma during the discharge evolution, and are collected by the pick-up coils laying in the intenral edge of the conductive thin-shell.

For this dataset, the time window size was set to encompass the duration of one discharge. While dividing the timeseries into multiple segments was explored, processing the entire discharge ultimately proved to be the most effective approach.

Figure 5.2 illustrates the reconstruction of plasma parameters after compression with a fixed maximum absolute error of 0.24. Overall the decoder manages to successfully reconstruct the signals with high accuracy and great detail, however it sometimes struggles with noisy data.

5.2.1 Comparison with other compression algorithms

In order to evaluate the performance of the proposed compression model, a comparison was made with two widely used general-purpose compressors, SZ3 and LFZip, both of which are well-established error-bounded lossy compression algorithms.

- **SZ3** [31] is modular prediction-based, error-bounded lossy compression framework for scientific datasets. It supports multialgorithm predictors and can automatically select the best-fit predictor for each data block based on the designed error estimation criterion.
- **LFZip** [32] performes multivariate floating-point time series lossy compression under maximum error distortion. The compressor is based on the prediction-quantization-entropy coder framework and uses linear models and neural networks to improve prediction.

Both compressors were chosen for their ability to handle scientific data efficiently, providing a relevant benchmark for assessing the performance of the new compression approach.

To assess the performance of the proposed compression model in comparison to the widely used general-purpose compressors SZ3 and LFZip, a series of experiments were conducted using datasets containing plasma diagnostic signals. The goal was to evaluate how well the proposed

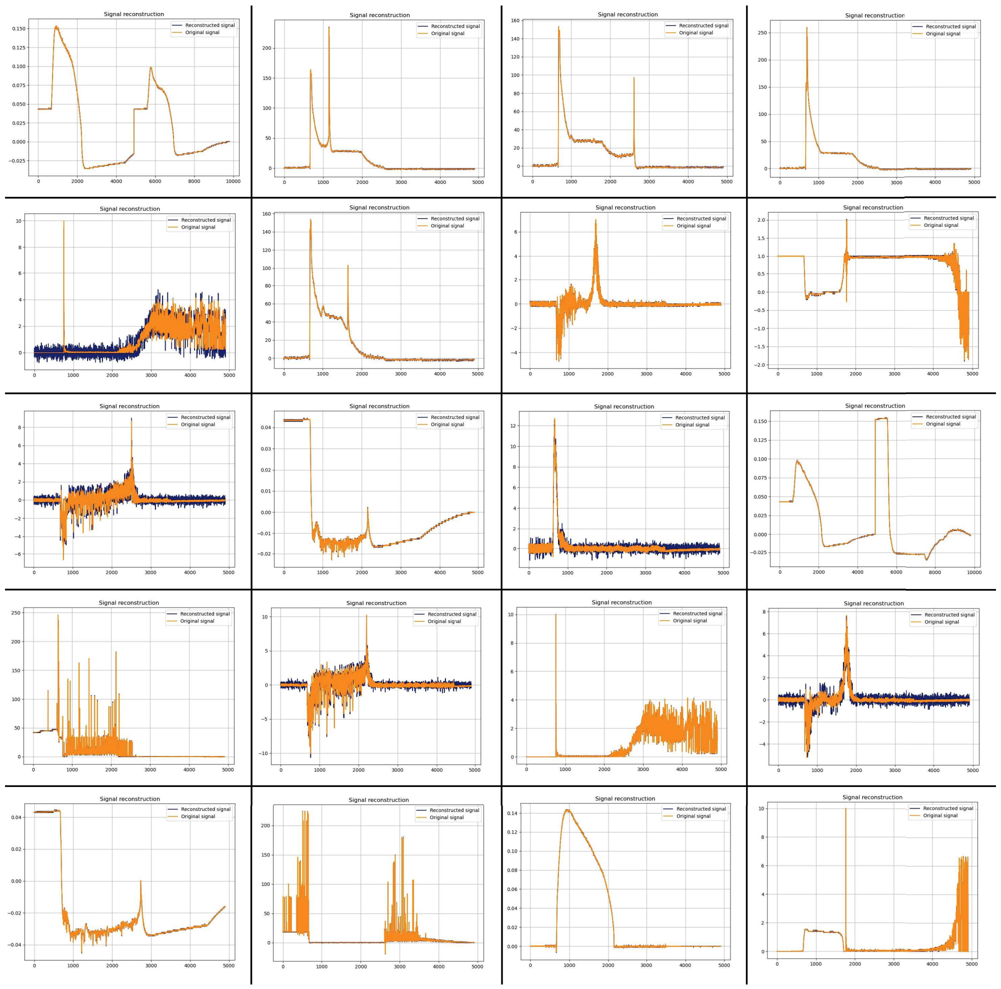

Figure 5.2: Examples of reconstructed signals. In orange the original signal, in blue the reconstruction from compressed data.

model handles various levels of error tolerance and data dimensionality in comparison to these established algorithms. The results of these experiments, shown in Figures 5.3 and 5.4, highlight the advantages and limitations of each compression method. Specifically, the comparison focuses on the compression rate achieved for different maximum absolute error levels and how the performance of the algorithms varies as the dimensionality of the datasets increases.

Figure 5.3 presents a comparison of compression rates for varying levels of maximum absolute error, using a dataset that includes the seven plasma quantities previously mentioned. The results clearly demonstrate that the proposed model outperforms both LFZip and SZ3, especially as the limit

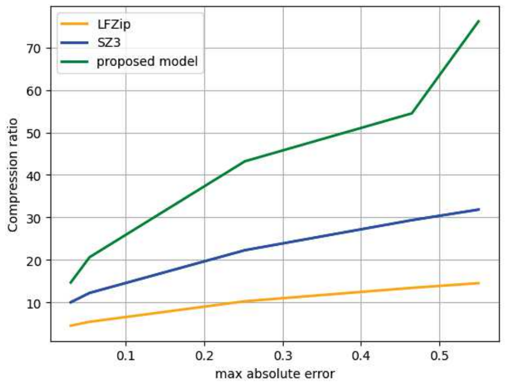

Figure 5.3: Comparison of compression rate for varying maximum absolute error on a dataset containing seven plasma variables (toroidal and poloidal magnetic field and tension, pinch and reversal parameters and safty factor at the minor radius), evaluated on the compression model proposed in this thesis, LFZip and SZ3.

on the maximum absolute error is relaxed. This indicates that the proposed model is more effective in capturing important signal features, achieving higher compression rates. As the error tolerance increases, the performance gap between the proposed model and the general-purpose algorithms grows, indicating that the proposed model achieves a greater increase in compression when higher reconstruction errors are allowed.

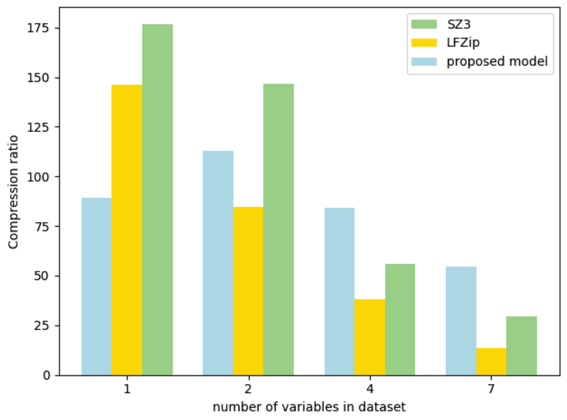

Figure 5.4: Compression rates for datasets with varying number of parameters, evaluated on the proposed model, LFZip and SZ3. The single-variable dataset contains the toroidal magnetic field, the two-variable dataset includes both toroidal and poloidal magnetic field, magnetic tension is added to expand the dataset to 4 variables, the pinch and reversal parameters along with the safety factor complete the seven plasma quantities used in the largest dataset.

In order to further assess the performance of the three algorithms across datasets with varying dimensionality, four different datasets were created, each containing a different number of variables. The first dataset is a single-variable dataset, which only includes the toroidal magnetic field. A second dataset was then created by adding the poloidal magnetic field, making it a two-variable dataset. The third dataset expands the previous by including magnetic tension, bringing the total number of variables to four. Finally, the largest dataset includes the pinch and reversal parameters, as well as the safety factor. Figure 5.4 illustrates the results of applying the three algorithms to these datasets with varying dimensions.

For single-variable datasets, both LFZip and SZ3 performe better than the proposed model, demonstrating their strength in handling simpler data structures. When the dataset expands to two variables, SZ3 continues to perform better than the proposed model, while LFZip falls behind, marking a shift in relative performance as data complexity slightly increases.

However, as the dimensionality of the datasets grows further, the proposed model begins to outpace both general-purpose compressors. For datasets containing four and seven variables, the proposed model achieves significantly higher compression rates than LFZip and SZ3. This improvement highlights the model's capacity to effectively handle the increased complexity and interdependencies present in multi-variable datasets.

This shift in performance can be attributed to the adaptive nature of the model compared to the non-adaptive approach of the general-purpose compressors. In datasets containing only one or two variables, the information learned from training on historical data may not be sufficient to provide an advantage over general-purpose compression algorithms, which are designed to perform well across a broad range of data. Conversely, for datasets that include more plasma quantities, the patterns and dependencies captured by the proposed model during training become more informative, allowing it to leverage these insights for better compression. In contrast, the general-purpose compressors struggle to exploit such domain-specific relationships, leading to their diminished effectiveness as the dimensionality increases.

Chapter 6

Conclusions

The goal of this thesis was to develop a deep learning compression algorithm trained on historical data from the RFX database. The motivation behind the choice of the architecture was to construct a model that leverages the power of the attention mechanism for capturing long-range dependencies and meaningful patterns while maintaining low computation complexity. This was achieved by strategically placing the attention blocks in the decoder, a counterintuitive choice, but as a result, the dimensionality of the input to the attention function was decreased from *window size* ≈ 5000 to *sequence length* $= 64$, reducing significantly the total number of model parameters. In addition, a distortion constraint was incorporated to ensure the reconstruction error remains within a predefined limit.

The obtained results are promising. The proposed algorithm outperforms the widely used general-purpose compressors, LFZip and SZ3 on datasets with more than two variables. Furthermore the performance gap between them widens as the limit on the maximum absolute error becomes more loose.

Given the satisfying results, the next step would be to implement the algorithm in the acquisition process in order to attempt a real-time compression. The idea is to use a buffer that stores the incoming data and when the signals of an entire discharge are collected, the data can be compressed and the buffer emptied, ready to collect data from the next pulse.

One potential improvement to the algorithm is to implement multiscale attention, where half of the attention heads focus on a fine-grained local scale and the other half on a broader global scale. The approach is expected to enhance the model's ability to handle noisy data.

Looking ahead, the model could also be used to leverage data-driven system identification by mapping the latent states to a simpler subspace using learned projections produced by autoencoders or similar networks. This would allow for the signals to be characterized in terms of their physical properties utilizing a semi-supervised learning approach to integrate domain knowledge.

Bibliography

[1] John A. Wesson. *Tokamaks*. Clarendon Press, 2008.

[2] E. Oz et al. Experimental verification of the kruskal-shafranov stability limit in line-tied partial-toroidal plasmas. *Physics of Plasmas*, 18(10):102107, Oct 2011.

[3] S. Cappello, D. Bonfiglio, and D. F. Escande. Magnetohydrodynamic dynamo in reversed field pinch plasmas: Electrostatic drift nature of the dynamo velocity fields. *Physics of Plasmas*, 13(5):056102, May 2006.

[4] J. B. Taylor. Relaxation of toroidal plasma and generation of reverse magnetic fields. *Physical Review Letters*, 33:1139, 1974.

[5] P. Piovesan et al. RFX-mod: A multi-configuration fusion facility for three-dimensional physics studies. *Physics of Plasmas*, 20(5):056112, May 2013.

[6] T. Bolzonella et al. Feedback control of resistive wall modes by saddle coils in RFX-mod. *Fusion Engineering and Design*, 82(5):1064–1072, 2007. Proceedings of the 24th Symposium on Fusion Technology.

[7] G. Manduchi et al. The New Feedback Control System of RFX-mod Based on the MARTe Real-Time Framework. *IEEE Transactions on Nuclear Science*, 61(3):1216–1221, 2014.

[8] N. Marconato et al. Design of the new electromagnetic measurement system for RFX-mod upgrade. *Fusion Engineering and Design*, 146:906–909, 2019. SI:SOFT-30.

[9] David J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

[10] Sheldon M. Ross. *A First Course in Probability*. Pearson, Boston, MA, 2014.

[11] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[12] Khalid Sayood. *Introduction to Data Compression, Fourth Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2012.

[13] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley-Interscience, Hoboken, NJ, 2nd edition, 2006.

[14] Giacomo Chiarot and Claudio Silvestri. Time series compression survey. *ACM Computing Surveys*, 55(10):1–32, February 2023.

[15] Przemysław Skibiński, Szymon Grabowski, and Sebastian Deorowicz. Revisiting dictionary-based compression. *Software: Practice and Experience*, 2023.

[16] Shuangshuang Chen and Wei Guo. Auto-encoders in deep learning—a review with new perspectives. *Mathematics*, 11(8), 2023.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[18] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[20] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[23] Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *The Twelfth International Conference on Learning Representations*, 2024.

[24] Jinxin Liu, Petar Djukic, Michel Kulhandjian, and Burak Kantarci. Deep dict: Deep learning-based lossy time series compressor for iot data, 2024.

[25] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A. Saurous. Tensorflow distributions, 2017.

[26] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer, 2018.

[27] Ilya Grebnov. Ilyagrebnov/libbsc. `https://github.com/IlyaGrebnov/libbsc`, May 2022. Original-date: 2011-05-11T06:39:49Z.

[28] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[29] Bhattacharya Sourav Prentow Thor Kjrgaard Mikkel Blunck, Henrik and Anind Dey. Heterogeneity Activity Recognition. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C5689X.

[30] Flavia Huerta and Ramon Huerta. Gas sensors for home activity monitoring. UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C5BS4F.

[31] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. Sz3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2023.

[32] Shubham Chandak, Kedar Tatwawadi, Chengtao Wen, Lingyun Wang, Juan Aparicio, and Tsachy Weissman. Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction, 2020.

[33] Rhys Goldstein, Michael Glueck, and Azam Khan. Real-time compression of time series building performance data. In *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association*, 2011.