Realtime capable turbulent transport modelling using neural networks

by

Karel van de Plassche

29th August 2017

Under supervision of

J. Citrin (DIFFER)

DIFFER
Dutch Institute for
Fundamental Energy Research

TU/e
Technische Universiteit
Eindhoven
University of Technology

Contents

Abstract

Predicting transport in fusion devices is important to interpret and optimize current experiments, and extrapolate to future machines. The purpose of this work is to predict turbulent heat and particle transport in real-time, to be used for fast discharge optimization and realtime control applications. Neural networks were used to emulate a first-principle based turbulent transport model. The training set for the networks consist of a large dataset of 3×10^8 flux calculations by the quasilinear gyrokinetic code QuaLiKiz, generated over a 9D input space. A Feed-Forward Neural Network was then trained on this data, reproducing the in- and output mapping of QuaLiKiz, but orders of magnitude faster. The neural network hyperparameters and pipeline were optimized for Electron Temperature Gradient (ETG) driven heat transport on a reduced 7D dataset. The ETG networks reached an RMS error of 5 %. Initial tests show similar results for the full 9D space and various other turbulence regimes. This indicates that generalization of the optimized training pipelines to the full dataset will be straightforward. The accuracy of these turbulent predictions at this speed is unprecedented, and opens new avenues in the modelling of fusion experiments.

Introduction

One of the possible solutions to reduce our dependency on fossil fuels is thermonuclear fusion. Fusion aims to provide CO2-free electricity generation at high-energy-density, leading to a small ecological footprint relative to power produced. The furthest developed thermonuclear fusion concept is the tokamak. Nuclear fusion needs a extremely high temperature to allow the fuel to fuse together and release energy. In a tokamak, the fusion fuel is heated, and the resulting plasma is confined in strong magnetic fields. The easiest attainable reaction is that of deuterium (D) and tritium (T), requiring temperatures of around 100 million Kelvin. The D-T fusion products are helium and a neutron, with a reaction energy of 14.1 MeV, eventually heating the walls and powering the electricity conversion cycle.

Maintaining the plasma at these temperatures is difficult, as instabilities and turbulence drive transport, leaking energy out of the plasma. An efficient and cost-effective reactor demands a detailed understanding of the transport processes, to optimize reactor plasma parameters. Furthermore, fast and accurate prediction of transport is needed to control the reactor plasma, as direct measurements of plasma parameters is complicated due to severely limited diagnostics [1]. The turbulent transport is driven by plasma temperature and density gradient[2], and is described by diffusive and convective transport coefficients. Calculating and characterising these transport coefficients is one of the main objectives of transport theory [3]. In the ideal case, direct control over these coefficients would give direct control of the density and temperature profiles, and thus over the fusion power and efficiency. In this work, we present a way to calculate turbulent transport in real-time.

A major branch of tokamak transport theory is direct numerical simulations of the governing processes. A significant challenge in these simulations is timescale separation; in a fusion plasma there are many different processes each happening at a different typical timescales. The macroscopic evolution of plasma profiles (temperature and density), evolve at a typical scale of 1 Hz to 10 Hz, and the underlying instabilities that drive turbulence have frequencies of 10 kHz to 100 kHz. However, the quickest process in the plasma, the electron gyro motion or cyclotron motion, evolves 7 orders of magnitude quicker, around 100 GHz. An overview of

the different timescales can be found in figure 1.1. This timescale separation makes solving the full system of equations that describe the fusion plasma computationally complex and expensive, and impossible to solve within reasonable time even on current supercomputers. One of the ways to reduce the computational complexity is averaging over the gyromotion, giving rise to the field of gyro-kinetics; the kinetic theory of charged particle rings.

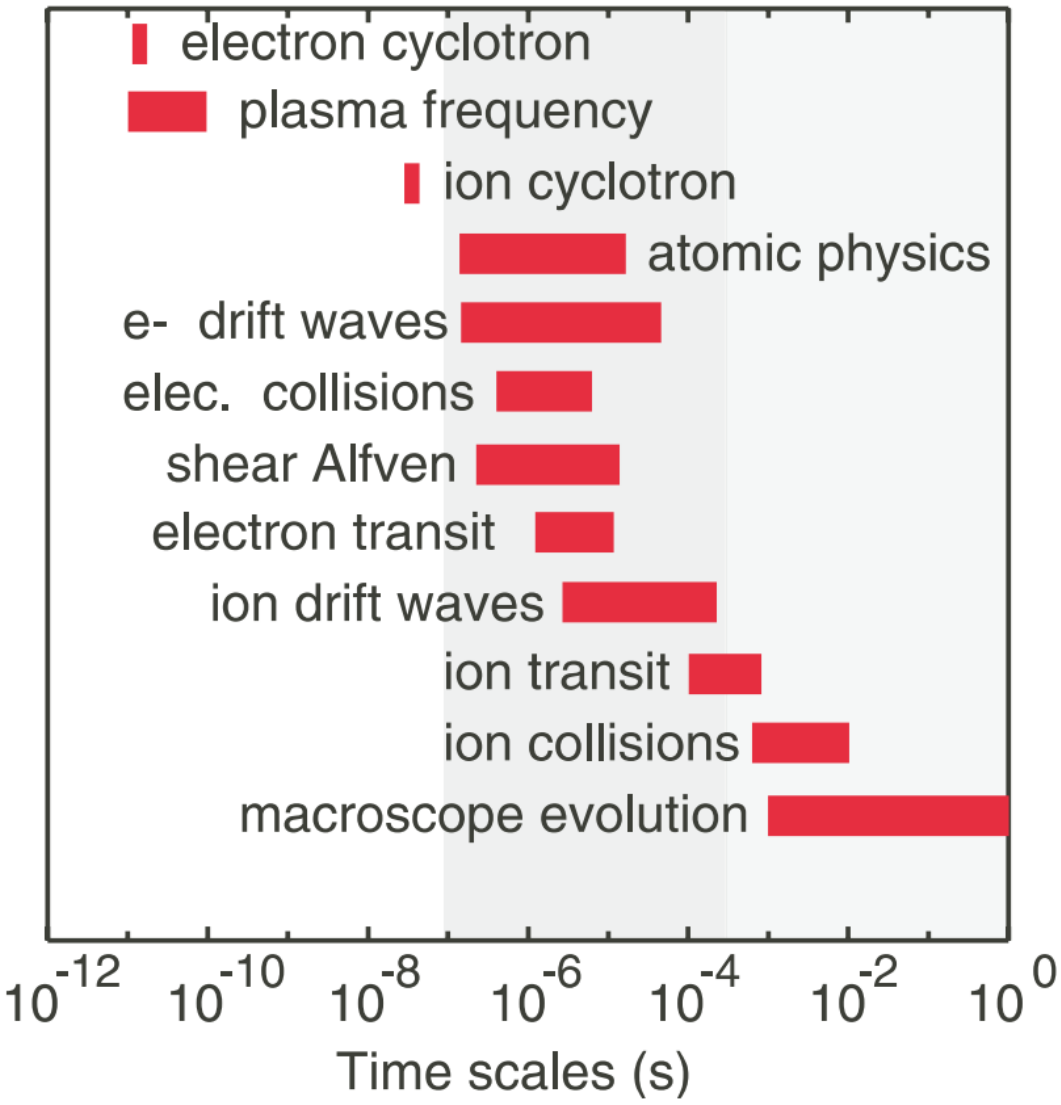


FIGURE 1.1: The different typical timescales in a fusion plasma. Note the seven orders of magnitude difference between the smallest and largest timescales. [4]

Even though the gyrokinetic approximation results in codes that are faster than codes solving the full system, they are still very computationally expensive. Finding the transport coefficients at one moment in time still takes up to 10 MCPUh[1] [5], whilst still having to make

[1]CPUs or CPU-seconds are a quantity used to measure code run-time in computational science. A code that takes 4 CPUs would take 4 s when ran on a single CPU core, while it would take (assuming perfect parallelization) 1 s when ran on four cores

assumptions that reduce the validity for direct comparison with experiments. Ideally, a real-time controller would need to know these coefficients at the timescale they evolve, so within a few milliseconds, using a decent sized compute node, for example on 24 cores [5]. More assumptions can be made to further reduce complexity, and so reduce the computational cost to run these codes.

The main code used in this project is QuaLiKiz[6]. This code is able to reproduce radial profiles in about 200 CPUs by making extra assumptions and simplifications, which will be described in chapter 2. This is an impressive eight orders of magnitude speedup compared to the slowest codes, while still being valid especially in the tokamak core. While aspects of the QuaLiKiz model still need improvement compared to full nonlinear predictions, the accuracy of predictions is sufficiently high such that the code has been successfully used in the reproduction of JET[6–8], ASDEX-U, and Tore-Supra temperature and density profiles[5, 9], and has been integrated in the CRONOS [10] and JETTO [11, 12] suites for integrated tokamak modelling. While it is still five orders of magnitude too slow to use for a real-time controller, it is fast enough to construct large-scale database of code in-output mappings. For example, of transport coefficient evaluations throughout a wide parameter space. This database can then be used to train a neural network to reproduce the QuaLiKiz in-output mapping. The neural network evaluation is then realtime capable. This is the focus of this work.

A neural network, or more specifically, a feed-forward neural network, is a nonlinear function that can approximate mapping of a (nonlinear) multi-dimensional space onto another multi-dimensional space [13, 14]. In other words, it can approximate and thus 'bypass' the calculation of transport coefficients from arbitrary input parameters with a code like QuaLiKiz. Neural networks are used in many fields, also in fusion [15, 16]. The relationship between in- and output is condensed into a single (analytical) function. The time needed to evaluate this function is in principle not dependent on the system being approximated, and can be many orders of magnitude faster. In a proof-of-principle with a 4D input space and 1D output space, the calculation of a full radial profile of transport coefficients was found to be around 1 CPUms, with a RMS error of only 4.2 % [17]. With this final step, the calculation is finally in the right order of magnitude for realtime application, for example in the RAPTOR simulation suite [18]. An overview of the completion time-scales can be found in table 1.1.

TABLE 1.1: Overview of different types of gyrokinetic codes and their estimated times to calculate a full radial profile. Note that the Nonlinear global is for a run with adiabatic electrons, which significantly reduce the validity of direct comparisons with experiments.

Gyrokinetic Simulation	Time to calculate profile at single point in time[2]
Nonlinear global (adiabatic electrons)	10 MCPUh[5]
Nonlinear local	1 MCPUh[5]
Linear local	500 CPUh[5]
QuaLiKiz Quasilinear local	300 CPUs[5]
Proof of principle Neural Network	1 mCPUs[17]
Real-time controller	$\mathcal{O}$(1 mCPUs)

1.1 Research question

One of the main challenges in fusion is to accurately predict particle and heat transport inside the plasma. This is needed for the interpretation and optimization of current-day experiments. It is also essential to increase the confidence in extrapolation to future devices. Real-time capable transport models would allow first-principle based predictions to be used for plasma control, something not possible up until now. In addition, even non-realtime but very fast models significantly enhances the ability to optimize tokamak discharges. A first proof of principle of a real-time capable first-principle based model has been made by J. Citrin [17], using a neural network to learn the in-output mapping of the QuaLiKiz gyrokinetic quasilinear transport model in a limited subspace. The purpose of this project is to expand this proof of principle to a more complete model, so using in- and output-space. To this end, the following question is proposed:

> Can we accurately [3] predict particle and heat transport coefficients [4] as function of a 9D [5] input space using a feed-forward neural network in realtime[6]?

Multiple steps have to be taken to answer this question.

1. Generate the training and validation set:
   (a) Optimize QuaLiKiz to generate the datasets [7] within reasonable time [8].
   (b) Use a supercomputer to run QuaLiKiz and generate the training and validation set for the neural network.

[2] These times are estimated, as most of these codes do not actually calculate profiles, but calculate seperate points which could be used to form a profile
[3] Relative error of 5% with respect to QuaLiKiz, based on the proof of concept [17]
[4] $\chi_{i,e}$ and $D_{i,e}$
[5] R/L_{T_i}, R/L_{T_e}, R/L_n, q, $\hat{s}$, ε, T_i/T_e, ν^*, Z_{eff}
[6] In the order of 1 mCPUs
[7] Initially a hyperrectangle with $\mathcal{O}(10^9)$ points in 9D + $k_\theta \rho_s$ space
[8] Within the supercomputer allocation of $\mathcal{O}(10^6)$ CPUh

2. Train a 9D neural network:

   (a) Train a neural network with the generated training set.

   (b) Compare the output of the neural network with data from the validation set.

This project will be done at the Dutch Institute for Fundamental Energy Research (DIFFER) under the supervision of J. Citrin. The supercomputer that will be used is Edison from the National Energy Research Scientific Computing Center in Berkeley, on which 2 MCPUh is made available. The QuaLiKiz code, written in FORTRAN, is made available for this project by CEA-Cadarache. Within the framework of the development of the code within this project, it has since been released as open source. The neural network training will be done in Python3.5, initially using the TensorFlow library created by the Google Brain team [19].

1.2 Overview

Chapter 2 treats basic gyrokinetics, as well as the QuaLiKiz assumptions in section 2.2.3. This will give the justification of the choice of parameters in section 2.3. Next, chapter 3 treats the basic concepts of neural network training. In this chapter, quantities or hyperparameters are introduced that will be further discussed in light of the trained neural networks for this work in chapter 5. The tools and dataset created for this work will be discussed in chapter 4.

Turbulent transport

2.1 Transport coefficients

An import measure of tokamak performance is called the fusion energy gain factor, or Q. It is defined as:

$$Q \equiv \frac{P_{fus}}{P_{ext}} \tag{2.1}$$

in which P_{fus} is the fusion power and P_{ext} is the external input power. Clearly one of the ways of increasing this performance is reducing the external power needed to sustain the fusion reaction. A measure for how well energy is confined in the (tokamak) system is the energy confinement time τ_E. A high confinement means that less energy is needed to heat the plasma. Unfortunately the confinement time is limited by the energy transport within the plasma. Thus, to understand and hopefully control and improve tokamak performance, transport has to be better understood.

Fusion power depends on the reaction rate $\langle \sigma v \rangle$, the densities of deuterium n_D and tritium n_T (assuming a D-T tokamak) and the fusion energy per reaction E_{fusion} as:

$$P_{fusion} = n_D n_T \langle \sigma v \rangle E_{fusion} \tag{2.2}$$

Assuming a 50-50 distribution of deuterium and tritium, this equation can be written as:

$$P_{fusion} = \frac{1}{4} n^2 \langle \sigma v \rangle E_{fusion} \tag{2.3}$$

A power-generating tokamak is operated at a specific temperature to maximise the fusion power. On the temperature range of interest, the reaction rate scales as:

$$\langle \sigma v \rangle \propto T^2 \tag{2.4}$$

Resulting in:

$$P_{fusion} \propto n^2 T^2 \propto p^2 \tag{2.5}$$

Equation (2.5) implies that it is beneficial for a tokamak to operate at an as high as possible pressure. However, as the pressure is necessarily zero at the plasma edge, a high pressure implies steep gradients. These gradients drive microinstabilities in the plasma, which give rise to turbulent transport and thus limit the maximum achievable pressure and the confinement time τ_E of the system.

The temperature and density can be directly related using the ion and electron particle flux $\Gamma_{i,e}$, and the ion and electron heat flux $q_{i,e}$ [3, 20] using transport equations. These equations can also be expressed using their transport diffusivities D and χ and pinches V. For heat transport, the pinch is typically small, so it is simplified by using the effective diffusivity χ^{eff}.

$$\Gamma_{i,e} = -D_{i,e}\frac{\mathrm{d}n_{i,e}}{\mathrm{d}r} + V_{i,e}n_{i,e} \tag{2.6}$$

$$q_{i,e} = -\chi^{eff}_{i,e}\frac{\mathrm{d}T_{i,e}}{\mathrm{d}r} \tag{2.7}$$

Calculating these transport coefficients is the objective of (turbulent) transport theory [3].

Detailed knowledge of the transport would enable researchers to better optimize and interpret current experiments, as well as open ways to extrapolate to future machines and aid the design of control systems for future and present-day devices. In this work, the QuaLiKiz code will be used to calculate these transport coefficients. QuaLiKiz is a quasi-linear gyrokinetic solver, which will be further described in section 2.2.3.

2.2 Gyrokinetics

2.2.1 The gyrokinetic assumption

A plasma can be described by the position, speed, and the evolution in time of all individual particles. As the amount of particles in a typical tokamak system is very large, instead an ensemble averaging of the particles is performed. The system is them described by a particle distribution function f_s, where the subscript s is a species index, i.e. ions or electrons. Collisions are included using a collision operator C_s, which gives a probabilistic description of binary collisions. This results in the Fokker-Planck equation [21, 22]:

$$\left[\frac{\partial}{\partial t} + \mathbf{v}\cdot\nabla + \frac{q_s}{m_s}\left(\mathbf{E} + \frac{\mathbf{v}}{c}\times\mathbf{B}\right)\cdot\frac{\partial}{\partial \mathbf{v}}\right] f_s = C_s f_s \tag{2.8}$$

This is a very good assumption for magnetic fusion plasmas with a large number of particles in the Debye sphere. As such, collective effects dominate over (binary) collisions. This equation is known by different names, for example the (collisional) Vlasov equation, the Boltzmann

kinetic equation or just simply the Fokker-Planck equation. It describes the evolution of the distribution function due to flows and forces. In a tokamak plasma, this equation is coupled to Maxwell's equations, with the charges and currents described by the distribution function moments themselves.

Particle and heat fluxes are determined from the evolution of the distribution functions. However, direct numerical simulation of this equation is nearly intractable. One reason for the difficulty of the equation is the dependence of the Fokker-Planck equation on effects that evolve at different timescales spanning over 13 order of magnitudes. Figure 1.1 gives an overview of the different timescales. This makes solving the involved differential equations computationally expensive. Fortunately, the gyrofrequency $\omega_c = \frac{|q|B}{m}$ evolves at a much faster rate (10 MHz to 100 MHz for ions) than the instabilities that are of interest (10 kHz to 100 kHz), an effect called timescale separation [23]. The timescale separation can be removed by averaging over the gyro-phase, giving rise to the field of gyro-kinetics; the kinetic theory of charged particle rings. So called full-f gyrokinetic solve the 5D gyroaverage equation of equation (2.8) coupled to Maxwell's equations directly. Some examples of full-f gyrokinetic codes are GT5D [24] and GYSELA [25].

Even with the previous simplifications, the computational cost to run these codes is still very large. For example, the estimated time to simulate a radial profile of transport coefficients is around 10 MCPUh [5], while the codes are heavily parallelized and can run on thousands of cores simultaneously. Here one CPUh is defined as the amount of time the code would run on one CPU core. This is too long for even a relatively small range of parameters scans. Furthermore, to get to these runtimes major assumptions have to be made, for example adiabatic electrons, so fusion relevant comparisons with experiments are impossible.

2.2.2 Delta-f splitting and locality

More assumptions can be made to reduce the computational cost of solving equation (2.8) without reducing the accuracy of the model. In the core of tokamak plasmas, the fluctuation of the particle density function is relatively small compared to the equilibrium distribution; only in the order of a few percent for $r/a \leq 0.8$ [26]. This allows equation (2.8) to be expanded with respect to a small fluctuation parameter δf describing the dynamics with a static background Maxwellian particle distribution f_0. In nonlinear simulations, only the leading nonlinear term describing the ExB drift is retained. In linear codes all nonlinear terms are neglected

Additionally, δf codes assume locality. This assumption is valid when the gradient lengths $(\frac{B}{\nabla B}, \frac{T}{\nabla T}, \frac{n}{\nabla n})$ are large compared to the size of the turbulence length scale, the size of the turbulent eddies. The Larmor radius with respect to the sound speed ρ_c is a good estimation

of the size of turbulent eddies [27]:

$$\rho_c = \frac{m v_{c,\perp}}{|q|\, B} \tag{2.9}$$

And the gradient length scales are given by:

$$L_B \equiv -\frac{B}{\nabla B} \tag{2.10a}$$

$$L_{T_{i,e}} \equiv -\frac{T_{i,e}}{\nabla T_{i,e}} \tag{2.10b}$$

$$L_n \equiv -\frac{n}{\nabla n} \tag{2.10c}$$

This assumption is justified for tokamaks equal or larger than present-day medium sized tokamaks, for example WEST, Asdex Upgrade and TCV. This means that the background plasma parameters in the simulation, e.g. temperature and density gradients, q-profile, etc., can be taken constant within the simulation domain representing a specific radial location. This allows for a spectral formulation in the spatial directions perpendicular to the magnetic field, saving a lot of calculation time.

Some examples of so called δf gyrokinetic codes are GENE [28] and GYRO [29]. Even though these simulations are orders of magnitude faster than full-f simulations, they are still too expensive for our purposes. The estimated time to simulate a radial profile of transport coefficients is still around 1×10^6 CPUh for non-linear runs (Although including more comprehensive physics such as kinetic electrons and magnetic fluctuations) or 500 CPUh for linear runs. Again, these codes are parallelized to a large degree.

2.2.3 QuaLiKiz

Gyrokinetic modelling can be sped up even further by making more assumptions, at the cost of losing accuracy. These models are known as reduced models. In this section, the most important assumptions of the reduced model used for this work, QuaLiKiz, are explored. However, a full derivation is beyond the scope of this work. For a full treatment, see [5, 30–33]. This step results in another major speedup, needing only around 300 CPU − seconds for a single profile calulation, or around 50 CPUh for the evolution of one second of JET plasma [6].

2.2.3.1 Electrostatic limit

If we assume no magnetic perturbations, corresponding to the electrostatic limit. This approximation is valid for low β plasmas [5]. We can then write the linearised (single-species)

Vlasov equation as:

$$\delta f_{s,k} = \frac{q_s}{T_s} f_{s,0} \left[1 - \frac{\omega_k - \omega^*_{s,k}}{\omega_k - k_\parallel v_{s,\parallel} - \omega^d_{s,k}} J_0 \left(k_\perp \rho_s \right) \right] \delta\phi_k \tag{2.11}$$

Here $f_{s,0}$ is the, assumed to be Maxwellian, equilibrium distribution function. $\delta\phi_k$ is the pertubative potential, defined by $\delta E_k = -ik_\theta \delta\phi_k$. Only the poloidal electric field is treated as it dominates the fluctuating $E \times B$-drift, which sets the transport levels. q_s, T_s, and $v_{s,\parallel}$ are respectively the particle charge, temperature, and parallel velocity. $k_\parallel$ and $k_\perp$ are the parallel and perpendicular wavenumber (with respect to the magnetic field line). $\omega^d_{s,k}$ and $\omega^*_{s,k}$ are the particle drift and diamagnetic frequency, and are treated in section 2.2.5. J_0 is a Bessel function of the 0-th order, which appears from the gyroaveraging in Fourier space. ρ_s is the Larmor radius for the species under consideration, weighted to the sound speed. Finally, ω_k is the complex frequency of the perturbation, the variable of interest. Here $\omega_k = \omega_{r,k} + i\gamma_k$ where $\omega_{r,k}$ is the real mode frequency, and γ is the growth rate.

2.2.3.2 Dispersion relation for calculating instabilities

To get the full characteristics of the perturbation, this equation has to be coupled to Maxwell's equations. In the electrostatic limit, this can be done by coupling the weak formulation of quasineutrality $\iiint \sum_s q_s \delta n_{s,k} \phi^*_k \, \mathrm{d}^3 \mathbf{r} = 0$ [34]:

$$\sum_s \mathcal{L}_{s,t} + \mathcal{L}_{s,p} = 0 \tag{2.12}$$

Where $\mathcal{L}_{s,t}$ and $\mathcal{L}_{s,p} = 0$ are the trapped and passing particle integrals for each species. These integrals have to be treated separately, as only the passing particles have a finite average transit frequency $k_\parallel v_{\parallel,s}$. The trapped species are bounce averaged, so $v_{\parallel,s} = 0$. The trapped particles also have an increased orbit width δ_s, the banana width, due to their bouncing motion. Assuming toroidal symmetry gives as integrals [33]:

$$\mathcal{L}_{s,t} = \left\langle \int_{-\infty}^{+\infty} \frac{\mathrm{d}k_r}{2\pi} J_0^2(k_\perp \rho_s) J_0^2(k_r \delta_s) \left| \tilde{\phi}(k_r) \right|^2 \times \frac{\omega_k - n\omega^*_{s,k}}{\omega_k - n\omega^d_{s,t} + i\nu} \right\rangle_t \tag{2.13}$$

$$\mathcal{L}_{s,p} = \left\langle \iint_{-\infty}^{+\infty} \frac{\mathrm{d}\theta \, \mathrm{d}x}{\pi} J_0^2(k_\perp \rho_s) \left| \tilde{\phi}(x,\theta) \right|^2 \frac{\omega_k - n\omega^*_{s,k}}{\omega_k - n\omega^d_{s,p} - k_\parallel v_{s,\parallel} + i0^+} \right\rangle_p \tag{2.14}$$

With the integrals $\langle \dots \rangle_{t,p}$ the velocity space integration in terms of energy $\mathcal{E}$ and pitch angle λ:

$$\langle \dots \rangle_t = \int_0^{+\infty} \frac{2}{\sqrt{\pi}} \sqrt{\mathcal{E}}\, \mathrm{d}\mathcal{E} \int_{\lambda_c}^1 \frac{\mathrm{d}\lambda}{4\bar{\omega}_b} \tag{2.15}$$

$$= f_t \int_0^{+\infty} \frac{2}{\sqrt{\pi}} \sqrt{\mathcal{E}}\, \mathrm{d}\mathcal{E} \int_0^1 K(\kappa)\kappa\, \mathrm{d}\kappa \tag{2.16}$$

$$\langle \dots \rangle_p = \int_0^{+\infty} \frac{2}{\sqrt{\pi}} \sqrt{\mathcal{E}}\, \mathrm{d}\mathcal{E} \int_0^{\lambda_c} \frac{\mathrm{d}\lambda}{4\bar{\omega}_b} \tag{2.17}$$

The elliptic integral $K(\kappa)$ comes from the averaging over the trapped particle bounce orbit. It is further treated in sections 2.2.5 and 4.1. $\bar{\omega}_b$ is related to the poloidal orbit frequency. ν is the collisionality. The only unknowns to be solved are ω and the poloidal eigenfunction $\tilde{\phi}$. In less simplified codes this eigenfunction is solved fully self-consistently. In QuaLiKiz it is solved via a separate method to save computing time, expanding the dispersion relation in small $\frac{k_\parallel v_\parallel}{\omega}$ and ω^d/ω, see section 2.2.3.4. All other variables are input parameters, or can be re-written or approximated as detailed in the next sections.

2.2.3.3 Mode structure

In a tokamak, the spatial structure of instabilities tends to be highly uniform along field lines, so $k_\parallel \ll k_\perp$. Physically it's related to field-line bending being energetically unfavourable. The uniformity is verified by nonlinear gyrokinetic simulations [35]. This justifies the following assumptions regarding the structure of the eigenfunction. Assume a spatially slowly-varying *envelope* and strongly varying *eikonal*, giving:

$$\phi = \phi(\theta, r) e^{\left(in[\phi - q(r)\theta]\right)} \tag{2.18}$$

where n is the toroidal mode number ϕ is the toroidal angle and θ is the poloidal angle. Along the field line of the instability being analysed, the eikonal is constant. Since the eikonal spatial variation is much faster than the envelope, the wavevector $\mathbf{k}$ can be approximated by defining it only with respect to the eikonal, and can be written as [30]:

$$k_\theta = \frac{nq}{r} = \frac{nq'}{\hat{s}} \tag{2.19a}$$

$$k_x = nq'\theta = k_\theta \hat{s}\theta \tag{2.19b}$$

$$k_\parallel \approx \frac{nq'x}{qR} = k_\theta \frac{\hat{s}x}{qR} \tag{2.19c}$$

These wavenumbers depend on two magnetic quantities, the safety factor q and its derivative the magnetic shear $\hat{s}$:

$$q = \frac{\mathrm{d}\phi}{\mathrm{d}\theta} \tag{2.20}$$

$$\hat{s} = \frac{r}{q}\frac{\mathrm{d}q}{\mathrm{d}r} \tag{2.21}$$

Where the safety factor represents the number of toroidal revolutions performed by a field line for one poloidal revolution.

One challenge now becomes apparent. The integral in equation (2.14) now contains x via the $k_\parallel$ term. Unfortunately, this integral can not be further simplified, so this is the main speed limitation to the QuaLiKiz code.

2.2.3.4 Gaussian eigenfunction ansatz

A major speedup is gained by an approximated solution of the eigenfunction. This is carried out by assuming the driving frequency ω is much larger than the particle transit frequency $k_\parallel v_{\parallel,s}$ and the particle drift frequency ω_d [36]. Furthermore, it is assumed the wave vectors are small, such that $k_\perp\rho_e \ll k_\perp\delta_e \ll k_\perp\delta_i \ll k_\perp\delta_i \ll 1$. Furthermore, it is assumed that the eigenfunction solution is a shifted Gaussian, such that:

$$\phi_{n\omega}(x) = \frac{\phi_0}{\left(\pi\Re(w^2)\right)^{1/4}} e^{-\frac{(x-x_0)^2}{2w^2}} \tag{2.22}$$

With these assumptions, one can expands the dispersion relation, and find a system of polynomial equations from which the width w, mode shift x_0, and eigenfrequency ω are calculated.

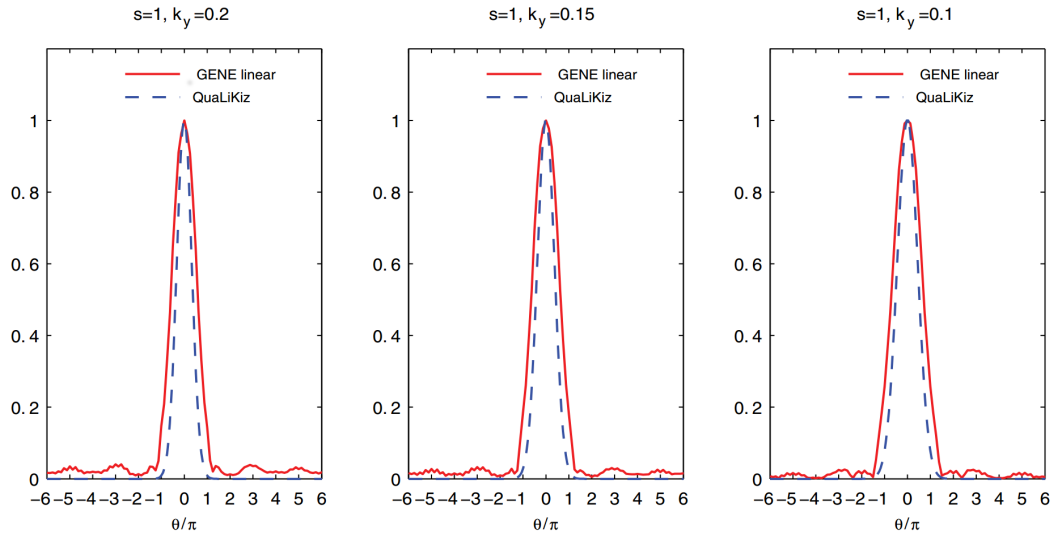


FIGURE 2.1: Some examples of the solution of the eigenfunction in QuaLiKiz and GENE. As can be seen, the QuaLiKiz approximation matches the more complete GENE solution fairly well. Figure taken with permission from [33].

2.2.3.5 Quasi-linear assumption

From the reduced gyrokinetic equation one gets the eigenfrequencies ω. These eigenfrequencies are then used to calculate the particle and heat fluxes. In QuaLiKiz, the quasilinear assumption is used to do this. This assumption has been extensively validated by comparisons with nonlinear simulations [31, 33, 37]. Assume the fluxes can be decomposed in a linear response R to the instabilties and the saturated electrostatic potential, as such:

$$\Gamma, Q = \sum_{k_\theta, \omega_{kj}} \Im \left(R_{lin}^{\Gamma,Q}(k_\theta, \omega_{kj}) \right) \left| \delta\phi(k_\theta, \omega_{kj}) \right|^2 \tag{2.23}$$

The linear response is related to moments of equation (2.11). The zeroth moment is related to density and particle transport, and the second to temperature and heat transport. This response is then evaluated at the calculated eigenfrequencies ω_k of which there can be several per ω_k, which leads to ω_{kj}.

Finally, the amplitude and spectrum of ϕ_0 from equation (2.22) is set by a so called saturation rule. In the case of QuaLiKiz, it is given by a model to capture the nonlinear ϕ spectrum based on simple analytical formulae tuned to nonlinear simulations. [33, 37].

2.2.4 Overview of QuaLiKiz assumptions

Summarizing, these are the main assumptions of the QuaLiKiz code.

1. Low Mach number $\frac{U_\parallel}{v_\parallel} \ll 1$
2. Electrostatic limit (section 2.2.3.1)
3. Eigenfunction is assumed to be Gaussian (section 2.2.3.4)
4. Shifted circle geometry $(s - \alpha)$ with small inverse aspect ratio
5. For passing particles: since $v_\parallel >> v_\perp$, a pitch angle averaged transit frequency as well as curvature and ∇B drift frequencies are performed. (equation (2.14))
6. For trapped particles: since the bounce frequency is larger than ω, a bounced average is performed. (equation (2.13))
7. Only collisions for trapped electrons are taken into account using a Krook type collision operator. (equation (2.8))

2.2.5 Physical implications

Now that the full system has been described, it is insightful to look at some physical implications of the formulas presented in the previous sections. The background distribution function gradients drive the pertubation and thus the instabilities, and they are represented in the diamagnetic frequency ω^*. The term appears in both equation (2.14) and equation (2.13), and can be written as:

$$\omega_s^* = -k_\theta \cdot \frac{T_s}{q_s B R}\left[\frac{R}{L_{n_s}} + \frac{R}{L_{T_s}}\left(\frac{m_s v_s^2}{2T_s} - \frac{3}{2}\right)\right] \tag{2.24}$$

One should note the appearance of the gradient length L_n and L_T, which also appear in the transport equations (2.6) and (2.7). This leads to the nonlinearity of the transport equation, since χ and D are functions of L_n and L_T themselves.

Another quantity appearing in the gyrokinetic system of equations is the trapped particle vertical drift frequency. This is the ∇B drift that leads to charge separation. It can be written as:

$$n\omega_t^d = -\frac{k_\theta T_s}{q_s B} f(\kappa) \tag{2.25}$$

Where:

$$f(\kappa) = 2\frac{E(\kappa)}{K(\kappa)} - 1 + 4s\left(\kappa^2 - 1 + \frac{E(\kappa)}{K(\kappa)}\right) \tag{2.26}$$

Where $E(\kappa)$ and $K(\kappa)$ are first and second order complete elliptic integrals. These integrals took up a significant fraction of the QuaLiKiz run time, but has been improved in this work, see section 4.1. The quantity κ is related to the pitch angle through:

$$\lambda \equiv \frac{v_\perp^2}{v_\parallel^2 + v_\perp^2} = 1 - 2\varepsilon\kappa^2 \tag{2.27}$$

Finally, the passing particle vertical drift frequency can be written as:

$$n\omega_p^d = -\frac{k_\theta T_s}{e_s B}\left(\cos\theta\left(\hat{s}\theta - \alpha\sin\theta\right)\sin\theta\right) \tag{2.28}$$

The formulas presented in this section show how the parameters chosen to investigate in this work relate to physical frequencies within the tokamak plasma.

2.2.6 GyroBohm normalization

Heat- and particle fluxes in QuaLiKiz are normalized to GyroBohm units, denoted by $[GB]$. For the remainder of this report, all fluxes are in GyroBohm units. They are related to their SI equivalents as:

$$\chi_{GB} \equiv \frac{\sqrt{m_s}}{q_s^2 B_0^2}\frac{T_s^{1.5}}{a} \tag{2.29}$$

$$q_{GB} \equiv \frac{q_{SI}}{\chi_{GB}}\frac{R_0}{n_s T_s} \tag{2.30}$$

$$\Gamma_{GB} \equiv \frac{\Gamma_{SI}}{\chi_{GB}}\frac{R_0}{n_s} \tag{2.31}$$

2.2.7 Expected effect of input on unstable modes

Turbulent transport is governed by the modes in the plasma that are unstable, of which the low-frequency ones dominate transport [2, 38]. Generally three of those modes are considered in transport models: the Ion Temperature Gradient mode (ITG)[39, 40], the Electron Temperature Gradient mode (ETG)[41], and the Trapped Electron Mode (TEM) [42, 43]. The Trapped Ion Mode (TIM) are usually suppressed by nonlinear effects [44–46]. QuaLiKiz can differentiate between these three modes based on the characteristic wavelength $k_\theta\rho_s$ and real angular frequency $\omega_{r,k}$, as shown in table 2.1. Postive real frequencies in QuaLiKiz are defined to be in the electron diamagnetic direction.

TABLE 2.1: The criteria on which QuaLiKiz determines the type of mode.

Mode	$k_\theta\rho_s$	$\omega_{r,k}$
TEM	≤ 2	> 0
ETG	> 2	> 0
ITG	≤ 2	< 0

This split in modes makes analysis easier, as the three modes react differently to changes input parameters. Table 2.2 summarizes these influences based on the previous discussion of the

dispersion relation. However, the influence on the eigenfunction solution is outside the scope of this analysis. In this table, in addition to the already discussed variables, the following definitions are used.

The effective ion charge:

$$Z_{eff} \equiv \frac{\sum_i n_i Z_i^2}{\sum_i n_i Z_i} = \frac{\sum_i n_i Z_i^2}{n_e} \tag{2.32}$$

The local MHD alpha:

$$\alpha = q^2 \sum_s \beta_s \left(\frac{R}{L_{T_s}} + \frac{R}{L_{n_s}} \right) \tag{2.33}$$

And the rotational flow shear:

$$\frac{\mathrm{d}V_\perp}{\mathrm{d}r} \tag{2.34}$$

β_s is the plasma-β. The normalized collision frequency ν^*, related to the collisionally ν, normalized by the trapped electron bounce frequency.

TABLE 2.2: Various QuaLiKiz input parameters and their expected effect on ETG, ITM and TEM instabilities. In this table, a $-$ denotes a stabilizing effect, a $+$ a destabilizing effect and a o a negligible effect. The influence of these parameters on the eigenfunction solution is outside the scope of this analysis.

		Effect on		
Effect of increase	Influences	ETG	ITG	TEM
R/L_{T_i}	ω^*	o	+	o
R/L_{T_e}	ω^*	+	o	+
R/L_n	ω^*	o	+	+
q	$k_\parallel$ magnitude	+	+	+
$\hat{s} \neq 0.5$[1]	ω^d	-	-	-
T_i/T_e	Relative weight linear response	+	-	+
ε	Increases trapped electron fraction	o	+	+
ν^*	Detraps electrons	o	-	-
Z_{eff}	Dilutes main ions	-	-	o
α	Shafranov shift, impacting ∇B drift vertical drift frequency	-	-	-
$\mathrm{d}V_\perp/\mathrm{d}r$	shears eddies	o	-	-

Two variables are not explored in this work: α and $\mathrm{d}V_\perp/\mathrm{d}r$. The effect of α can be roughly approximated by shifting $\hat{s}$, and so computation time is saved by not scanning this variable. Furthermore, the impact of α at low magnatic shear is not validated in QuaLiKiz [6]. To calculate the effect of $\mathrm{d}V_\perp/\mathrm{d}r$, QuaLiKiz has to be run with rotation. Running with rotation increases the runtime by a factor 4, which would increase the computational time beyond the

scope of this work. As a workaround, the effect of flow-shear can be captured in post-processing using Waltz-rule like techniques [47, 48]. However, it is out of scope for this work.

2.3 Real-time capable gyrokinetic-based model

Reduced models enable gyro-kinetic calculations for a reasonable computational cost. In the case of QuaLiKiz, one second of plasma evolution in the JET tokamak can be calculated in about 100 CPUh. While impressive, this is still impractical for discharge optimization, and too long for real-time simulation and control. In this work, the QuaLiKiz model is approximated by using a neural network regression, see chapter 3. The result of this regression is an analytical function, which can be evaluated within tens of microseconds, allowing for a full radial profile of all transport coefficients to be calculated within a millisecond [17].

A neural network needs a large database of code in-and output to train, and is only accurate within the parameters contained in the dataset. So, ideally, the dataset should contain parameters of interest for which this model will be used. This usually means experimentally relevant ranges. However, a full analysis of the experimentally relevant subspace is beyond the scope of this work, although this is planned for future work [49]. Instead, the dataset is based on the limits of parameters found by experience in gyrokinetic modelling of experiments. By then scanning a regular hyperrectangle between these ranges, it is assumed the relevant experimental values are captured within this dataset. The database and other methods in this work are modular. If it is found in future work that more training points are needed, for example in important regions in parameter space, the neural network can be retrained. As such, by using the tools developed in this work, one can easily retrain the network with new data.

The hyperrectangle of input parameters contained in the database can be found in table 2.3. The most important parameters are the gradients, as is clear from the analysis in this chapter and experience. Then, a wide range of wavenumbers is needed to sufficiently cover both ion and electron lengthscales. As such these parameters are scanned most densely. Also important are the safety factor q and shear $\hat{s}$, which were also used in the proof-of-principle [17]. The other parameters are less important, and so are scanned less densely. Note that the full subspace of the proof-of-concept is contained within this dataset.

TABLE 2.3: Description of the hyperrectangle of QuaLiKiz input parameters. The ranges of the rectangle are based on experience with gyrokinetic modelling of experimental tokamak plasmas, and are expected to cover the relevant range for core turbulent transport.

variable	# points	min	max
$k_\theta \rho_s$	18	0.1	36
R/L_{T_i}	12	0	14
R/L_{T_e}	12	0	14
R/L_n	12	-5	6
q	10	0.66	15
$\hat{s}$	10	-1	5
ε	8	0.03	0.33
T_i/T_e	7	0.25	2.5
ν^*	6	1×10^{-5}	1
Z_{eff}	5	1	3
Total	3×10^8	≈ 1.3 MCPUh	

Machine Learning

In the past few years machine learning has gained increased interest from researchers and companies alike. Machine learning is a subfield of computer science that gives computers the ability to learn without being explicitly programmed [50]. It has been applied from Google DeepMind's AlphaGo beating the world champion in the game Go [51], to researchers finding novel ways to detect breast cancer [52]. In the context of fusion, machine learning has been used as early as the 1990s [15]. For example, in the prediction of energy confinement scaling [16] and the extraction of plasma equilibrium parameters [53]. Recent increases in processing power have facilitated new opportunities and renewed interest. For example, progress has been made on the regression of DIII-D heat fluxes from an experimental power balance database [54], and on emulating a multi-machine database of the EPED1 [55] pedestal stability code output. In this work, machine learning techniques are applied to emulate the QuaLiKiz code, to allow for sub-millisecond gyrokinetic modelling of turbulent fluxes.

3.1 Neural networks

Neural networks are a class of machine learning models often used for pattern recognition and regression applications. A neural network is a collection of nodes or functions, commonly called neurons, that are connected to each other in a specific way, as described in section 3.1.2. This work focusses on a specific class of neural networks, the feedforward neural network (FFNN).

3.1.1 Neurons

The basic building block of an FFNN is the neuron. In mathematical terms, a neuron is described by multiplying a weight w_i to the individual inputs x_i [56]. Then, all results are summed together and a bias b is applied. Finally a so called activation function f is applied,

such that:

$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right) \tag{3.1}$$

Historically this activation function has been a bounded, nonlinear function, which when applied acts as a binary classifier. This was inspired by the characteristics of biological neurons. It can be represented as a directed graph in the following way:

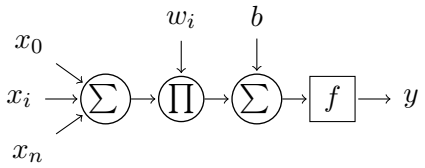


FIGURE 3.1: Flow diagram of a single neuron

Or more condensed:

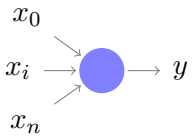


FIGURE 3.2: Schematic representation of a single neuron

3.1.2 Feedforward Neural Networks

A feedforward neural network is formed by combining neurons in layers, such that every layer is fully connected with the next one. In a feedforward network, only connections with the next layer are allowed, but connections within the layer, or loops to previous layers are not. The neurons within a single layer are fully parallel and thus don't interact. Typically an FFNN consists of an input layer, one or more hidden layers and an output layer. An example of an FFNN with four inputs, a single output and a single hidden layer can be found in figure 3.3.

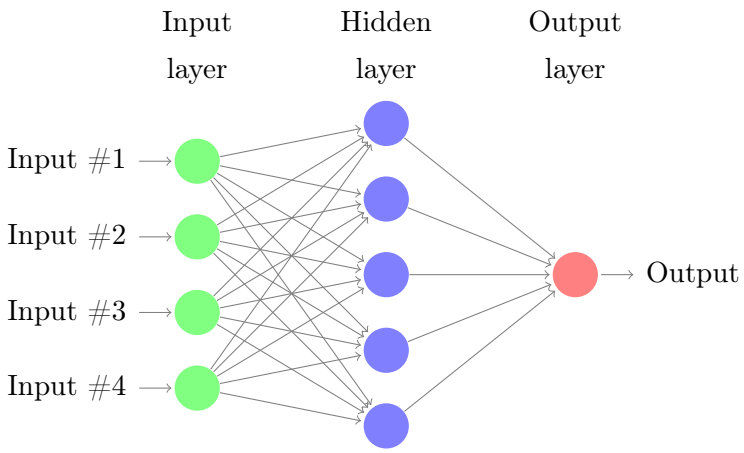


FIGURE 3.3: Schematic overview of a single layer Feed-Forward neural network.

In essence, an FFNN is simply a matrix equation that creates a nonlinear mapping between an input vector $\mathbf{x}$ to an output vector $\mathbf{y}$, albeit one with many free parameters in the form of the weights $\mathbf{W}$ and biases b:

$$\mathbf{y}(\mathbf{x}, \mathbf{W}, \mathbf{b}, f) \tag{3.2}$$

FFNNs are widely used, because an FFNN using a constant, bounded, and monotonic-increasing activation function with one hidden layer can approximate any continuous function up to an arbitrary error. This property is commonly known as the universal approximation theorem [57].

3.1.3 Activation functions

For the FFNN to act as an approximator, the activation function has to be a constant, bounded, and monotonic-increasing function. Moreover, for convergence reasons, it is useful to use a function that results in a gradual change in output for a gradual change in input [58]. Usually a computationally cheap function is chosen, as the activation function is evaluated very often during neural network training, at least once per neuron for every training sample. The gradient of the outputs with respect to the input are also of interested in this work, as they are needed for fast integrated modelling. The gradients are vital for use in real-time capable solvers such as RAPTOR [18] and for trajectory optimization [17]. For these cases, it is best to use functions with easily determined derivatives. One class of functions satisfying all aforementioned criteria are the sigmoid functions. As such, sigmoids are the main focus of this work. A sigmoid function is a mathematical function having an 'S' shaped curve. One common formulation is the following:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

with its derivative:

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right) \tag{3.4}$$

It has been found that a sigmoid centered around zero oftentimes converges in less optimization steps [59]. This results in:

$$\sigma(x) = \frac{2}{1 + e^{-2x}} - 1 \equiv \tanh x \tag{3.5}$$

As such, this is the function used in both the QuaLiKiz neural network proof of principle and in this work.

3.1.4 Neural Networks as QuaLiKiz emulator

In this work, the ability to approximate any function of an FFNN is used to emulate the QuaLiKiz code. The input $\mathbf{x}$ is a 9D vector of local plasma parameters most relevant for turbulent transport, as discussed in section 2.3. The output vector $\mathbf{y}$ is the transport coefficient or flux of interest, i.e. heat fluxes, particle diffusivities and convective terms. The activation function f is defined in equation (3.5), and the weights and biases are set through an optimization algorithm, as described in section 3.2.2.

It is important to note that the approximation theorem is only an existence theorem, it does not guarantee that the training converges within finite time. Neither does it say anything about the optimal network topology (number of layers, neurons per layer) or activation function needed. As training a neural network is generally a non-convex, highly-dimensional problem, it is impossible to make general claims about convergence. While theoretical proofs exist for very specific cases, it is usually sufficient to use heuristics to reach convergence within reasonable time [59]. Unfortunately, the effectiveness of these heuristics are very problem specific. One such heuristic is that a single FFNN layer is generally insufficient. Two layers give better results, and simple problems usually do not improve much by increasing the amount of layers. Networks with many layers have had a lot of success solving complex problems [60]. These networks are called deep networks, the topic of deep learning, However, they are harder to train and are computationally more expensive to evaluate. In light of the promising earlier results [17, 54, 55] we focus on using shallow neural nets in this work.

3.2 Training

With the shape of the neural network and the activation function defined, the next step is to train the neural network. In other words, determine the weights and biases such that any input x gives the correct output y. To this end, a dataset of 3×10^8 QuaLiKiz runs has been generated, see chapter 4. Each run gives a single input-output relation made by QuaLiKiz.

The weights and biases of the neural network then have to be optimized such that the input-output mapping is reproduced. As the mapping between QuaLiKiz input and output is known this makes the training so-called supervised learning. The most widely used [13] supervised learning algorithm is called backpropagation. Backpropagation works by first defining a loss function[1], whose minimization is the basis of the optimization of the weights and biases. The network has finished training as soon as the minimum loss has been found.

3.2.1 Loss function

The loss function should represent the goodness of the fit, and should be a function of the outputs of the neural network. A common goodness measure L_{good} for regression is the mean squared error (MSE), which is also used in this work:

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{3.6}$$

Or, alternatively the mean absolute error (MABSE):

$$L_{MABSE} = \frac{1}{n}\sum_{i=1}^{n}\left|(\hat{y}_i - y_i)\right| \tag{3.7}$$

Here, $\hat{y}$ is the output of the neural network, y the target value as given by the samples and n the number of samples. In this work, a regularisation term is also included. Training without regularisation tends to cause overfitting, caused by the many free variables of the neural network. Overfitting results in a network that reproduces the training set well, but is unable to generalize outside the training set. This means that the fluxes between two data points will vary wildly, resulting in a severe over or underestimation of the real flux. Moreover, it results in gradients flipping signs where they should be constant, pushing the system in an unphysical regime when used in integrated modelling. Figure 3.4 contains a simplified example of overfitting. In this picture one can find the datapoints of a linear function with gaussian noise. The complex model is represents a result of a non-regularized network, while the simple model has very high regularization.

[1] Sometimes called cost or error function

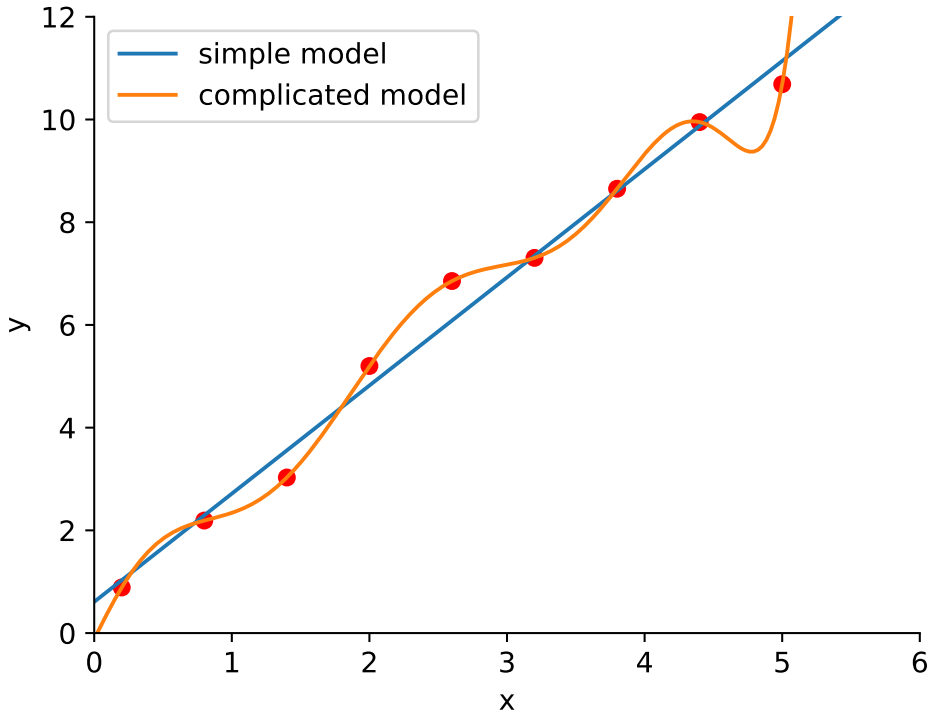


FIGURE 3.4: A simplified example of overfitting. The datapoints are generated on the line $y = 2x+1+noise$ with gaussian noise. Then, a polynomial function is fitted on the data, the simple one representing the regularized case ($y = \sum_{i=0}^{i\leq 1} a_i x_i$) and a complex one representing the non-regularized case ($y = \sum_{i=0}^{i\leq 9} a_i x_i$). Even though the non-regularized fit matches the data points exactly, it does not represent the most probable model. Regularization is needed for this reason, although the implicit assumption is that the simpler model is the best model. In this work however, it is known that the training variables should be smooth functions of the input, so this assumption is justified.

To prevent this, a function that represents the regularization L_{regu} is added to the loss function. One commonly used function is the L2 norm. The L2 norm penalises large weights, a common sign of overfitting. This results in:

$$L_{L2} = \frac{1}{2} \sum_{i,j} w_{i,j}^2 \tag{3.8}$$

Alternatively, one can use the L1 norm, which also penalises large weights:

$$L_{L1} = \sum_{i,j} |w_{i,j}| \tag{3.9}$$

The total loss is the sum of both losses, with a weight λ representing the relative importance of regularization:

$$L = L_{good} + \lambda L_{regu} \tag{3.10}$$

3.2.2 Optimizers

The next component of backpropagation is the optimizer. The optimizer is used to iteratively update the weights and biases until the loss function is, ideally, at its global minimum. Backpropagation got its name from the way it calculates the gradients used to update the weight and biases in the next iteration. Because an FFNN is layered, the gradient of the loss function is calculated by determining the derivative of the n-th layer with respect to the $n-1$-th layer. In other words, the error is propagated back, from the output layer to the input layer. More mathematically [59], each layer of the NNFF is a vector function depending on the output of the previous layer $\mathbf{X}_{n-1}$ and its trainable variables, the weights and biases $\mathbf{W}_n \supset \{w_n^{i,j}, b_n^j\}$. Together with the matrix activation function F, that applies the previously defined activation function to each element, the input of layer n is:

$$\mathbf{y}_n = \mathbf{W}_n \mathbf{x}_{n-1} \tag{3.11}$$

$$\mathbf{x}_n = F(\mathbf{y}_n) \tag{3.12}$$

Here the bias $\mathbf{b}_n$ can be taken into account by extending the input vector $\mathbf{x}_n$ with a one, such that:

$$\mathbf{y}_n = \begin{bmatrix} \mathbf{W}_n & \mathbf{b}_n \end{bmatrix} \begin{bmatrix} \mathbf{x}_n \\ 1 \end{bmatrix} \tag{3.13}$$

Then the derivative of each layer, and thus the total derivative, can be determined as:

$$\frac{\partial L}{\partial \mathbf{y}_n} = F'(\mathbf{y}_n) \frac{\partial L}{\partial \mathbf{x}_n} \tag{3.14}$$

$$\frac{\partial L}{\partial \mathbf{W}_n} = \mathbf{x}_{n-1} \frac{\partial L}{\partial \mathbf{y}_n} \tag{3.15}$$

$$\frac{\partial L}{\partial \mathbf{x}_{n-1}} = \mathbf{W}_n^\top \frac{\partial L}{\partial \mathbf{y}_n} \tag{3.16}$$

Using the gradient, the trainable variables can now be updated using a learning rule. In training step k the learning rule has the form[2]:

$$\mathbf{W}_{k+1} \equiv \mathbf{W}_k - \Delta \mathbf{W}_k = \mathbf{W}_k - \gamma \left. \frac{\partial L}{\partial \mathbf{W}} \right|_k \equiv \mathbf{W}_k - \gamma \nabla L_k \tag{3.17}$$

Where γ is the step size, which is constant in the simplest case. The step size determines how quickly the weights are being changed, so it is usually tweaked with a factor commonly called the learning rate η. In more advanced algorithms, the step size can be adjusted per step, so $\gamma = \eta(k)$. In the class of Newton-based optimizers, the step size is calculated based on (an estimation of) the inverse Hessian of the cost function, $\gamma = \eta \big[\mathbf{H}(\mathbf{W})\big]^{-1}$ Newton-based

[2] In some algorithms, the average gradient is used instead of the gradient

algorithms generally converge in less steps, but the steps are computationally more expensive. An overview of common choices for optimizer algorithms can be found in table 3.1.

Order	Algorithm	$\Delta \mathbf{W}_k$	Hyper-parameters	TensorFlow function	Method
1st	Gradient Descent	$\eta \nabla L_k$	η	`GradientDescentOptimizer`	Basic
	Momentum[61]	$\eta \nabla L_k - p \Delta \mathbf{W}_{k-1}$	η, p	`MomentumOptimizer`	Bigger steps as more steps are in the same direction
	Adagrad[62]	$\frac{\eta}{\sum_{i,i} (\nabla L_k)^2} \odot \nabla L_k$	η	`AdagradOptimizer`	Decay step size per parameters based on adaptive gradient.
	Adadelta[63]	$\frac{RMS[\Delta \mathbf{W}_{k-1}]}{RMS[\nabla L_k]}$	ρ, η_0	`AdadeltaOptimizer`	Decay step size using decaying average, less aggressive than Adagrad
	Adam[64]	$\eta / \frac{\nabla L_k}{1-\beta_2} \cdot \frac{(\nabla L_k)^2}{1-\beta_1}$	η, β_1, β_2	`AdamOptimizer`	Use decaying moving averages of the estimated gradient and squared gradient.
2nd	Newtons Method	$\eta \left[\mathbf{H}(\mathbf{W})\right]^{-1} \nabla L_k$	η	-	Very slow, uses unacceptable amount of memory for larger datasets.
	BFGS	$\left[\hat{\mathbf{H}}(\mathbf{W})\right]^{-1} \nabla L_k$	-	`ScipyOptimizerInterface`	Uses approximation of Hessian, low-memory version available.

TABLE 3.1: An non-exhaustive list of optimization algorithms. Loosely ordered on complexity. The displayed $\Delta \mathbf{W}_k$ and hyper-parameters only show the most important parts, for a full description look in the referenced papers. ($RMS\left[x\right]_t = \sqrt{E\left[x\right]_t} = \sqrt{\rho E\left[x\right]_{t-1} + (1-\rho)\, x_t}$)

Constructing the QuaLiKiz dataset

A dataset with QuaLiKiz in- and output has been generated to train the neural network on. First, QuaLiKiz had to be prepared to do massive parallel runs. This makes it possible to generate as many datapoints as possible within a reasonable time using the available computational budget. Intensive profiling is needed, as QuaLiKiz has not been used for this purpose before. During profiling, two main bottlenecks were identified: the calculation of elliptic integrals in for the trapped particles (section 4.1), and to IO to disk (section 4.2). Then, tools have been developed to run massively parallel QuaLiKiz jobs (section 4.3). Finally, statistics of the generated dataset are shown (section 4.4).

4.1 Elliptic integrals

The integral for trapped particles, equation (2.15), contains two elliptic integrals. It was found during profiling that the calculation of these integral using the Carlson method [65] was taking up a significant part of runtime. As can be seen in figure 4.1, solving these integrals could take up to 40% of the total walltime. For most QuaLiKiz runs, these two functions dominated the total runtime.

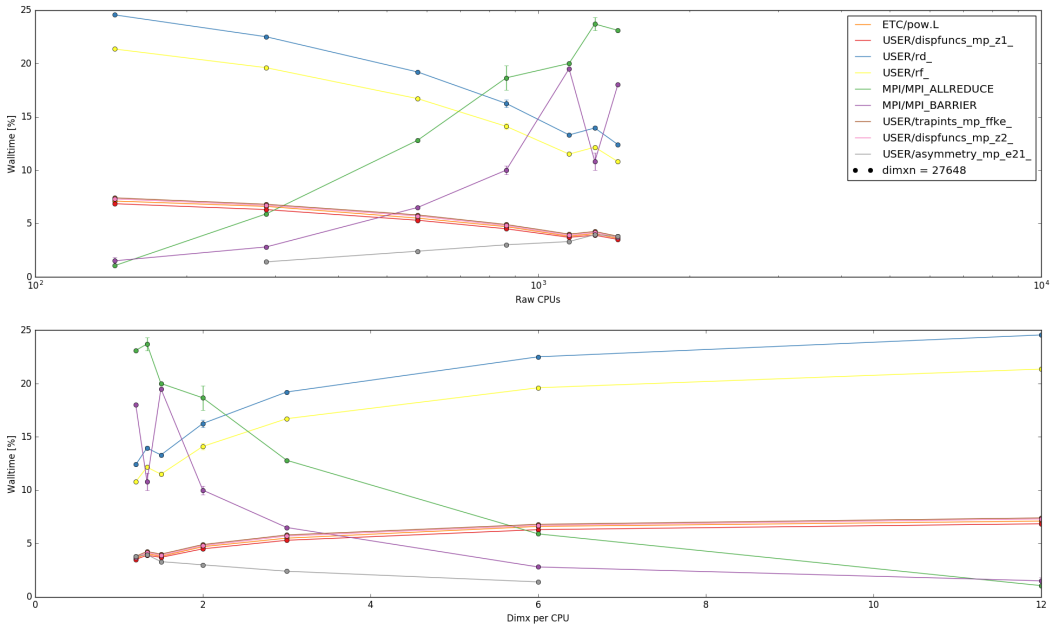


FIGURE 4.1: The relative CPU time (y-axis) used by specific QuaLiKiz functions as function of the amount of CPU cores used (top) and the amount of QuaLiKiz points per core (bottom). As can be seen in the top plot, the MPI functions (green and purple) use up to 45 % of the CPU time for a large amount of CPU cores. From the bottom plot it can be seen that the elliptic integrals rd and rf (blue and yellow) use up to 40 % of CPU time. As such, these functions are the subject of optimization in section 4.1 and section 4.2 respectively.

A suitable replacement was found to remove this bottleneck, and by doing so allowed for a much larger dataset to be generated. The used algorithm was designed by T. Fukushima [66], and uses an internal table containing pre-calculated coefficients which are normally calculated during the integration routine. In doing so, this algorithm was orders of magnitude faster, while it was able to give the same results on the integration domain within machine precision. As can be seen in figure 4.2, the elliptic functions no longer dominate the runtime. This update to QuaLiKiz has been incorporated into the latest release version [6].

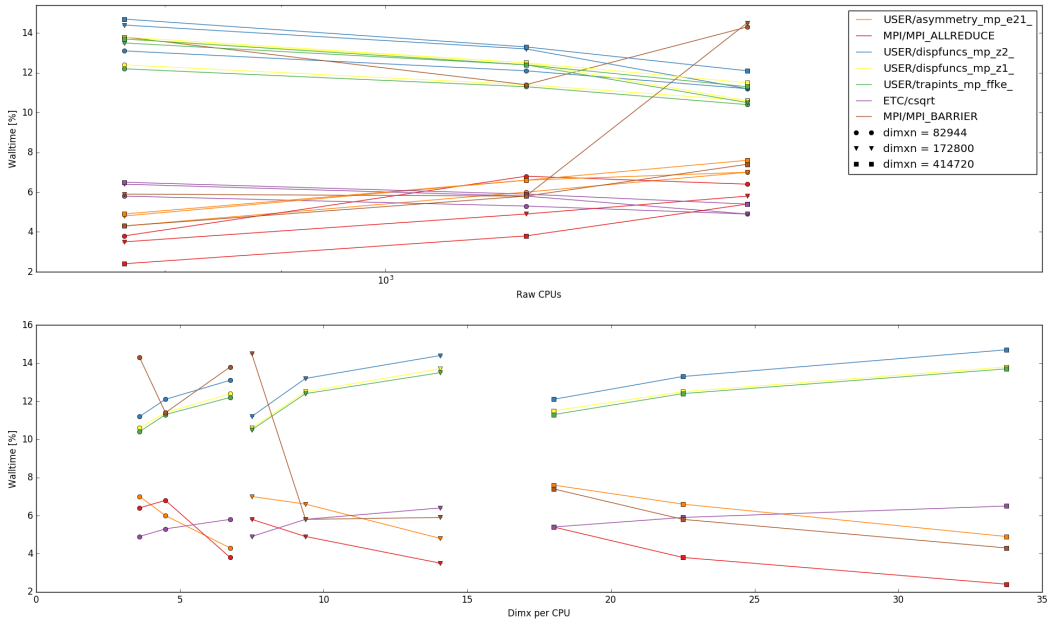


FIGURE 4.2: The relative CPU time (y-axis) used by specific QuaLiKiz functions as function of the amount of CPU cores used (top) and the amount of QuaLiKiz points per core (bottom) after massive-run optimization. Both plots do not show a clear type of function that dominates the runtime. The dataset can now be generated within the computational budget available.

4.2 Fully-parallel QuaLiKiz

Another bottleneck was found while profiling QuaLiKiz for a large amount of points. Prior to this work, QuaLiKiz was mainly used for runs containing at most a few hundred points. The bulk of the QuaLiKiz calculation is in the eigenvalue calculation. As such, only this part of QuaLiKiz was parallelized, while the rest was handled by a single core. For small runs, this is no problem. However, for a larger amount of points, the non-parallel part of QuaLiKiz started to dominate the runtime. For example, for a run of 1×10^5 points the non-parallel part used around 45% of the walltime. For runs with 1×10^4 points the walltime used by the non-parallel part was below 1%. A large speedup was gained by parallelizing the rest of QuaLiKiz. As can be seen in figure 4.2, even for 4×10^5 points the previously non-parallel part only took around 8% walltime.

4.3 Managing massive QuaLiKiz runs

The dataset generated contains 3×10^8 points and each point contains 16 growth-rate calculations. The average time to calculate a single growth-rate is about 1 s, made possible by

the optimizations to QuaLiKiz described in this chapter. Generating this dataset on a single core would take 150 years, which is clearly infeasible. As such, the dataset was generated on the Edison supercomputer at the NERSC supercomputer in U.S. Berkeley. Edison has 5586 compute nodes, each containing 24 2.4 GHz cores with hyperthreading, which means that one node can do 48 flux calculations in parallel. QuaLiKiz runs fully in memory, so the amount of points being calculated at a single node is limited by the amount of RAM available. The memory usage of QuaLiKiz itself was made as low as possible by removing any unnecessary variables. On Edison, each node has 64 GB of RAM, which allowed for a maximum of 34 560 points per job. However, to make data analysis easier it was chosen to split the full 9D hyperrectangle up in jobs each containing a 4D hyperrectangle. As such, each job contains 17 280 points and uses around 76.8 CPUh, which is more or less a single wallhour. Then, the jobs are grouped together in a batch of 5D hyper rectangles. To get even more parallelism, the batch is run on 48 nodes at the same time. Summarising, there are 1680 batches, each taking about an hour. It would take 70 days to run these batches assuming they are run sequentially without downtime in-between.

A framework to run massive QuaLiKiz jobs was designed, as micro-managing 1680 batches for several months is undesirable. The framework consists of two parts: the QuaLiKiz-pythontools to initialize and run QuaLiKiz hyperrectangle scans, and the QuaLiKiz-jobmanager to set up large scale QuaLiKiz scans on a supercomputer.

4.3.1 QuaLiKiz-pythontools

The QuaLiKiz-pythontools are a set of classes and functions implemented in Python, allowing a researcher to conveniently work with QuaLiKiz. At the heart of the tools is the hyperrectangle scan. The scan is set up with a human-readable settings file, as reproducibility is very important. One specifies the QuaLiKiz input parameters, and the parameters to be varied during the scan. The tool then calculates the specific input needed for QuaLiKiz while taking into account some physical considerations. For example, the densities of species needed to maintain quasi-neutrality, or the temperature needed for a specific collisionally. Finally, the input files are generated and QuaLiKiz is run.

Tools have been designed to do output processing as well. The sheer size of the dataset suggests that an efficient way of storing and indexing data has to be used. The QuaLiKiz output is parsed and stored in a hyperrectangle with the varied input parameters as axes. Then, all data that does not vary is removed, and saved as a single value. Finally, the set is stored in HDF5 [67] and compressed. Processing the data this way allowed the final database to be only 20 GB instead of the 9 TB size of the raw output. The folded hypercube structure

also allows for sub-ms querying of the database, as well as loading only part of the dataset in memory, allowing data analytics on normal desktop computers.

The tools are open source and available at
https://github.com/QuaLiKiz-group/QuaLiKiz-pythontools.

4.3.2 QuaLiKiz-jobmanager

The QuaLiKiz-jobmanager was developed to manage the large amount of runs. It uses an SQL database to keep track of the status of all runs. If needed, new jobs are submitted to the supercomputer using the QuaLiKiz-pythontools. The output of runs that are finished are parsed, folded into a hyperrectangle, compressed and archived automatically. This allows for quick analytics on the output of the QuaLiKiz runs while generating the dataset simultaneously. This allowed for small bugs in QuaLiKiz to be fixed real-time, barely slowing down the dataset generation. These bugs only became apparent in exotic corners of parameter space, outside the space typically explored in integrated modelling. It also makes future expansion of the dataset easy to set up and manage.

The tools are open source and available at
https://github.com/QuaLiKiz-group/QuaLiKiz-jobmanager.

4.4 Dataset validation

4.4.1 Statistics

The dataset covers a 9D input dimension hyperdimensional space. As such, visualizing the dataset is complicated and outside the scope of this work. Instead, some relevant statistics are shown in this section. For example, the various scalings in table 2.2 should be reproduced by QuaLiKiz. The scaling should become apparent by looking at the population of stable points for a specific input parameter, although cross-parameter effects are not visible in this way. Here stable is defined as a point where QuaLiKiz found a growth rate of 0. The result of this analysis can be found in figure 4.3.

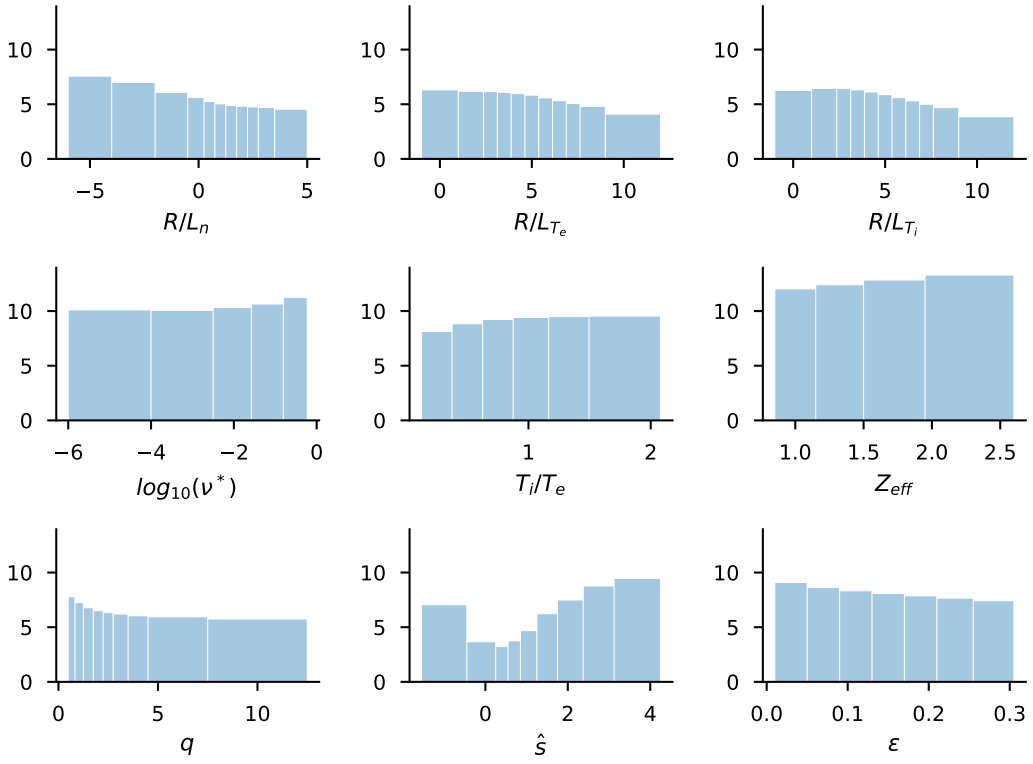


FIGURE 4.3: Stable point fraction for the full flux in the dataset dependent on input parameter. Yaxes show population in bin over the total amount of points in percent. As the input does not form an orthogonal basis for the output, direct conclusions based on this plot are not possible. However, general trends can be observed. For example, $^{R}/_{L_n}$, $^{R}/_{L_{T_e}}$, $^{R}/_{L_{T_i}}$ are destabilizing, as an increase of these parameters reduces the stable population. The other parameters, ν^*, T_i/T_e, Z_{eff}, q, ε, and a very high or very low magnetic shear $\hat{s}$ have a stabilizing effect.

Transport coefficients in experimental plasmas are usually limited to a range between 0 and 10 in GyroBohm units. The population of the dataset within certain ranges of q_s are shown in figure 4.4. As can be seen, most of the generated datapoint are within experimentally relevant ranges.

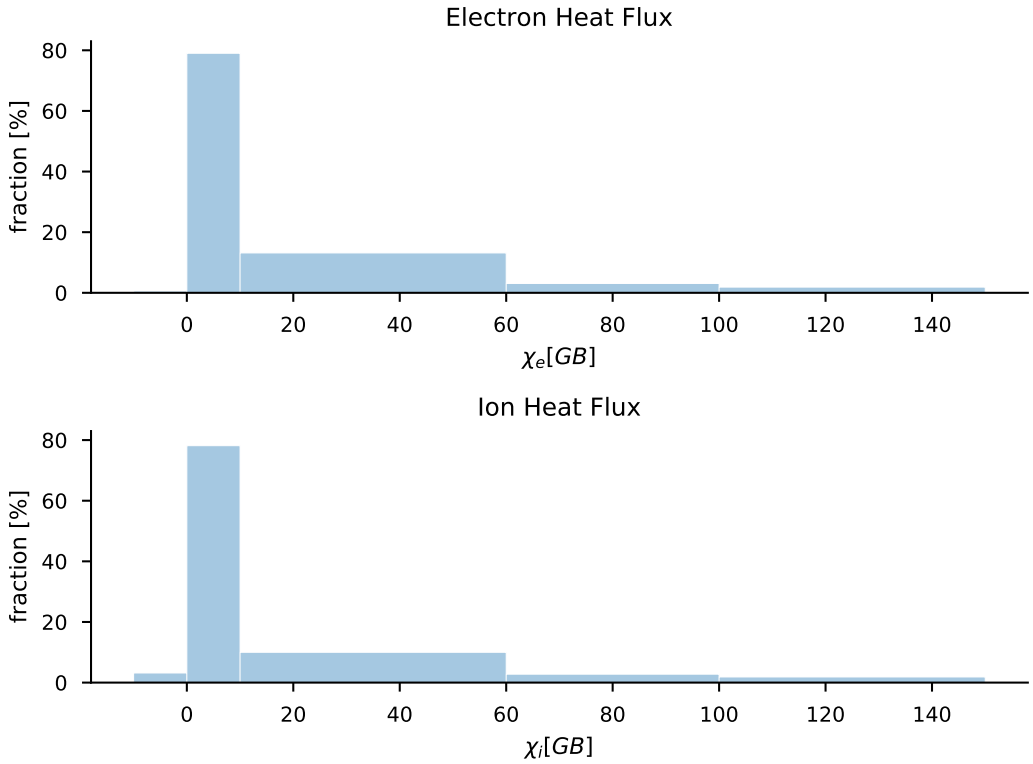


FIGURE 4.4: Fraction of dataset within a certain range of heat flux. Most of the dataset is within experimentally relevant range, between 0 and 10 in GyroBohm units. A very small fraction has a negative heatflux, see section 4.4.3. The rest of the fluxes above expected fluxes for experimental fusion plasma's, but can still be used in the Neural Network training process.

4.4.2 QuaLiKiz-dataslicer

It is crucial to visually check the dataset, as global statistics hide local details that might be important for transport modelling. A tool has been developed to look at 1D slices through the dataset. This way, one can easily identify QuaLiKiz points that might be in the wrong place. By looking at the $k_\theta \rho_s$ spectra, one can identify possible reasons why this particular point is wrong, for example, an eigenvalue not found by the eigenvalue solver. Also, one can plot various trained neural networks in the same plot, allowing networks to be rapidly compared qualitatively, and increase our intuition throughout the multi-dimensional space.

4.4.3 Validation of QuaLiKiz

The large dataset of QuaLiKiz runs can be used for the validation of QuaLiKiz against more complete codes. However, to do a direct comparison, one would need a database with different gyrokinetic codes using the same normalizations. The GyroKineticDataBase (GKDB) project

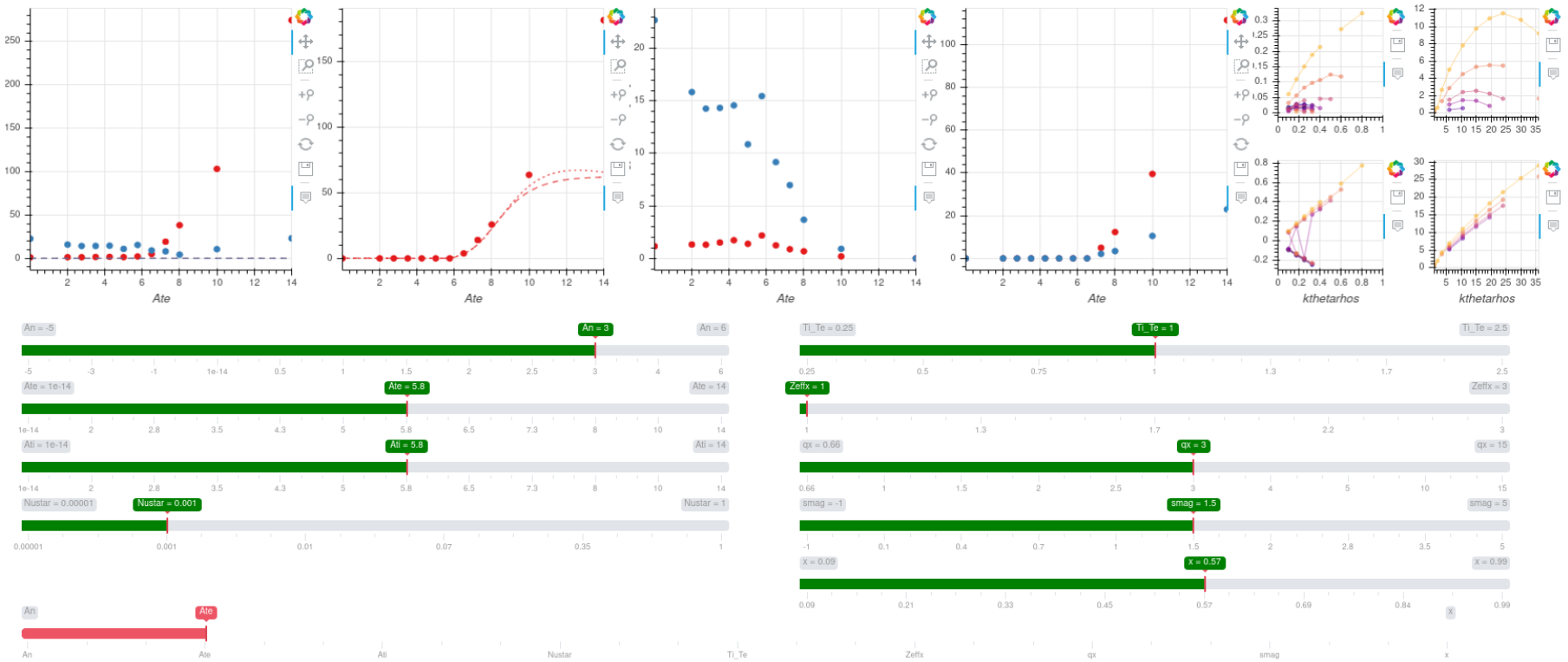


FIGURE 4.5: Screenshot of the QuaLiKiz-dataslicer. One can adjust the value plotted on the x-axis of the plots, currently showing L_{T_e}. Then, the sliders can be adjusted to choose a specific slice. The dataslicer can be found online at qualikiz.dataslicer.com

was set up to allow for such an cross-code validation. It is envisioned that part of the generated database will be included in the GKDB. Although the project is at a very early stage at time of writing, it can be found at gkdb.org.

One example of a subspace that would need to be validated with more complete codes is a region of negative fluxes. While only occuring for a small minority of cases, a negative heat flux may be indicative of a strong heat pinch in certain corners of parameter space. According to figure 4.6, this tends to occur for hollow density profiles (negative $^{R}/_{L_n}$) and high electron temperature gradients. A more complete analysis is necessary to judge physical correctness and experimental relevance of these observations.

A figure with the population fraction that result in negative fluxes can be found in figure 4.6.

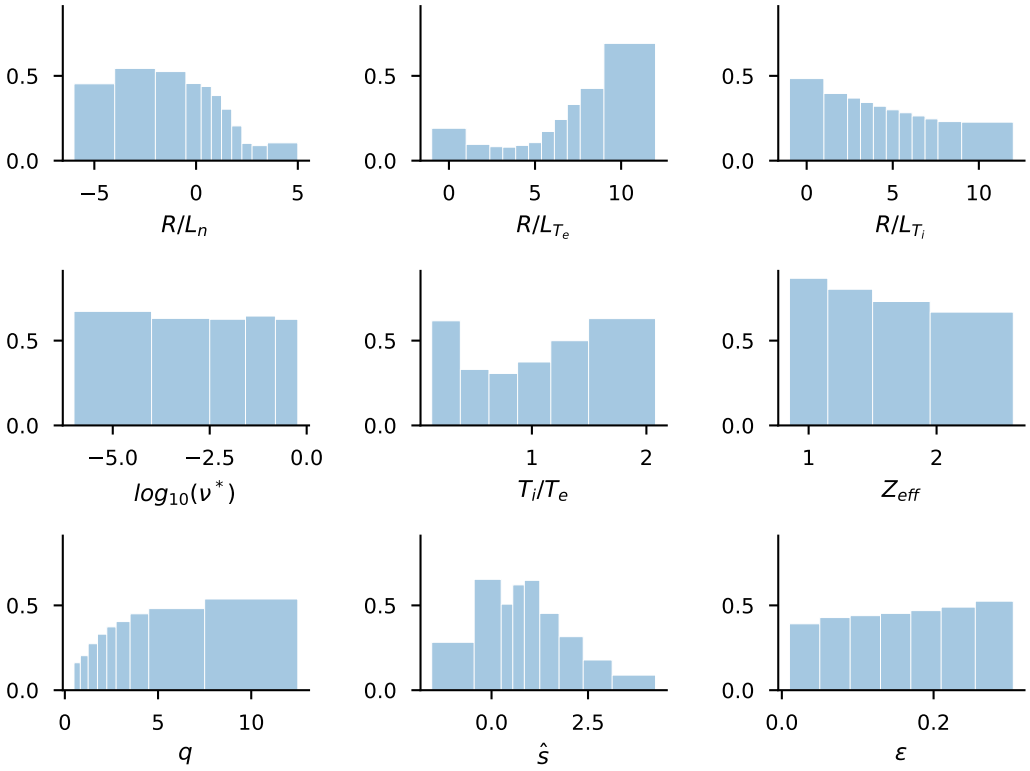


FIGURE 4.6: Fraction of dataset with negative energy fluxes dependent on input parameter. This figure shows that the negative fluxes mostly occur for hollow density profiles (negative $^{R}/_{L_n}$) and high electron temperature gradients.

Training and validation of Neural Networks

The generated dataset of QuaLiKiz in- output described in chapter 4 can now used to train neural networks. Because of the huge amount of datapoints, training these networks can take a long time. Since the training time is approximately linearly related to training set size, we focus instead on a 7D subset of the dataset. For this dataset, $Z_{eff} = 1$ and $\nu^* = 10^{-3}$. Even for this 30 times smaller dataset, training still takes around 2 hours per network. To allow for a more extensive optimization of network models and hyperparameters, priority is given to optimizing for electron ETG flux $q_{ETG,e}$. It is expected, confirmed by very preliminary tests, that the 9D networks and networks of TEM and ITG fluxes are optimal at similar hyper parameters as the ETG heat fluxes. This can be explained because of the similar structure of the threshold behaviour in R/L_{T_e} for TEM and ETG, and R/L_{T_i} in ITG.

The final goal, beyond the scope of this work, is the inclusion of these networks in transport modelling. As such, the main focus of this chapter is on a feature very important for transport modelling: the critical gradient. In this chapter the behaviour of this feature is shown with respect to different hyperparameters. First, the hyperparameters subject to tuning are shown (section 5.1). Next, RMS error, a common measure of goodness is discussed (section 5.2). Then, a qualitative analysis of trained $q_{ETG,e}$ networks is shown. Finally, a short preliminary note on execution speed and extension to other output variables is given in sections 5.4 and 5.5.

5.1 Hyperparameters

A pipeline has been set up to create and compare networks with different hyperparameters. The experimenter can choose hyperparameters using a settings file. After training, this file and the resulting network are stored in a SQL database. Afterwards, one can easily query the database and compare different networks. Additionally, some post-processing is set up to determine different measures of goodness against different subsets of the dataset. For example, the RMS error against the stable population of QuaLiKiz points, as in section 5.2.

At time of writing, these hyperparameters are chosen manually. However, an automatic optimum search routine is planned for future work. The main hyperparameters available can be found in table 5.1

TABLE 5.1: The main hyperparameters available in the neural network training pipeline. The parameters not contained in this report are greyed out.

Class	Description	NNDB name
Topology	Hidden layer structure	`hidden_neurons`
	Hidden layer activation function	`hidden_activation`
	Output layer activation function	`output_activation`
Network parameters	Measure of goodness	`goodness`
	Cost function L2 weight	`cost_l2_scale`
	Cost function L1 weight	`cost_l1_scale`
	Measure deciding when to stop early	`early_stop_measure`
	Filter used for training set	`filter_id`
Training parameters	Amount of batches per epoch	`minibatches`
	Optimizer	`optimizer`
	Standardization	`standardization`
	Non-improving epochs before stopping	`early_stop_after`

5.2 Measure of goodness

A common way to judge the performance of regression neural network is by using the root mean square (RMS) error. It is the root of equation (3.6), so:

$$RMS = \sqrt{\left(\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\right)} \tag{5.1}$$

Or in its relative form:

$$RMS_{rel} = \frac{RMS}{y_{\max} - y_{\min}} \tag{5.2}$$

Commonly it is given as a single value. Unfortunately, this way information about the error distribution in hyperspace is lost.

To prevent losing this information, one can use a more visual way of evaluating performance. One can look at a plot of the predicted value by the neural network versus the points in the dataset. For a perfect neural network and a perfect target model, these points should all lie on a straight line with a slope of 1. However, in the case of regression one preferably smooths over noisy or incorrect data. In this case a distribution around this straight line is obtained.

By plotting prediction versus target, more details about the distribution of the error can be seen. For the 7D dataset, this is displayed in figure 5.1 for a typical $q_{e,ETG}$ network. Most of the population of fluxes lies in the lower flux region. A large part of the predicted fluxes lie close to the straight line. However, there are two populations of points that need to be further investigated, both near the origin of the figure. These are located where QuaLiKiz predicts a value near zero, but the network predicts a value between zero and ten, and the region where the network predicts a low value, even for points where QuaLiKiz predicts a value between zero and five.

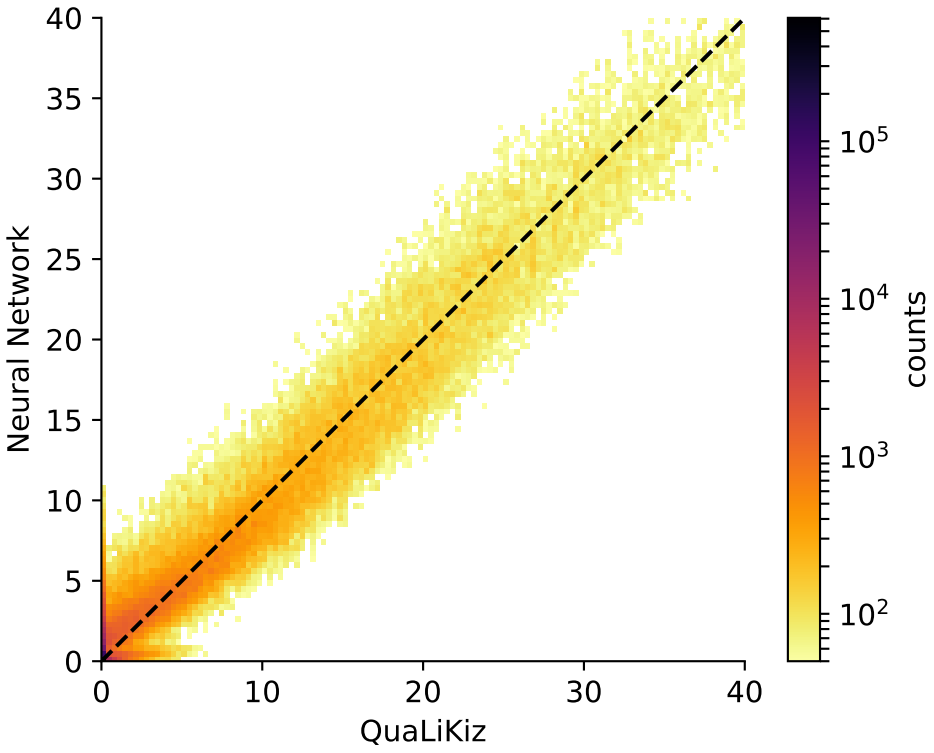


FIGURE 5.1: A plot of the QuaLiKiz prediction versus the neural network prediction. Only the unstable points are shown. Most of the population lies close to the prediction equals model line, e.g. the line with slope one (dashed). However there are two populations of points that need to be further investigated, both near the origin of the figure. The points where QuaLiKiz predicts a value near zero, but the network predicts a value between zero and ten, and the region where the network predicts a low value, even for points where QuaLiKiz predicts a value between zero and five.

5.3 Comparison of hyperparameters

5.3.1 Optimizer algorithms

Two of the algorithms described in section 3.2.2 are further explored: Adam and L-BFGS. During initial testing, it was found that the other algorithms were not converging without hyperparameter tuning. As this leaves one algorithm of each order, it was deemed to be sufficient for this work. A typical convergence curve can be found in figure 5.2. In this figure, the loss, representing how well the network performs as defined in section 3.2.1, is shown for the training set and the validation set. These two sets are randomly determined subsection of the full dataset. The validation set is not used in the optimization of the weights and biases, and is as such a measure for how well the network generalizes.

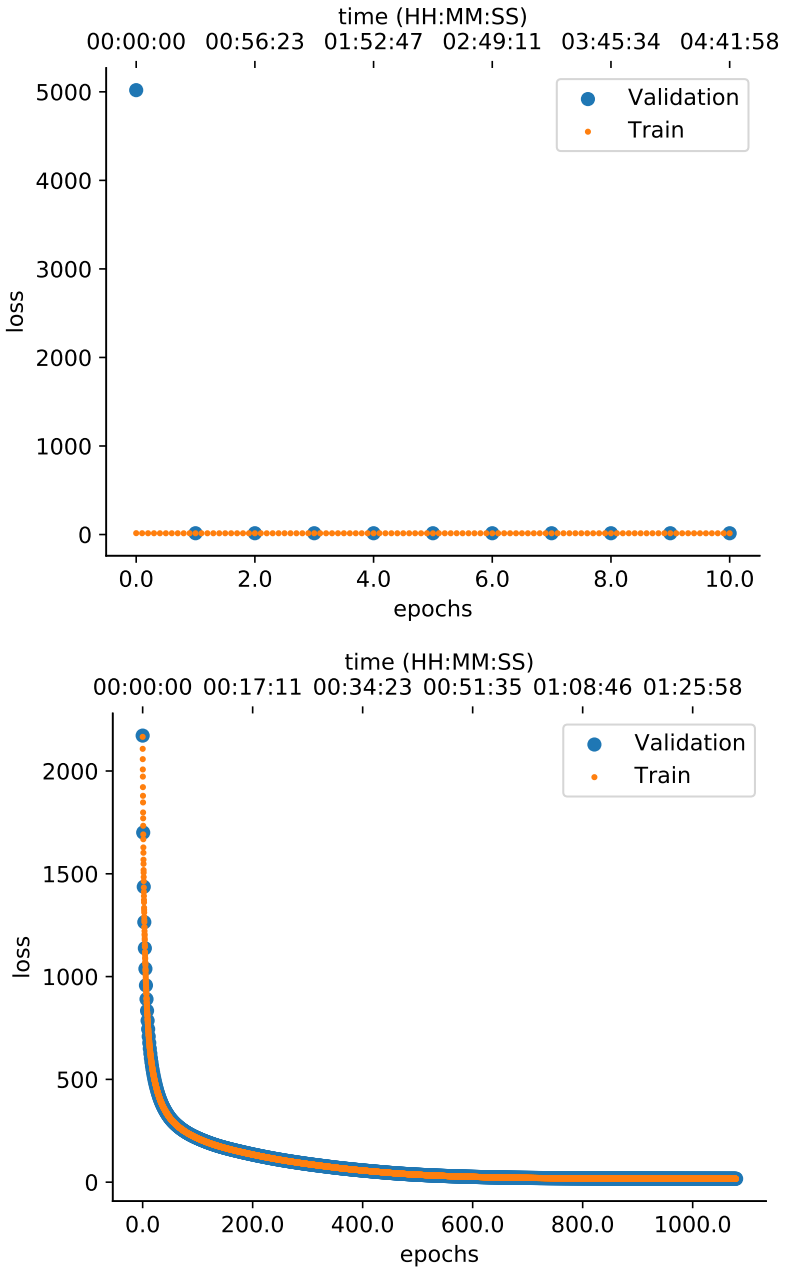


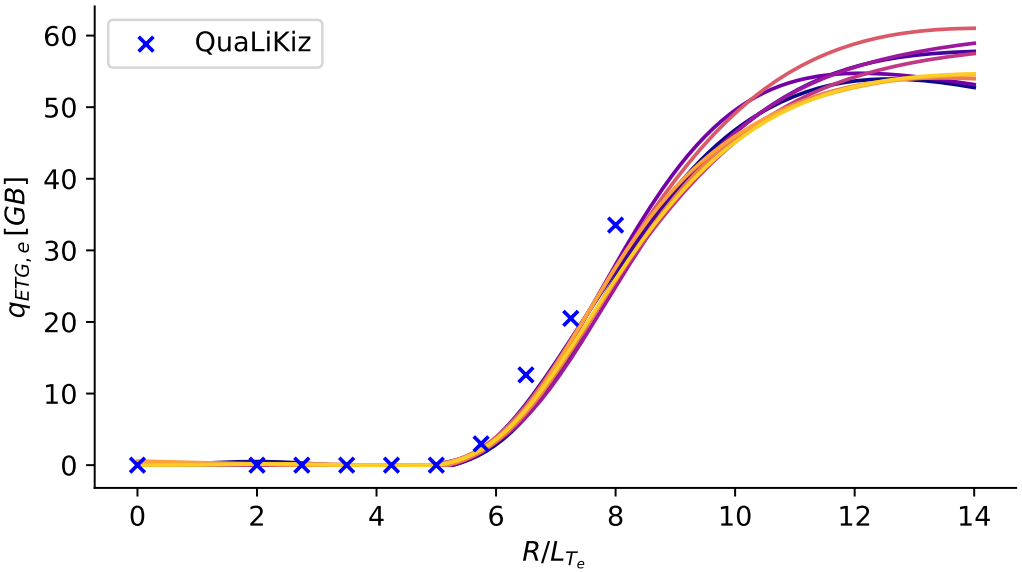
FIGURE 5.2: A typical convergence curve for the L-BFGS (left) and Adam (right) algorithm. The hyperparameters used for Adam were $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\lambda = 0.001$. L-BFGS has no hyperparameters. The loss for for the validation set is given in blue, and the loss for the training set in orange. Training was considered done if the loss did not improve for 5 epochs. Note the difference between a first and second order algorithm. The former converges in many quick epochs, while the latter converges in less epochs using considerably more time. An epoch is defined as training on every sample exactly once. Both training optimizers show that the validation and train losses are similar, which shows that the network is not overfitting.

While both algorithms show completely different convergence behaviour, they both converge to about the same final loss. As expected, this results in similar neural networks, confirmed

by visual inspection. Unless explicitly stated, the rest of the networks in this work are trained with the Adam algorithm, as this results in less walltime used for training.

5.3.2 Reproducibility

It is useful to have a baseline of comparison before networks with different hyperparameters can be compared. Networks trained with exactly the same hyperparameters have been trained for this purpose. The only differences are different initial values of biases and weights and a different split of the train, test, and validation set. This is due to the random normal initialization and the shuffling of samples respectively. One would expect to see slightly different networks behaving largely the same, due to each network being a different local optimum. A typical slice can be found in figure 5.3. As can be seen, the networks produce similar values for the region within the training set.

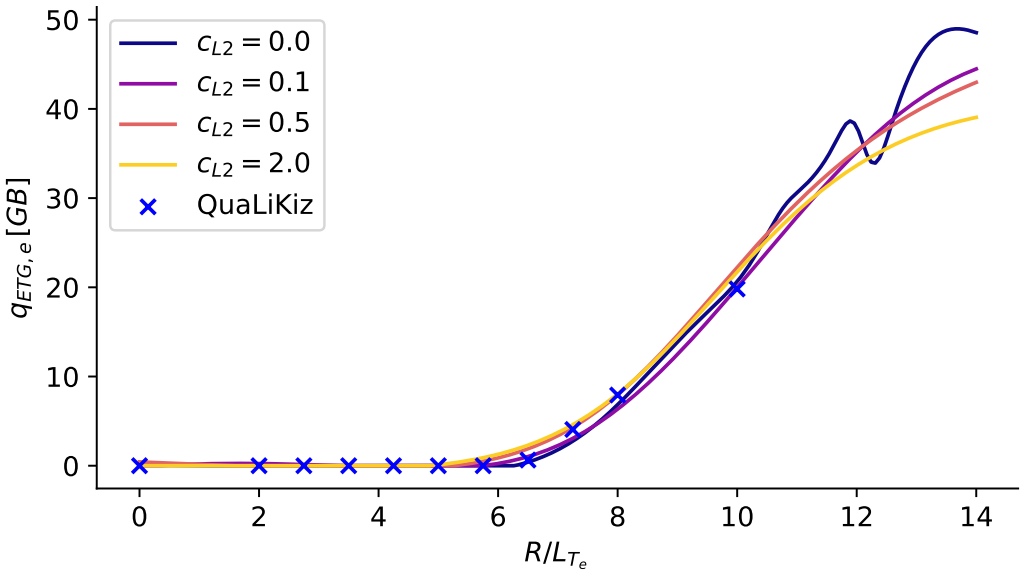


R/L_n	R/L_{T_i}	T_i/T_e	q	$\hat{s}$	$\varepsilon\,(r/R)$
2.50	14.00	1.33	1.00	0.70	0.03

FIGURE 5.3: Different networks with the same hyperparameters. The table shows the values of the other input parameters. The lines are different trained neural networks, all using the same hyperparameters. Notice the similarity of the networks within the bounds of the training set. However, as expected, the networks are diverging when extrapolating. The networks have an average RMS error of 3.35 with a standard deviation of 0.05 on the test set.

5.3.3 Regularization

Next, the regularization parameter is varied. The parameter is equivalent to λ in equation (3.10) using the L_2 loss defined in equation (3.8).



R/L_n	R/L_{T_i}	T_i/T_e	q	$\hat{s}$	$\varepsilon\,(r/R)$
2.00	2.00	0.50	3.00	0.10	0.11

FIGURE 5.4: A slice over R/L_{T_e} comparing different regularization parameters. The table shows the values of the other input parameters. The blue line shows very non-monotonous behaviour, a sign of under-regularization. The yellow line has trouble capturing both the points of high and low flux, a sign of over-regularization.

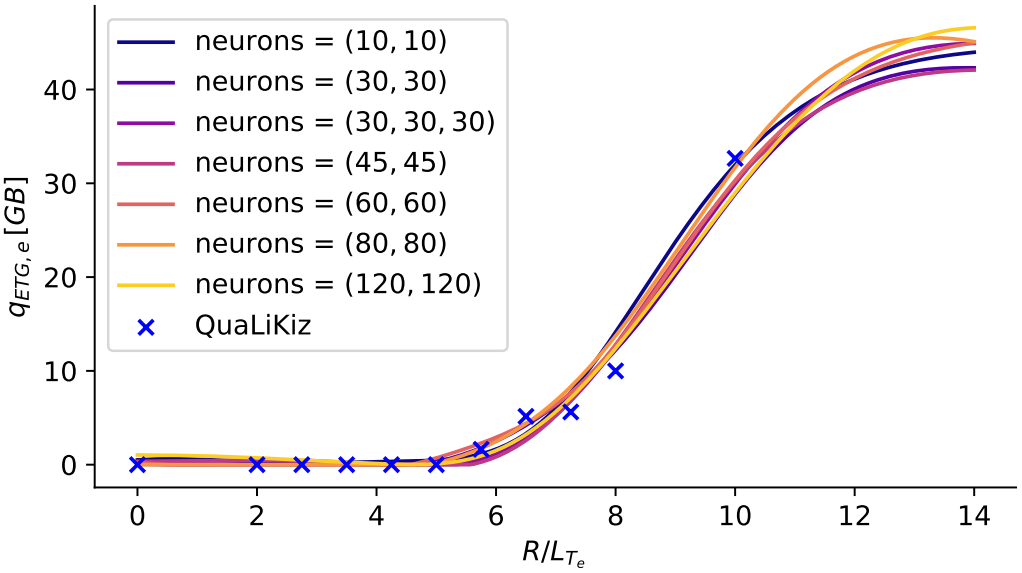
In the figure, the visual change between different networks is large, but the influence on the global RMS error is relatively minor, as shown in table 5.2.

TABLE 5.2: RMS error of networks trained with a different regularization parameter c_{L2}.

c_{L2}	RMS error
0.00	3.1605
0.05	3.4149
0.10	3.2631
0.20	3.5454
0.35	3.7103
0.50	3.8537
1.00	4.3036
2.00	5.0452

5.3.4 Topology

An important parameter for the evaluation speed is the amount of layers and neurons per layer in the network. To investigate the effect on network performance, this network topology is varied. A typical slice can be found in figure 5.4.



R/L_n	R/L_{T_i}	T_i/T_e	q	$\hat{s}$	$\varepsilon\,(r/R)$
2.00	3.50	0.50	10.00	0.10	0.15

FIGURE 5.5: A slice over R/L_{T_e} comparing different topologies of hidden neurons. The table shows the values of the other input parameters. The networks look very similar, but seem to extrapolate slightly differently. As the topology influences the loss function, a numerical comparison is not possible in this plot.

Note that the measure of regularization defined in equation (3.8) does not scale with amount of neurons. As such, the effects in this comparison might be to regularization effects only, and not by the topology. A separation of these effects is left as future work.

5.3.5 Filtering

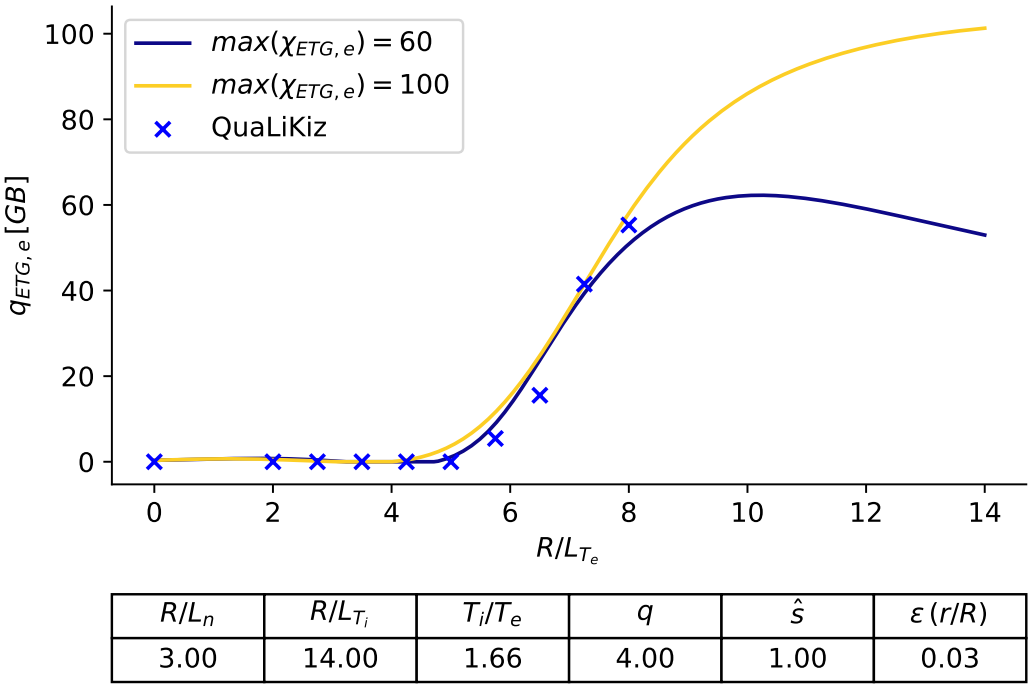
Data filtering and validation is an important step of neural network training. Only the two most analysed styles of filtering data are presented in this report. First, any point that has a negative ETG, ITG, TEM or total flux is filtered out. It is generally accepted that these fluxes should be positive, and as such they are filtered out until checked by more complete gyrokinetic codes.

Only the unstable points are included in the training set to cleanly capture the instability thresholds. As the networks are regularized, including the stable point, which have zero flux, in the training set would create a smooth bend near threshold. By filtering out the zeros, the networks extrapolate into the stable region with negative fluxes. Then, these predictions of negative fluxes are clipped to zero in postprocessing. This should result in a sharper threshold. However, the robustness of this process must be checked and is part of the sanity checks of the final networks

Finally, all point below a filter-dependent maximum are filtered out. The filter differences can be found in table 5.3 and a typical slice can be found in figure 5.6.

TABLE 5.3: 9676800 QuaLiKiz points were included in the total 7D dataset. This table shows the fraction of points left after the filter is applied.

filter name	filter stable	filter maximum [GB]	Electron ETG points left [%]
max = 60	yes	60	20
max = 100	yes	100	26



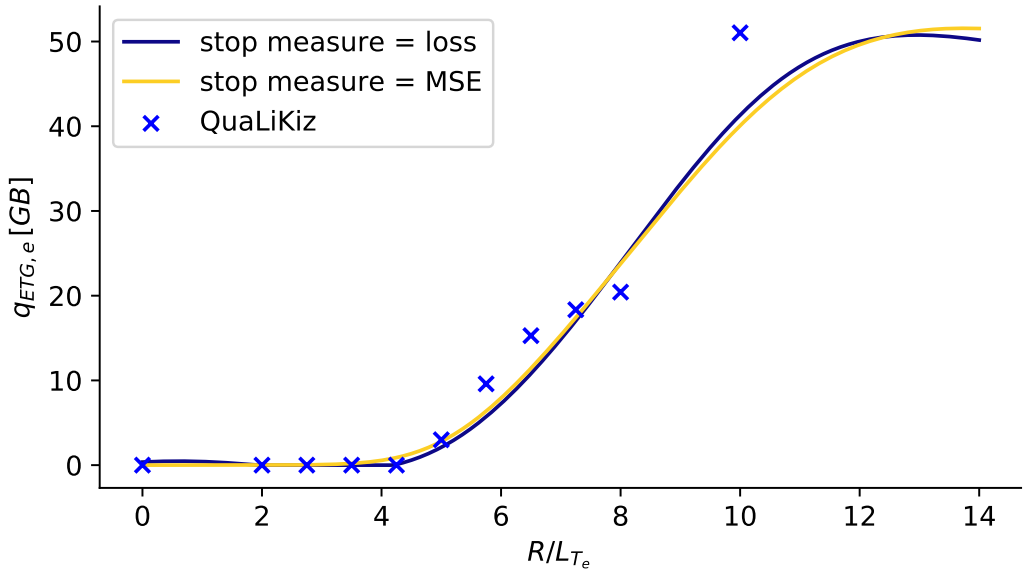
R/L_n	R/L_{T_i}	T_i/T_e	q	$\hat{s}$	$\varepsilon\,(r/R)$
3.00	14.00	1.66	4.00	1.00	0.03

FIGURE 5.6: A slice over R/L_{T_e} comparing different filters. The table shows the values of the other input parameters. The networks that includes QuaLiKiz points with higher flux naturally approximates the data better for higher flux. However, there seems to be a qualitative effect on the steepness and location of the threshold.

Naturally the networks trained with the dataset including the higher fluxes fits the QuaLiKiz points with high flux better. However, this seems to have effect on the steepness of the fit after threshold, as well as the prediction of where the threshold is in $^{R}/_{L_{T_e}}$ space. This has only been confirmed in a qualitative analysis, a more complete quantitative analysis is needed to confirm this difference.

5.3.6 Early stop measure

The measure by which to decide to stop training is generally the loss function, as this is the object of optimization. However, in principle any measure can be used to decide to stop. Most of the early trained networks stopped based on the RMS error, as initially this was thought to be the main measure of goodness of interest. In the final stages focus moved away from RMS error, and as such a comparison with the more general stopping on loss was made. A typical slice of this comparison can be found in figure 5.7.



R/L_n	R/L_{T_i}	T_i/T_e	q	$\hat{s}$	$\varepsilon\,(r/R)$
2.00	3.50	1.33	1.50	0.10	0.07

FIGURE 5.7: A slice over $^{R}/_{L_{T_e}}$ comparing different measures of early stopping. The table shows the values of the other input parameters. The training was stopped after 5 successive steps that did not improve either RMS error, or the loss function. In a qualitative comparison, the networks behave mostly similar, and only differ in subtle ways. More quantitative analyses is needed to quantify this difference.

In both cases the training was stopped after either the RMS error or loss did not improve after 5 successive steps of the optimizer algorithm. The two measures look similar if one

looks at many R/L_{T_e} slices, but there seem to be subtle differences which currently cannot be quantified. However, the effect seems to be insignificant and thus will not be the subject of future optimization.

5.4 Extension to TEM, ITG and 9D heat fluxes

Some preliminary tests have been done to check the extension of output space from ETG fluxes to TEM and ITG fluxes, as well as an extension to 9D ETG heat fluxes. A summary of these best hyperparameters found for 7D ETG heat flux can be found in table 5.4.

TABLE 5.4: Summary of the hyperparameters found for training ETG heat flux.

Hidden layers	3
Nodes per layer	30
Activation function	tanh
Measure of goodness	Mean square error
Regularization	L2 loss and early stopping
Early stop criteria	Loss
Filtering	Max = 60

Then, networks are trained using the same pipeline and hyperparameters. This results in the following networks, summarized in table 5.5.

TABLE 5.5: RMS errors and L_2 norms of neural networks for different training targets. All networks use the same hyperparameters, as specified in table 5.4

Training Target	L_2 norm	RMS error [GB]	RMS error [%]
Ion TEM Heat Flux	9.05491	2.26533	4.554682
Electron TEM Heat Flux	13.91730	2.97436	4.210860
Electron ETG Heat Flux	14.41390	3.18009	4.903538
Electron ETG Heat Flux (9D)	49.8078	3.8967	2.6508
Ion ITG Heat Flux	39.87620	5.41400	5.351478
Electron ITG Heat Flux	8.75148	2.31380	5.654686

As this small test shows, the RMS errors are close to those of the 7D ETG heat flux ones, and better or close to the requirements introduced in section 1.1. This validates the approach taken in this work: First, the optimization in 7D and then a simple extension to 9D and other-flux networks.

5.5 Note on evaluation speed

As in the end the evaluation of the neural network is just a simple matrix equation, it can usually be evaluated within a few microseconds. This is confirmed by the proof-of-concept[17] and early tests within the compiled version of RAPTOR.

Discussion

In this work, we have generated a large database containing 3×10^8 QuaLiKiz runs. The QuaLiKiz inputs cover a wide range of parameter space relevant for the core region of tokamak experiments. The output most notably contain particle and heat fluxes in the ETG, ITG, and TEM regime. This dataset was then used to train Feed-Forward Neural Networks, which provide a regression function able to emulate QuaLiKiz. Extensive work has been done to optimize the hyperparameters that govern these networks for the 7D ETG heat flux. These optimal network has a reasonable RMS error and L2-norm, and is in qualitative agreement with the QuaLiKiz points and their predicted threshold behaviour. Initial test show that these hyperparameters also give good results for the other heat fluxes, as well as for the 9D networks. The set up pipeline makes it easy to extend this work to 9D heat and particle fluxes, as well as carry out extra quantitative analysis. Finally, these networks can be coupled to a transport model in an integrated modelling suite for validation against experiments, optimization of plasma shots and control applications.

6.1 QuaLiKiz dataset

The dataset generated for this work is the largest collection of linear gyrokinetic results ever. Additionally, the extra parallelization makes the code massively parallel, being able to be run on thousands of cores simultaneously. Together with the pipeline that has been set it, this makes it easy to expand the database as needed.

The database can also be used to explore the parameter space of tokamak core plasma's. In this way, interesting new regions can be found for additional (virtual) experiments. For example, turbulence simulations in a stable region are unlikely to give additional information. As such, one could find the stable regions using the database to avoid running expensive simulations in an uninteresting region of parameter space. Another region of interest is the space near the turbulent threshold, as that is the region most tokamak experiments operate in. This exploration is further simplified by the design of the QuaLiKiz-dataslicer.

Finally, as the GyroKineticDataBase project is extended to include more gyrokinetic codes, the dataset allows for cross-code validation of QuaLiKiz.

6.2 QuaLiKiz Neural Networks

The neural networks trained in this work are a significant improvement of the proof of concept [17]. The initial input space has been increased from 4D to 9D. The outputs investigated are electron and ion heat fluxes in the ITG, ETG, and TEM regime, more general than the proof of concept which only included ITG heat fluxes. Regression of growth rates has not been done, but the set up pipeline makes it easy to extend this work to include them, as well as networks for the particle fluxes. However, multiple challenges have been identified during validation of these neural networks.

6.2.1 RMS error as measure of goodness

Often times, the RMS error of the network predictions is used as the end-all measure of goodness of the neural networks. However, for networks that are meant to be used for transport modelling, other, more local effects are far more important than the RMS error. In this work, three such effects have been identified.

The first is threshold miss-prediction, when the networks predicts the threshold to be located different to the original QuaLiKiz threshold. It is related to the relative sparseness of the dataset, so the threshold itself is almost never contained in the training data. Combined with the smoothness of neural networks, it can mean that the threshold is found on a significantly different $^{R}/_{L_{T_e}}$ or $^{R}/_{L_{T_i}}$, resulting in turbulent fluxes appearing in stable regimes.

A related effect threshold mismatch. This happens when the network for, for example, electron heat flux finds a threshold at a different location than the ion heat flux. This could mean that the ion temperature remains fixed at a certain gradient (if the mode is ITG), while no electron heat flux from ITG is predicted in the q_e model. This would result in a runaway of the T_e profile, until other instabilities are triggered. This is physically inconsistent. It has been presented at EPS 2017 [68, 69].

The last effect is residual flux pop-back. As the network is extrapolating in stable regions, the flux may pop back up slightly. In this case, the network predicts a very small, but non-zero flux in regions where there should not be any turbulent fluxes. This can happen even when including points of zero flux, as the RMS error is biased for larger errors, it allows for small errors to be made around 0. Although the influence might be small, it is hard to verify without doing actual transport simulations, so this effect should be kept in mind during validation.

Although all these effects have been found qualitatively, a quantitative mathematically rigorous measure has to be determined to fully compare different networks to determine which one is best. Fortunately, using the developed tools extending the pipeline with these extra analytics should be straightforward, and no problems are foreseen in extending this work.

6.2.2 Comparison with state-of-the-art

The neural network regression of a turbulent transport model carried out in this work is the largest one in the world to date. The only similar work was done by O. Meneghini *et al.*[70], in which a neural network regression was done on the TGLF turbulent transport model. However, the database used for this regression was based on only 24 DIII-D shots. There is no hyperspace analysis presented in the paper, but it is expected that the parameter space covered is relatively narrow to this work. The size of the database is not mentioned in the paper, but assuming 20 points per radial slice and the reported 4000 time slices of 24 shots, 10^6 total points are obtained, two orders of magnitude less than the database in this work. However, their database covers 24 input parameters versus 9 in this work.

Outlook

Multiple topics have been identified to improve upon this work in the future.

The dataset can be expanded with data near the thresholds, using the already generated one as guide. The networks are then automatically biased to give more importance to the threshold region, as there are more data points there. Expanding the training set also improves generalization. Related to this is implementation of instance weighting. With instance weighting, every point in the dataset is given a relative importance. This importance is then incorporated in the loss function, so that the trained network better approximates the important points. More smartly filtering out points from the training set can also lead to improvement. It is known that QuaLiKiz sometimes fails to predict a growth rate, which leads to a smaller flux than in reality. By looking through the data space and removing these points from the dataset, it can be avoided that the networks train on these incorrect points, skewing the prediction near them.

Adding extra dimensions to the dataset is also possible. However, the curse of dimensionality makes adding dimensions in the hyperrectangle way as in this work computationally too expensive. However, this cost can be circumvented by restricting oneself to the (input)parameters found in experimental data instead of a hyperrectangle. This will need a large-scale analyses of experimental data, after which gyrokinetic code can be run. Fortunately, much of the pipeline set up in this work can be re-used. Currently, this analysis of experimental data is being done for JET profiles, consisting of 2000 discharges with 7 time-slices each, and will be extended in future work [49].

The GKDB can be expanded with gyrokinetic data from other less simplified codes. One can then train neural networks on this database. However, because of the computational cost involved, it will take simulation data from multiple sources. Currently it is planned to supply an open-source interface to the GKDB, such that the community can freely contribute.

As noted in section 6.2.1, a well-defined measure of goodness has to be determined. This will make comparison of networks easier than the visual method used in this work.

The tuning of hyperparameters can be done automatically, as opposed to a manual grid-like search like in this work. Using Beyesian optimization [71], one tries to predict the best next parameters to try. This is a point were either the network is expected to perform best, or a point were the uncertainty of how well the network will perform is highest. This way, one needs to train less networks than a naive grid search.

The 9D input space can be expanded to 10D in post-processing by using a Waltz-like rule for including the impact of rotational flow shear. A toy model for this rule has already been set up [48], as well as the pipeline to train 10D networks. However, the robustness of this rule still has to be tested.

Finally, the created networks have to be validated using a integrated modelling suite. This is planned to be done within the RAPTOR framework [18]. The 4D networks of the proof of principle are currently integrated in RAPTOR, and extending this to 9D is of the highest priority.

Conclusion

The goal of this work was to accurately predict particle and heat transport coefficients as function of a 9D input space using a feed-forward neural network in realtime. The input parameters of interest were $q_{i,e}$ and $\Gamma_{i,e}$ as function of a $-\frac{R}{T_{i,e}}\frac{\mathrm{d}T_{i,e}}{\mathrm{d}r}$, $\frac{T_i}{T_e}$, $-\frac{R}{n}\frac{\mathrm{d}n}{\mathrm{d}r}$, Z_{eff}, ν^*, q_x, $\hat{s}$, r/R. A dataset containing 3×10^8 QuaLiKiz runs scanning a hyperrectangle of these input parameters has been made as training set for the neural networks. To allow such a large-scale run, QuaLiKiz has been further optimized and parallelized. Furthermore, tools have been developed in Python to run, manage, and validate large-scale QuaLiKiz runs.

Multiple neural networks with different hyperparameters have been trained using the QuaLiKiz dataset. The main focus was on the ETG electron heat flux, as it is expected that similar hyperparameters work for the other fluxes because of the similar structure of the in- and output mapping. To save training time, two of the input dimensions are taken as constant, namely $Z_{eff} = 1$ and $\nu^* = 10^{-3}$. These networks have been extensively validated using visual inspection. The best of these networks have an RMS error of only 3.18 with a standard deviation of 0.05, or a normalized RMS deviation of 4.9 percent, which is within the stated goal. Similar values of error have been found for the other heat fluxes, and for the initial full 9D training carried out so far. It is expected that similar errors will be found for the particle fluxes. However, it has also been found that RMS error is oftentimes not the right measure for the goodness of neural network. In this work we have identified three other important measures: threshold miss-prediction, threshold mismatch, and residual flux pop-back. These measures will have to be further quantified to rigorously compare neural networks, and to determine which is best. Still, the set up pipeline make it easy to extend this work to include these measures, and so expand the neural network to better represent reality.

The trained networks in this work represent the current state-of-the art. Including these networks in transport models will open new pathways to do fast optimization of scenarios and realtime control.

Bibliography

[1] A. Donné, A. Costley, and A. Morris. Diagnostics for plasma control on demo: challenges of implementation. *Nuclear Fusion*, 52(7):074015, 2012. URL http://stacks.iop.org/0029-5515/52/i=7/a=074015.

[2] W. Horton. Drift waves and transport. *Reviews of Modern Physics*, 71(3):735, 1999. URL http://hdl.handle.net/2152/61083.

[3] E. Doyle, W. Houlberg, Y. Kamada, et al. Chapter 2: Plasma confinement and transport. *Nuclear Fusion*, 47(6):S18, 2007. URL http://stacks.iop.org/0029-5515/47/i=6/a=S02.

[4] T. D. Rognlien. Understanding of edge plasmas in magnetic fusion energy devices. *Plasma Physics and Controlled Fusion*, 47(5A):A283, 2005. URL http://stacks.iop.org/0741-3335/47/i=5A/a=020.

[5] C. Bourdelle, J. Citrin, B. Baiocchi, et al. Core turbulent transport in tokamak plasmas: bridging theory and experiment with qualikiz. *Plasma Physics and Controlled Fusion*, 58 (1):014036, 2016. URL http://stacks.iop.org/0741-3335/58/i=1/a=014036.

[6] J. Citrin, C. Bourdelle, F. J. Casson, et al. Tractable flux-driven temperature, density, and rotation profile evolution with the quasilinear gyrokinetic transport model qualikiz. *Plasma Physics and Controlled Fusion (Submitted)*, 2017.

[7] B. Baiocchi, J. Garcia, M. Beurskens, et al. Turbulent transport analysis of jet h-mode and hybrid plasmas using qualikiz and trapped gyro landau fluid. *Plasma Physics and Controlled Fusion*, 57(3):035003, 2015. URL http://stacks.iop.org/0741-3335/57/i=3/a=035003.

[8] B. Baiocchi, C. Bourdelle, C. Angioni, et al. Transport analysis and modelling of the evolution of hollow density profiles plasmas in jet and implication for iter. *Nuclear Fusion*, 55(12):123001, 2015. URL http://stacks.iop.org/0029-5515/55/i=12/a=123001.

[9] O. Linder. To be determined. *Nuclear Fusion (to be submitted)*, 2017.

[10] J. Artaud, V. Basiuk, F. Imbeaux, et al. The cronos suite of codes for integrated tokamak modelling. *Nuclear Fusion*, 50(4):043001, 2010. URL http://stacks.iop.org/0029-5515/50/i=4/a=043001.

[11] G. Cenacchi and A. Taroni. Jetto a free boundary plasma transport code. *JET-IR*, page 84, 1988. ENEA-RT-TIB–88-5.

[12] M. Romanelli, G. Corrigan, V. Parail, et al. Jintrac: A system of codes for integrated simulation of tokamak scenarios. *Plasma and Fusion research*, 9:3403023–3403023, 2014.

[13] C. M. Bishop. *Neural Networks for Pattern Recognition (Advanced Texts in Econometrics)*. Clarendon Press, 1996. ISBN 026218253X.

[14] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[15] C. M. Bishop. Neural networks and their applications. *Review of Scientific Instruments*, 65(6):1803–1832, 1994. doi: http://dx.doi.org/10.1063/1.1144830. URL http://scitation.aip.org/content/aip/journal/rsi/65/6/10.1063/1.1144830;jsessionid=c5Ts1e2qaUKPf7ZywZv0Jjwv.x-aip-live-06.

[16] L. Allen and C. M. Bishop. Neural network approach to energy confinement scaling in tokamaks. *Plasma Physics and Controlled Fusion*, 34(7):1291, 1992. URL http://stacks.iop.org/0741-3335/34/i=7/a=008.

[17] J. Citrin, S. Breton, F. Felici, et al. Real-time capable first principle based modelling of tokamak turbulent transport. *Nuclear Fusion*, 55(9):092001, 2015. URL http://stacks.iop.org/0029-5515/55/i=9/a=092001.

[18] F. Felici and O. Sauter. Non-linear model-based optimization of actuator trajectories for tokamak plasma profile control. *Plasma Physics and Controlled Fusion*, 54(2):025002, 2012. URL http://stacks.iop.org/0741-3335/54/i=2/a=025002.

[19] M. Abadi, A. Agarwal, P. Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[20] W. Horton, B. Hu, J. Q. Dong, et al. Turbulent electron thermal transport in tokamaks. *New Journal of Physics*, 5(1):14, 2003.

[21] D. R. Nicholson. *Introduction to plasma theory*. Wiley New York, 1983.

[22] H. Sugama and W. Horton. Neoclassical electron and ion transport in toroidally rotating plasmas. *Physics of Plasmas*, 4(6):2215–2228, 1997. doi: http://dx.doi.org/10.1063/1.

872385. URL http://scitation.aip.org/content/aip/journal/pop/4/6/10.1063/1.872385.

[23] N. Crouseilles, M. Mehrenberger, and H. Sellama. Numerical solution of the gyroaverage operator for the finite gyroradius guiding-center model. *Communications in Computational Physics*, 8(3):484, 2010. URL http://www-irma.u-strasbg.fr/~mehrenbe/cicp484.pdf.

[24] Y. Idomura, M. Ida, T. Kano, et al. Conservative global gyrokinetic toroidal full-f five-dimensional vlasov simulation. *Computer Physics Communications*, 179(6):391 – 403, 2008. ISSN 0010-4655. doi: http://dx.doi.org/10.1016/j.cpc.2008.04.005. URL http://www.sciencedirect.com/science/article/pii/S0010465508001409.

[25] V. Grandgirard, Y. Sarazin, X. Garbet, et al. Computing {ITG} turbulence with a full-f semi-lagrangian code. *Communications in Nonlinear Science and Numerical Simulation*, 13(1):81–87, 2008. ISSN 1007-5704. doi: 10.1016/j.cnsns.2007.05.016. URL http://www.sciencedirect.com/science/article/pii/S1007570407001220. Vlasovia 2006: The Second International Workshop on the Theory and Applications of the Vlasov Equation.

[26] A. E. White, W. A. Peebles, T. L. Rhodes, et al. Measurements of the cross-phase angle between density and electron temperature fluctuations and comparison with gyrokinetic simulations. *Physics of Plasmas*, 17(5):056103, 2010. doi: http://dx.doi.org/10.1063/1.3323084. URL http://scitation.aip.org/content/aip/journal/pop/17/5/10.1063/1.3323084.

[27] F. F. Chen. *Introduction to Plasma Physics and Controlled Fusion*, volume Volume 1: Plasma Physics. Springer US, 2 edition, 1984. ISBN 978-1-4419-3201-3. URL http://www.ebook.de/de/product/13413995/francis_f_chen_introduction_to_plasma_physics_and_controlled_fusion.html.

[28] J. Candy and R. Waltz. An eulerian gyrokinetic-maxwell solver. *Journal of Computational Physics*, 186(2):545 – 581, 2003. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/S0021-9991(03)00079-2. URL http://www.sciencedirect.com/science/article/pii/S0021999103000792.

[29] F. Jenko. Massively parallel vlasov simulation of electromagnetic drift-wave turbulence. *Computer Physics Communications*, 125(1):196 – 209, 2000. ISSN 0010-4655. doi: http://dx.doi.org/10.1016/S0010-4655(99)00489-0. URL http://www.sciencedirect.com/science/article/pii/S0010465599004890.

[30] C. Bourdelle. *Turbulent Transport in Tokamak Plasmas: bridging theory and experiment.* Accreditation to supervise research, Aix Marseille Université, January 2015. URL https://tel.archives-ouvertes.fr/tel-01113299.

[31] A. Casati. *Ph.D. dissertation*. PhD thesis, Université de Provence (Aix-Marseille I), 2009.

[32] C. Bourdelle, X. Garbet, F. Imbeaux, et al. A new gyrokinetic quasilinear transport model applied to particle transport in tokamak plasmas. *Physics of Plasmas*, 14(11): 112501, 2007. doi: 10.1063/1.2800869. URL http://dx.doi.org/10.1063/1.2800869.

[33] J. Citrin, C. Bourdelle, P. Cottier, et al. Quasilinear transport modelling at low magnetic shear. *Physics of Plasmas*, 19(6):062305, 2012. doi: 10.1063/1.4719697. URL http://dx.doi.org/10.1063/1.4719697.

[34] X. Garbet, L. Laurent, F. Mourgues, et al. Variational calculation of electromagnetic instabilities in tokamaks. *Journal of Computational Physics*, 87(2):249 – 269, 1990. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/0021-9991(90)90253-W. URL http://www.sciencedirect.com/science/article/pii/002199919090253W.

[35] J. Candy, R. E. Waltz, and M. N. Rosenbluth. Smoothness of turbulent transport across a minimum-q surface. *Physics of Plasmas*, 11(5):1879–1890, 2004. doi: 10.1063/1.1689967. URL http://dx.doi.org/10.1063/1.1689967.

[36] C. Bourdelle, X. Garbet, G. Hoang, et al. Stability analysis of improved confinement discharges: internal transport barriers in tore supra and radiative improved mode in textor. *Nuclear Fusion*, 42(7):892, 2002. URL http://stacks.iop.org/0029-5515/42/i=7/a=312.

[37] A. Casati, C. Bourdelle, X. Garbet, et al. Validating a quasi-linear transport model versus nonlinear simulations. *Nuclear Fusion*, 49(8):085012, 2009. URL http://stacks.iop.org/0029-5515/49/i=8/a=085012.

[38] J. Connor and H. Wilson. *Survey of theories of anomalous transport*. AEA Technology Fusion, 1993.

[39] B. Coppi, M. N. Rosenbluth, and R. Z. Sagdeev. Instabilities due to temperature gradients in complex magnetic field configurations. *The Physics of Fluids*, 10(3):582–587, 1967. doi: 10.1063/1.1762151. URL http://aip.scitation.org/doi/abs/10.1063/1.1762151.

[40] S. C. Guo and F. Romanelli. The linear threshold of the ion-temperature-gradient-driven mode. *Physics of Fluids B: Plasma Physics*, 5(2):520–533, 1993. doi: 10.1063/1.860537. URL http://dx.doi.org/10.1063/1.860537.

[41] B. Kadomtsev and O. Pogutse. Plasma instability due to particle trapping in a toroidal geometry. *Sov. Phys. JETP*, 24:1172–1179, 1967.

[42] W. Horton, B. G. Hong, and W. M. Tang. Toroidal electron temperature gradient driven drift modes. *The Physics of Fluids*, 31(10):2971–2983, 1988. doi: 10.1063/1.866954. URL http://aip.scitation.org/doi/abs/10.1063/1.866954.

[43] F. Jenko, W. Dorland, and G. W. Hammett. Critical gradient formula for toroidal electron temperature gradient modes. *Physics of Plasmas*, 8(9):4096–4104, 2001. doi: 10.1063/1.1391261. URL http://dx.doi.org/10.1063/1.1391261.

[44] R. E. LaQuey, S. M. Mahajan, P. H. Rutherford, et al. Nonlinear saturation of the trapped-ion mode. *Phys. Rev. Lett.*, 34:391–394, Feb 1975. doi: 10.1103/PhysRevLett.34.391. URL https://link.aps.org/doi/10.1103/PhysRevLett.34.391.

[45] B. Cohen, J. Krommes, W. Tang, et al. Non-linear saturation of the dissipative trapped-ion mode by mode coupling. *Nuclear Fusion*, 16(6):971, 1976. URL http://stacks.iop.org/0029-5515/16/i=6/a=009.

[46] O. Linder. Comparison of tokamak linear microstability calculations between the gyrokinetic codes qualikiz and gene. Master's thesis, University of Technology Eindhoven, 2016.

[47] R. E. Waltz, G. M. Staebler, W. Dorland, et al. A gyro-landau-fluid transport model. *Physics of Plasmas*, 4(7):2482–2496, 1997. doi: 10.1063/1.872228. URL http://dx.doi.org/10.1063/1.872228.

[48] V. Dagnelie, J. Citrin, F. Jenko, et al. Decomposition of linear ITG modes with flow shear in ballooning space. In *ECA Vol. 41F*, 2017.

[49] A. Ho, J. Citrin, C. Bourdelle, et al. Applying neural networks for tokamak plasma turbulence predictions. In *ECA Vol. 41F*, 2017.

[50] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[51] D. Silver, A. Huang, C. J. Maddison, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[52] M. Karabatak and M. C. Ince. An expert system for detection of breast cancer based on association rules and neural network. *Expert Systems with Applications*, 36(2, Part 2):3465 – 3469, 2009. ISSN 0957-4174. doi: http://doi.org/10.1016/j.eswa.2008.02.064. URL http://www.sciencedirect.com/science/article/pii/S0957417408001103.

[53] J. Lister and H. Schnurrenberger. Fast non-linear extraction of plasma equilibrium parameters using a neural network mapping. *Nuclear Fusion*, 31(7):1291, 1991. URL http://stacks.iop.org/0029-5515/31/i=7/a=005.

[54] O. Meneghini, C. J. Luna, S. P. Smith, et al. Modeling of transport phenomena in tokamak plasmas with neural networks. *Physics of Plasmas*, 21(6):060702, 2014. doi: 10.1063/1.4885343. URL http://dx.doi.org/10.1063/1.4885343.

[55] P. B. Snyder, R. J. Groebner, A. W. Leonard, et al. Development and validation of a predictive model for the pedestal height. *Physics of Plasmas*, 16(5):056118, 2009. doi: 10.1063/1.3122146. URL http://dx.doi.org/10.1063/1.3122146.

[56] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. ISBN 0132733501.

[57] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[58] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com.

[59] Y. A. LeCun, L. Bottou, G. B. Orr, et al. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 1998.

[60] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature14539. Insight.

[61] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[62] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[63] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

[64] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[65] B. C. Carlson. Numerical computation of real or complex elliptic integrals. *Numerical Algorithms*, 10(1):13–26, Mar 1995. ISSN 1572-9265. doi: 10.1007/BF02198293. URL https://doi.org/10.1007/BF02198293.

[66] T. Fukushima. Precise and fast computation of complete elliptic integrals by piecewise minimax rational function approximation. *Journal of Computational and Applied Mathematics*, 282:71 – 76, 2015. ISSN 0377-0427. doi: http://dx.doi.org/10.1016/j.cam.2014.12.038. URL http://www.sciencedirect.com/science/article/pii/S0377042715000023.

[67] The HDF Group. Hierarchical data format version 5, 2000-2010. URL http://www.hdfgroup.org/HDF5.

[68] K. L. van de Plassche, J. Citrin, C. Bourdelle, et al. Realtime capable quasilinear gyrokinetic modelling using neural networks. In *ECA Vol. 41F*, 2017.

[69] J. Citrin, F. Felici, A. Teplukhina, et al. First multi-channel core transport simulations with raptor using a neural network transport model. In *ECA Vol. 41F*, 2017.

[70] O. Meneghini, S. Smith, P. Snyder, et al. Self-consistent core-pedestal transport simulations with neural network accelerated models. *Nuclear Fusion*, 57(8):086034, 2017. URL http://stacks.iop.org/0029-5515/57/i=8/a=086034.

[71] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *CoRR*, abs/1206.2944, 2012. URL http://arxiv.org/abs/1206.2944.