

CZECH TECHNICAL UNIVERSITY IN PRAGUE  
FACULTY OF NUCLEAR SCIENCES AND PHYSICAL  
ENGINEERING

Department: Department of Software Engineering  
Study programme: Applications of Informatics in Natural science



Application of machine learning in  
modelling of laser-plasma  
interactions

RESEARCH PROJECT

Author: Bc. Samuel Šitina  
Supervisor: Doc. Ing. Ondřej Klimo, Ph.D.  
Year: 2023

České vysoké učení technické v Praze  
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Akademický rok 2022/2023

## ZADÁNÍ VÝZKUMNÉHO ÚKOLU

**Student:** Bc. Samuel Šitina  
**Studijní program:** Aplikace informatiky v přírodních vědách  
**Název práce:** Aplikace strojového učení při modelování interakce laserového záření s plazmatem

### Pokyny pro vypracování:

1. Seznamte se s Particle-in-Cell (PIC) kódem EPOCH [1] či SMILEI [2] za účelem simulací interakce laserového záření s terčí [3].
2. Navrhněte a implementujte způsob automatického spouštění simulací (vzorkování daného prostoru fyzikálních parametrů) a vyhodnocování jejich výsledků (soustřeďte se na teplotu horkých elektronů a absorpci laserového záření). Můžete použít a upravit některý z již vytvořených nástrojů.
3. Seznamte se s vícevrstvou neuronovou sítí [4].
4. Vytvořte matematický model absorpce laserového záření v terči pomocí vícevrstvé neuronové sítě.

### Doporučená literatura:

[1] T. D. Arber, K. Bennett, C. S. Brady, A. Lawrence-Douglas, M. G. Ramsay, N. J. Sircombe, P. Gillies, R. G. Evans, H. Schmitz, A. R. Bell, and C. P. Ridgers. "Contemporary particle-in-cell approach to laser-plasma modelling". In: Plasma Phys. Control. Fusion 57 (2015), p. 113001. doi: 10.1088/0741-3335/57/11/113001.

[2] J. Derouillat, A. Beck, F. Pérez, T. Vinci, M. Chieramello, A. Grassi, M. Flé, G. Bouchard, I. Plotnikov, N. Aunai, J. Dargent, C. Riconda and M. Grech, SMILEI: a collaborative, open-source, multi-purpose particle-in-cell code for plasma simulation, Comput. Phys. Commun. 222, 351-373 (2018), arXiv:1702.05128

[3] Sheng Zheng-Ming, Weng Su-Ming, Yu Lu-Le, Wang Wei-Min, Cui Yun-Qian, Chen Min, and Zhang Jie. "Absorption of ultrashort intense lasers in laser-solid interactions". In: Chinese Physics B 24 (2015), p. 015201. doi: 10.1088/1674-1056/24/1/015201.

[4] Bishop, Christopher M. Pattern Recognition and Machine Learning. New York :Springer, 2006.

**Jméno a pracoviště vedoucího práce:**

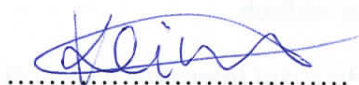
**doc. Ing. Ondřej Klimo, Ph.D.**

Katedra fyzikální elektroniky, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

**Datum zadání výzkumného úkolu: 14. 10. 2022**

**Termín odevzdání výzkumného úkolu: 31. 8. 2023**

V Praze dne 12.10.2022



vedoucí práce



vedoucí katedry

### **Statement of originality**

Hereby I declare that this paper is my original authorial work, which I have worked out on my own with the guidance of my supervisor. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed in complete reference to the due source.

In Prague on .....

.....  
Bc. Samuel Šitina

## **Acknowledgment**

I would like to thank my supervisor Doc. Ing. Ondřej Klimo, Ph.D. for valuable guidance throughout the entire process. I would like to thank my parents who consistently support me in unimaginable ways. I would also like to thank my roommate Michal who somehow does not stop showing me what can I do better by giving me an example how to perform on the highest level.

A huge thank you goes to my cycling partner Tomáš and my bicycle for keeping me sane despite everything that has been happening lately.

Bc. Samuel Šitina

*Title:* Application of machine learning in modelling of laser-plasma interactions

*Author:* Bc. Samuel Šitina

*Study programme:* Applications of Informatics in Natural science

*Type of thesis:* Research project

*Supervisor:* Doc. Ing. Ondřej Klimo, Ph.D. Department of Physical Electronics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague

*Abstract:* Hot electron temperature in the interaction between ultra-intense short-pulse laser and solid-density plasma is modelled using a neural network. The dataset consists of the results of 243 particle-in-cell simulations using EPOCH code executed with different simulation parameters of laser intensity, angle of incidence and characteristic scale of the pre-plasma. The automation of EPOCH results postprocessing employing an explicit method for exponential-sum fit is presented and implemented. The simulation results correspond with previous published research in this field. It was shown that over-parametrized neural network can be trained to produce smooth predictions. If the dataset is enlarged, this approach will become more precise and it might speed up the hot electron temperature approximation without much compromise.

*Keywords:* PIC, deeplearning, absorption, laser, hot electron

*Abstrakt česky :* Teplota horkých elektronů při interakci ultraintenzivního krátkého laserového pulsu s plazmatem o hustotě pevné látky je modelována pomocí neuronové sítě. Soubor dat se skládá z výsledků 243 simulací pomocí kódu EPOCH založeného na metodě Particle-in-cell provedených s různými simulačními parametry intenzity laseru, úhlu dopadu a charakteristické délky hustotního profilu plazmatu. Vytvořili jsme automatický postup zpracování výsledků kódu EPOCH využívající explicitní metodu pro aproximaci dat součtem exponenciálních funkcí. Výsledky simulace odpovídají dříve publikovanému výzkumu v této oblasti. Ukázalo se, že příliš parametrizovaná neuronová síť lze trénovat tak, aby produkovaly výsledky s hladkým průběhem hodnot. Pokud se soubor dat rozšíří, bude tento přístup přesnější a může urychlit aproximaci teploty horkých elektronů.

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Particle-in-Cell in laser-plasma interactions</b>	<b>3</b>
1.1 The computational cycle . . . . .	3
1.2 Hot electron production . . . . .	7
<b>2 The electron temperatures</b>	<b>9</b>
2.1 Exponential-sum fitting . . . . .	10
<b>3 Neural Networks</b>	<b>16</b>
3.1 Deep neural network . . . . .	16
3.2 Backpropagation . . . . .	17
3.3 Activation functions . . . . .	18
3.4 Small datasets . . . . .	19
3.5 Tensorflow . . . . .	22
<b>4 Results</b>	<b>24</b>
4.1 EPOCH simulations and electron temperatures . . . . .	24
4.1.1 Simulation results . . . . .	24
4.1.2 Exponential-sum fit results . . . . .	25
4.1.3 The dataset . . . . .	28
4.2 Tuning the network . . . . .	30
4.2.1 The final model . . . . .	33
4.3 Discussion . . . . .	33
<b>Conclusion</b>	<b>39</b>
<b>Bibliography</b>	<b>40</b>
<b>List of Figures</b>	<b>44</b>
<b>Attachments</b>	<b>46</b>
<b>A EPOCH input file - input.deck</b>	<b>46</b>

# Introduction

Plasma, the fourth state of matter, constitutes 99% of the visible universe. On Earth, plasmas are present in plasma torches, street lightning but also in fusion devices, which have the potential to change the way we generate electricity [16]. Plasma has been observed in technology since 1810 in various cases of electrical discharge. However, a deeper study of plasma has begun in the 1950s in the context of a controlled nuclear fusion [5], [11]. Nowadays, there is little to no doubt that it is worth pursuing a deeper knowledge of plasma behavior as well as its interaction with other forms of matter and energy.

When it comes to studying plasma, there are many technological difficulties that need to be overcome to create precise, but safe environment for a high quality research. The rapid advances in the Computer Science made it possible to use simulations based on current mathematical models letting us explore the complex processes of plasma without the necessity of doing a real experiment. This offers a chance for software engineers to make a contribution to the plasma research.

In this thesis we are interested in laser-plasma interactions which are often simulated using particle-in-cell (PIC) methods. These date back to 1970s, but since then there have been made radical improvements in the algorithms themselves [3] and, of course, in the computational power. There are many widely used open-source PIC codes such as EPOCH, Smilei, OSIRIS, VPIC and many others. We will be using EPOCH.

The goal of this thesis is to apply neural network to one particular type of simulation and show the possibility of prediction of the temperature of the hot electrons that are produced in laser-plasma interactions. The temperatures differ based on the initial parameters of the simulation. This is a standard machine learning problem, because the data-set is simply a set of parameters describing some physical properties. We will obtain the data by running a particular type of simulation using EPOCH code. It is worth mentioning that the data-set to our neural-network is quite small, because the simulations are time-consuming, so there will also be discussion about how to deal with small datasets.

Despite the fact that this thesis is about applying a machine learning technique, it is necessary to briefly mention the physical principles which underlie the used simulation. PIC method and the physics behind laser-plasma interaction will be the topics of the first chapter.

In the processing of the simulation results, there is a sub-problem of identifying the electron temperatures. After a closer look, it will be clear that it is an equivalent of



fitting a function that is a sum of exponential distributions. This problem is not new and there already exists a solution [19], [22]. However, as it is an important problem that needed to be addressed and as there is a chance that our implementation of the solution might be used in the future by other researchers in this field, we will dedicate second chapter to this.

The first widely reported commercial application of neural networks is more than 30 years old. Since then, neural networks gained a reputation of great applicability to a wide range of problems [38]. Today, with powerful computers and easily accessible libraries such as PyTorch or TensorFlow, creating a functioning neural network tailored to a specific case takes only a little effort. What is not always so easy is the fine-tuning and a correct interpretation of the prediction made by a neural network. For example, in [15], there has been made an argument that the correct interpretation can be truly difficult and that certain network architectures are vulnerable to being 'fragile' when it comes to interpretation. Although wrong interpretation does not make the neural network useless, we should be careful in any claims about the network capabilities before we have the results that back them up.

The principles of how a deep neural network works and how it can be trained will be summarized in the third chapter. How we trained our particular network, what we did to maximize its reliability by fine-tuning and the results of the best version will be presented in chapter four.

# Chapter 1

## Particle-in-Cell in laser-plasma interactions

PIC method was introduced by Oscar Buneman and John Dawson more than 60 years ago. They simulated the motion of more than 1000 plasma particles on what Dawson called *a high-speed computing machine* [10], [13]. Dawson motivates these simulations by stating that thanks to them *we can obtain as complete information as we desire about the motion of the system* and therefore *the theoretical predictions may be checked in more complete detail than it is possible with real plasma* [13]. This idea is the essence of the computational physics. It is probably the reason why this field has gained so much popularity.

The approach of the modern PIC codes is a little bit different than it was 60 years ago. The improvements include using higher order shape functions, using flux of charge to evaluate the interaction between fields and particles instead of directly calculating moments of distribution functions and others [3]. Also, we are now able to include other physical effects such as collisions and QED processes. Although these can improve the precision of the simulation in some cases, in other cases these additions are not necessary, simply because the benefit from improved precision does not exceed the computational costs.

If we wanted to roughly quantify the improvement that has been made with the power of the computers and the algorithms, we can say that on modern - as we call them - *supercomputers* we can work with more than  $10^{10}$  plasma particles [36].

In this chapter we will discuss the general numerical scheme of PIC simulation, but also the basic physical principles on which it is based. Even though it is not what we want to implement ourselves, we think it might be easier for the reader to think about the simulations once he understands the idea behind it.

### 1.1 The computational cycle

The simulation runs in a cycle. In each step, we solve for electromagnetic fields created by the charged particles. Then we evaluate the equations of motion that is

caused by the Lorentz force [7]. The laser pulse is included as an external source of electromagnetic radiation at the boundary.

Usually, the finite-difference time-domain method (FDTD) is used for numerically solving Maxwell's equations, which fully describe the electromagnetic field. *Finite-difference* means that electric field  $\vec{E}$  and magnetic field  $\vec{B}$  are specified on the points of a grid - usually a Yee grid. A detailed description of a Yee grid can be found in the original article: [40]. The most important idea of a Yee grid is hinted in figure 1.1 - the components of the magnetic field are calculated in the middle of the faces of an imaginary cube while the electric field is calculated in the middle of the edges. The cube represents one cell of the 3-dimensional grid. We can write derivatives of electric field [3]:

$$\left(\frac{\partial E_y}{\partial x}\right)_{i+\frac{1}{2},j,k} = \frac{E_{y,i+1,j,k} - E_{y,i,j,k}}{\Delta x} \quad (1.1)$$

More importantly, this derivative happens to be second order accurate at the point of the cube where we calculate  $B_{z,i,j,k}$ , because the formula is centered. Also, because of the Maxwell's equations, this derivative is exclusively used to calculate time-derivative of  $B_{z,i,j,k}$ . A similar relationship can be found when calculating all the components of  $\vec{B}$  from  $\vec{E}$  and vice versa. Therefore, all used numerical derivatives are second order accurate [3].

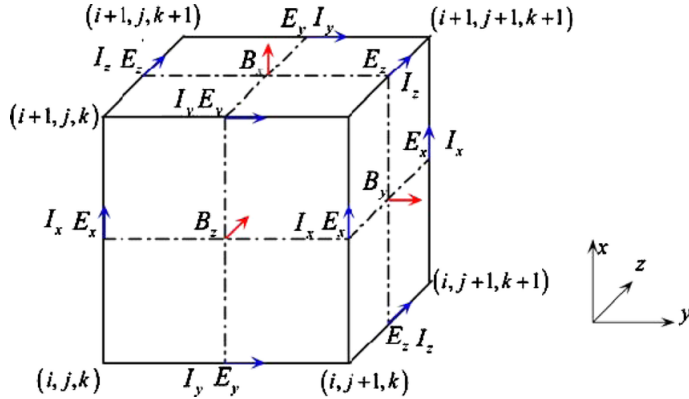


Figure 1.1: An illustration of a Yee grid [37]

## Updating the fields

In EPOCH, but also in other PIC codes, fields are updated at the half time-step as well as at the full time-step. The first part - time-step from  $n$  to  $n + 1/2$  - uses currents calculated at  $n$ :

$$\vec{E}^{n+1/2} = \vec{E}^n + \frac{\Delta t}{2} \left( c^2 \nabla \times \vec{B}^n - \frac{\vec{J}^n}{\epsilon_0} \right) \quad (1.2)$$

$$\vec{B}^{n+1/2} = \vec{B}^n - \frac{\Delta t}{2} (\nabla \times \vec{E}^{n+1/2}) \quad (1.3)$$

where  $\vec{J}$  is the current density and  $\Delta t$  is a size of a full time-step [3].

The second step - with the updated currents - is similar:

$$\vec{B}^{n+1} = \vec{B}^{n+1/2} - \frac{\Delta t}{2} (\nabla \times \vec{E}^{n+1/2}) \quad (1.4)$$

$$\vec{E}^{n+1} = \vec{E}^{n+1/2} + \frac{\Delta t}{2} \left( c^2 \nabla \times \vec{B}^{n+1} - \frac{\vec{J}^{n+1}}{\epsilon_0} \right) \quad (1.5)$$

## Moving the particles

In general, the equations of motion can be written as follows:

$$\frac{d\vec{x}_l}{dt} = \vec{v}_l \text{ and } \frac{d\vec{p}_l}{dt} = \vec{F}_l \quad (1.6)$$

where vectors  $\vec{x}_l$ ,  $\vec{v}_l$  and  $\vec{p}_l$  represent the position, velocity and momentum of the  $l$ -th macro-particle.  $\vec{F}_l = \vec{F}_l(t, \vec{x}_l, \vec{v}_l, \vec{E}, \vec{B})$  is the force. Fields  $\vec{E}$  and  $\vec{B}$  are functions of the positions and velocities of the all charged particles [36]. In this case, the right hand side of the second equation is the Lorentz force and the time-step formula for the momentum can be written as [3]:

$$\vec{p}_l^{n+1} = \vec{p}_l^n + q_l \Delta t \left[ \vec{E}^{n+1/2}(\vec{x}_l^{n+1/2}) + \vec{v}_l^{n+1/2} \times \vec{B}^{n+1/2}(\vec{x}_l^{n+1/2}) \right] \quad (1.7)$$

where  $q_l$  is the charge of the  $l$ -th particle. The velocity can be calculated from the momentum using:

$$\vec{v}_l = \frac{\vec{p}_l}{\gamma_l m_l} \quad (1.8)$$

where  $m_l$  is the mass of the particle and  $\gamma_l = [p_l^2/(m_l^2 c^2) + 1]^{1/2}$  is the corresponding gamma-factor [3].

The update of the particle position is calculated from the velocity, but this is also done in more than one step. First, we calculate movement of half time-step from the old velocity as we need it to update the momentum in equation 1.7 [3]:

$$\vec{x}_l^{n+1/2} = \vec{x}_l^n + \frac{\Delta t}{2} \vec{v}_l^n \quad (1.9)$$

In similar way, we can then calculate  $\vec{x}_l^{n+1}$  and  $\vec{x}_l^{n+3/2}$  [3], which are needed for calculating the currents.

The currents which we need for updating the fields can be calculated for instance using the mechanism presented in [14]. In general, the modern approach to calculating the currents is based on solving the continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{J} = 0 \quad (1.10)$$

where  $\rho$  is the charge density and  $\vec{J}$  is the electric current. Equation 1.10 can be written in finite differences [14]:

$$\frac{\rho_{i+1/2,j+1/2,k+1/2}^{n+1} - \rho_{i+1/2,j+1/2,k+1/2}^n}{\Delta t} + \frac{J_{x,i+1,j+1/2,k+1/2}^{n+1/2} - J_{x,i,j+1/2,k+1/2}^{n+1/2}}{\Delta x} + \dots = 0 \quad (1.11)$$

The charge density is calculated separately using so-called *form-factors* of particular macro-particles:

$$\rho_{i,j,k} = \sum_l Q_l S_{i,j,k}(x_l, y_l, z_l) \quad (1.12)$$

where  $Q_l$  is the charge and  $S_{i,j,k}(x_l, y_l, z_l)$  is the form-factor. During the motion of the macro-particles the total charge should not change. Because of that, the form-factors of all macro-particles must satisfy the condition [14]:

$$\sum_{i,j,k} S_{i,j,k}(x_l, y_l, z_l) = 1 \quad (1.13)$$

Next, it is possible to write the exact formula for calculating the currents. It can be found in [14].

As the macro-particle of the simulation represents many real particles, a *weight* is assigned to each macro-particle. If we knew the distribution of the particles in the macro-particle, calculating the weight would be easy. But even though we do not know the distribution exactly, we could choose a function that would represent the distribution - these functions are often called *shape functions* as they refer to the physical shape of the macro-particles [3].

Any function with a unit integral and compact support can be used as a shape function. An even distribution of particle in volume  $\Delta x \times \Delta y \times \Delta z$  is referred to as the 'top hat' shape function. It was already mentioned, that using higher order shape functions are one of the improvements programmers were able to make thanks to more powerful computers. A higher order shape functions are for example triangular shape functions with volume of  $2\Delta x \times 2\Delta y \times 2\Delta z$ . The weight is then calculated as a convolution of shape function with the 'top hat' function [3].

## Particle collisions and other effects

Because of the nature of working with macro-particles instead of all particles, some effects are neglected. The interactions effective over a range shorter than  $\Delta x$  must be taken into account in other way. However, it is worth noting that for example collisions can be ignored if the studied plasma has low density and high temperature [3].

In particular for collisions, EPOCH uses technique described in [32]. It is a fully relativistic energy-conserving binary collision model. The random generation of scattering angles is restricted so it prefers small angle scattering. Exclusion of large angle scattering improves the behavior of the simulation with limited number of particles per cell. The input file of EPOCH code allows us to choose whether we want to include collisions. In our case, we want to work with laser intensities larger than  $10^{16}\text{W.cm}^{-2}$  and according to the conclusion of [12], including collisions is therefore not necessary.

Other effects such as ionization and quantum effects like photon emission or photon pair production are also often included. A description of situations when these effects are eligible and deeper explanation of the corresponding methods can be found in [3].

## 1.2 Hot electron production

Several different processes have been proposed to describe ultrashort intense laser pulse absorption in the plasma on the surface of a solid target. These mostly depend on several parameters of the laser pulse, the target and geometry of the interaction [34].

In this work, we concentrate on the interaction which is accompanied by generation of copious amounts of hot electrons with temperature in the sub-relativistic domain ( $E_k < mc^2$ ).

This is motivated by possible application of the presented work for optimization of X-ray sources based on laser-plasma interaction, where the characteristic X-ray generated by ionization of atoms with hot electrons have mostly energy in the range up to tens of keV.

The most important laser absorption processes which are relevant for this work are: vacuum heating, resonance absorption, ponderomotive (of  $\vec{J} \times \vec{B}$ ) heating [34]. The former two require p-polarized laser beam and oblique incidence on a finite preplasma where the absorption process depends on the characteristic scale length of this preplasma and the laser incidence angle. These processes are mostly dominant in the laser intensity regime, where the velocity of electron oscillation in the laser field is sub-relativistic ( $I < 1.37 \times 10^{18}\text{Wcm}^{-2}$  for  $1\mu\text{m}$  laser wavelength) [12]. In the higher laser intensity regime, the electrons are driven by the  $\vec{v} \times \vec{B}$  component of the Lorentz force directly and the absorption due to this force dominates at normal incidence (normal with respect to the target surface). In this regime, the absorption increases with the density scale length as it is accompanied by the hole boring [12], which enables the laser pulse to propagate deeper into the target and interact with the plasma in more favorable conditions (standing wave is formed inside the plasma channel formed due to hole boring [4]).

The theoretical models of laser absorption do not provide conclusive answer about the hot electron temperature depending on the parameters of the interaction. There exist several scaling laws providing the hot electron temperature scaling with laser intensity, but this is not sufficient for applications like optimization of the hot elec-

trons for generation of secondary radiation sources. Running simulations for every particular set of condition to create and tabulate an enough dense set of parameters does not seem to be feasible either. Therefore, the neural network approach may possibly provide an interesting alternative of finding the optimum or at least narrowing the range of parameters where the global optimum can be found.

# Chapter 2

## The electron temperatures

The result of the PIC simulation using EPOCH contains a lot of information. As we mentioned, we are interested in the temperatures of electrons. In other words, we would like to find the energy distribution of hot electrons and their characteristic parameters. EPOCH provides the momenta  $\vec{p}_e$ , but using simple relativistic formula one can easily calculate the kinetic energy  $E_k$  as:

$$E_k = m_0 \cdot c^2 \left( \sqrt{1 + \left( \frac{p}{m_0 \cdot c} \right)^2} - 1 \right) \quad (2.1)$$

where  $p = \sqrt{\vec{p} \cdot \vec{p}}$ ,  $m_0 = 9.109 \cdot 10^{-31} \text{kg}$  is the electron rest-mass and  $c = 3 \cdot 10^8 \text{m.s}^{-1}$  is the speed of light [27].

An example of the histogram from our simulations can be seen in figure 2.1. Logarithmic scale allow us to see on which parts of the spectrum which temperature dominates. Here in our example, there are two main parts which can be seen with a naked eye, but that usually does not mean that those are the only two. Also, it is apparent that the energy distribution of electrons is not perfectly smooth and some bins of the histogram have the value of zero. In real world, it is unlikely that something like this would actually happen. In the results of a simulation like ours, it is not that rare, but it must be taken into an account that to some parts of the spectrum cannot be assigned the same level of statistical significance as to others. For each temperature, we assume an exponential (Boltzmann) distribution of electron energy:

$$N = N_0 \cdot \exp \left( -\frac{E}{k_B \cdot T} \right) \quad (2.2)$$

where  $N$  is the number of electrons,  $T$  is the temperature and  $k_B = 1.38 \cdot 10^{-23} \text{J.K}^{-1}$  is the Boltzmann constant [27]. It is desired to plot the histogram in a logarithmic scale because the electron counts vary in several orders of magnitude.

Several processes may contribute to laser absorption and produce hot electrons with different temperatures simultaneously in a single experiment/simulation. This situation is also present in our study, which complicates the automation of extracting the



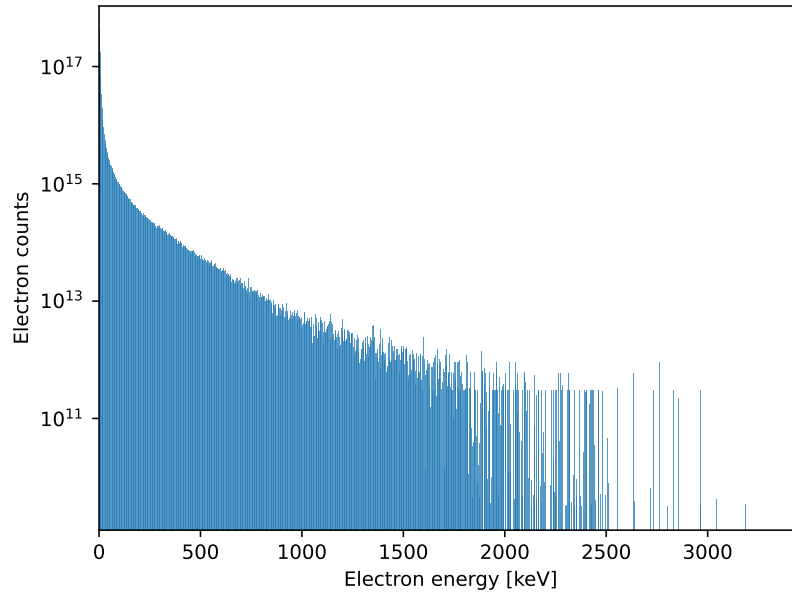


Figure 2.1: An example of electron energy distribution with intensity of laser  $I = 10^{19} \text{W.cm}^{-2}$ , characteristic scale of the exponential preplasma profile  $L = 0.1 \mu\text{m}$  and angle of incidence with respect to target normal direction  $\alpha = 10^\circ$ .

hot electron temperature from the simulation results. From now on, we will assume that each of the electrons belongs to one of the Boltzmann distributions describing its energy and that the final histogram is a accumulation of all these distributions. In practice, that means we have to fit sum of more than one exponential distributions.

In this chapter, we will present what we found was a reliable method of fitting the electron energy histogram and finding the hot electron temperature using an explicit algorithm for exponential-sum fitting.

## 2.1 Exponential-sum fitting

Exponential-sum fitting is not a new problem. Princeton University professor For-man S. Acton in his book *Numerical Methods That Work* from 1970 among other things wrote an essay called *What not to compute*. In this essay, he mentions the fitting of sum of exponential functions and describes it as *extremely ill conditioned* [2]. In this thesis, we had no choice but to find a way how to find the electron temperatures despite it clearly being a generally difficult problem. The task is to find the parameters  $m, a_0, a_1, \dots, a_m, b_1, \dots, b_m$  so that the function

$$f(x) = a_0 + \sum_{i=1}^m a_i e^{b_i x} \quad (2.3)$$

generalizes the data as good as possible. Formally, for a set of data ( $n$  data points)

$((x_1, y_1), \dots, (x_n, y_n))$  we are minimizing error function

$$e = \sum_{i=1}^n w_i (y_i - f(x_i))^2 \quad (2.4)$$

where  $w_i$  are the weights.

Modern programming libraries allow us to use of well-known and usually efficient methods like non-linear least squares. However, these methods do not produce satisfying results in this case due to the mentioned ill-conditioning. The quality of the initial guess is too important for these methods to converge.

As of today, there has already been done a significant amount of work trying to provide reliable method for solving this problem. We can mention Prony's method [31] and its variations proposed for example in [29] - it was originally meant for signal approximation (finding of complex parameters in the exponential functions, similar to Fourier transformation), but it has also other applications.

Another, much simpler method worth mentioning is called *successive subtraction* [39]. It is often used in case of exponential decays (negative parameters  $b_i$ ) which resembles our problem of exponential distributions. The main idea of this method is to only fit the end of the decay with one exponential function, subtract the result from the data and in iterative manner find all exponential components. It is very simple and it is a good candidate for our problem since we mostly want to know the temperature of the hottest electrons - *the tail*. However, the automation of this method also can get difficult, because it is not self-evident how to choose an interval that can be fitted as a *tail* in each iteration.

If the goal is to fit only the hottest electrons, the standard approach is to manually select a reasonable part of the tail of the distribution and fit it with one exponential function. For example in [12], where they are studying similar problem as us they seem to be using this to find the temperature of hot electrons. The nature of their research probably allowed them to do it manually for each simulation. We, on the other hand, and this is the point of this discussion, have to find a way to automate it.

Next method one might consider is presented in [39]. It is called *exponential sum fitting of transmissions* (ESFT) and was developed for fitting radiative transmission functions in context of atmospheric research. It is an iterative method and it might be an interesting alternative if the reliability of more simple methods will not be sufficient for our problem.

We only mentioned a few methods. A summary with a deeper explanation of these and other methods can be found in [39], [20] or in [19].

We should not forget that we attempted to train a neural network for fitting the histogram. The idea was that if we train it on particular range of parameters which we expect from our simulations, it might produce reasonable results at least on this part of the 'spectrum'. As it turns out, it is extremely hard to even create a dataset that resembles the real-world data. What is more, the ill-conditioning of this whole

problem is still present and the neural network designed by us simply could not produce valuable results.

## The 'Jacquelin' method

We will call the method we chose *Jacquelin method*, because we will be referencing an article by J.Jacquelin who developed it for his own purpose [22]. This method is explicit and non-iterative which is its main advantage. The original derivation presents a universal strategy how to explicitly approximate any function that is a solution to some integral or differential equation. Let us start with derivation the numerical algorithm for a function

$$y(x) = a_0 + a_1 e^{b_1 x} + a_2 e^{b_2 x} \quad (2.5)$$

We start by calculating these integrals:

$$\begin{aligned} S(x) &= \int_{x_0}^x y(t) dt \\ &= \int_{x_0}^x a_0 + a_1 e^{b_1 t} + a_2 e^{b_2 t} dt \\ &= a_0(x - x_0) + \frac{a_1}{b_1}(e^{b_1 x} - e^{b_1 x_0}) + \frac{a_2}{b_2}(e^{b_2 x} - e^{b_2 x_0}) \\ \\ SS(x) &= \int_{x_0}^x S(t) dt \\ &= \int_{x_0}^x \int_{x_0}^t y(u) du dt \\ &= \int_{x_0}^x a_0 t + \frac{a_1}{b_1} e^{b_1 t} + \frac{a_2}{b_2} e^{b_2 t} - \left( \frac{a_1}{b_1} e^{a_1 x_0} + \frac{a_2}{b_2} e^{a_2 x_0} + a_0 x_0 \right) dt \\ &= \frac{1}{2} a_0 x^2 - \left( \frac{a_1}{b_1} e^{a_1 x_0} + \frac{a_2}{b_2} e^{a_2 x_0} + a_0 x_0 \right) (x - x_0) - \frac{1}{2} a_0 x_0^2 \\ &\quad + \frac{a_1}{b_1^2} (e^{b_1 x} - e^{b_1 x_0}) + \frac{a_2}{b_2^2} (e^{b_2 x} - e^{b_2 x_0}) \end{aligned}$$

Now, if we reorganize the terms it can be shown that there are constants  $A$ ,  $B$ ,  $C$ ,  $D$  and  $E$  so that  $y(x)$  can be written as:

$$y(x) = A \cdot SS(x) + B \cdot S(x) + Cx^2 + Dx + E \quad (2.6)$$

where  $A = b_1 b_2$ ,  $B = -(b_1 + b_2)$ . We only need  $A$  and  $B$ , because we can rewrite it as:

$$b_1 = \frac{1}{2} \left( B + \sqrt{B^2 + 4A} \right) \quad (2.7)$$

$$b_2 = \frac{1}{2} \left( B - \sqrt{B^2 + 4A} \right) \quad (2.8)$$

The equation 2.6 is linear in the unknown parameters and therefore it can be rewritten as:

$$\mathbf{y} = \mathbf{X} \cdot \mathbf{b} \quad (2.9)$$

where  $\mathbf{y}$  is vector:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad (2.10)$$

after we set  $S_k = S(x_k)$  and  $SS_k = SS(x_k)$  we can write:

$$\mathbf{X} = \begin{pmatrix} SS_1 & S_1 & x_1^2 & x_1 & 1 \\ SS_2 & S_2 & x_2^2 & x_2 & 1 \\ SS_3 & S_3 & x_3^2 & x_3 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ SS_n & S_n & x_n^2 & x_n & 1 \end{pmatrix} \quad (2.11)$$

and

$$\mathbf{b} = \begin{pmatrix} A \\ B \\ C \\ D \\ E \end{pmatrix} \quad (2.12)$$

and we can use the well-known least squares method to calculate the parameters  $A, \dots, E$ . We sort the data so that if  $i < j$ , then  $x_i < x_j$  for  $\forall i, j \in \{1, \dots, n\}$ . The integrals are computed numerically as:

$$S_i = \begin{cases} 0 & i = 0 \\ S_{i-1} + \frac{1}{2}(y_i + y_{i-1})(x_i - x_{i-1}) & i \in \{2, \dots, n\} \end{cases} \quad (2.13)$$

and

$$SS_i = \begin{cases} 0 & i = 0 \\ SS_{i-1} + \frac{1}{2}(S_i + S_{i-1})(x_i - x_{i-1}) & i \in \{2, \dots, n\} \end{cases} \quad (2.14)$$

Now, the solution to the linear equation 2.9 can be written as [28]:

$$\mathbf{b} = (\mathbf{X}^T \mathbf{X})^{-1} \cdot (\mathbf{X}^T \mathbf{y}) \quad (2.15)$$

which in our case can be expanded to:

$$\begin{pmatrix} A \\ B \\ C \\ D \\ E \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n SS_i^2 & \sum_{i=1}^n SS_i S_i & \sum_{i=1}^n SS_i x_i^2 & \sum_{i=1}^n SS_i x_i & \sum_{i=1}^n SS_i \\ \sum_{i=1}^n SS_i S_i & \sum_{i=1}^n S_i^2 & \sum_{i=1}^n S_i x_i^2 & \sum_{i=1}^n S_i x_i & \sum_{i=1}^n S_i \\ \sum_{i=1}^n SS_i x_i^2 & \sum_{i=1}^n S_i x_i^2 & \sum_{i=1}^n x_i^4 & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^2 \\ \sum_{i=1}^n SS_i x_i & \sum_{i=1}^n S_i x_i & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n SS_i & \sum_{i=1}^n S_i & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i & n \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^n SS_i y_i \\ \sum_{i=1}^n S_i y_i \\ \sum_{i=1}^n x_i^2 y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n y_i \end{pmatrix} \quad (2.16)$$

After we compute  $b_1$  and  $b_2$  from  $\mathbf{b}$ , we still have 3 unknown parameters  $a_0$ ,  $a_1$  and  $a_2$ . For those, we will take the original equation 2.5. As it is already linear in the parameters  $a_0$ ,  $a_1$  and  $a_2$ , we only need to pre-compute the values of  $\alpha_k = e^{b_1 x_k}$  and  $\beta_k = e^{b_2 x_k}$ . One can see that we get a very similar equation as 2.16, that can be written in this form:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} n & \sum_{i=1}^n \alpha_i & \sum_{i=1}^n \beta_i \\ \sum_{i=1}^n \alpha_i & \sum_{i=1}^n \alpha_i^2 & \sum_{i=1}^n \alpha_i \beta_i \\ \sum_{i=1}^n \beta_i & \sum_{i=1}^n \alpha_i \beta_i & \sum_{i=1}^n \beta_i^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n \alpha_i y_i \\ \sum_{i=1}^n \beta_i y_i \end{pmatrix} \quad (2.17)$$

In [22], only the case without the constant  $a_0$  is presented, but as we have shown the derivation of the generalized version with constant is not difficult. An extension of Jacquelin method for more than two exponential terms is also not very complicated. Note that adding one term adds two parameters, which leads to matrix of size  $7 \times 7$ . However, it always leads to solving for the roots of the polynomial created by the first  $N$  elements of the vector  $\mathbf{b}$ , where  $N$  is the number of exponential terms. In case of two exponential terms, equations 2.7 solve for the roots of the polynomial:

$$b^2 - Bb - A = 0 \quad (2.18)$$

In case of three, we would need to calculate  $SSS(x)$  from  $SS(x)$ . There would also be a term  $x^3$ . If we had built  $\mathbf{X}$  as  $[SSS(x), SS(x), S(x), x^3, x^2, x^1, 1]$ , we would solve for the roots of:

$$b^3 - Cb^2 - Bb - A = 0 \quad (2.19)$$

The rest is analogous. We implemented this method in Python with the number of terms as a parameter, because the other terms are added similarly. This method is not completely stable. There is always a chance that the inverse matrices cannot be calculated numerically. To get rid of this exception, we could be using for example the Moore-Penrose pseudoinverse. Another problem might occur when looking for the roots of the polynomial - in general, the roots can be complex.

In either of these cases, reducing the number of exponential term seemed to improve the stability. That is not surprising, because there is a cumulative error in the process

of numerically calculating the integrals. Reducing the number of integrals reduces the error caused by noise, which improves the stability.

# Chapter 3

## Neural Networks

The goal of any machine learning problem is to use data to generalize some relationship between input and output. The input is usually a vector of *features*  $\mathbf{x}_i$  and corresponding output (or *target*)  $\mathbf{y}_i$  is in general also a vector. In case of regression, output might be just one number. In case of classification, the target might be represented by a vector with number of elements equal to number of classes, where the correct class has value of 1 and all the others have value of 0.

Let  $\bar{f}$  be the theoretical function that fully represents the relationship between the inputs and output. The dataset contains  $N$  pairs of input and output  $(\mathbf{x}_i, \mathbf{y}_i)$ . The dataset is divided into training set and test set, so that we have an independent way of evaluating the ability of the model to predict output based on the samples it has not been trained on [17].

### 3.1 Deep neural network

**Deep neural networks** (DNN) are deep learning models. The goal is to approximate the function  $\bar{f}$ . This is done by parameterizing the function of the chosen model  $f(\mathbf{x})$  as  $f(\mathbf{x}; \boldsymbol{\theta})$  and learning set of parameters  $\boldsymbol{\theta}$  using the set of training data to find the best approximation [17].

The structure of a DNN can be usually described as *a directed acyclic graph* and sometimes these neural networks are referred to as **deep feedforward networks** [17]. An illustration of a fully-connected deep network can be seen in figure 3.1.

Each layer of DNN represents a function on its own - the input is the previous layer and the outputs are then passed to the next layer. For example, for a network with 3 layers we can then write  $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$  where  $f^{(i)}(\mathbf{x})$  is the function corresponding to the  $i$ -th layer. The length of this chain is the *depth* of the network.

One element of a layer is called *neuron* and it too represents a function. This one is usually a linear combination of outputs from the previous layer plugged into so-called *activation function* result of which is a number [17]. The parameters of the linear combination - *weights* - have to be found by a learning process. The

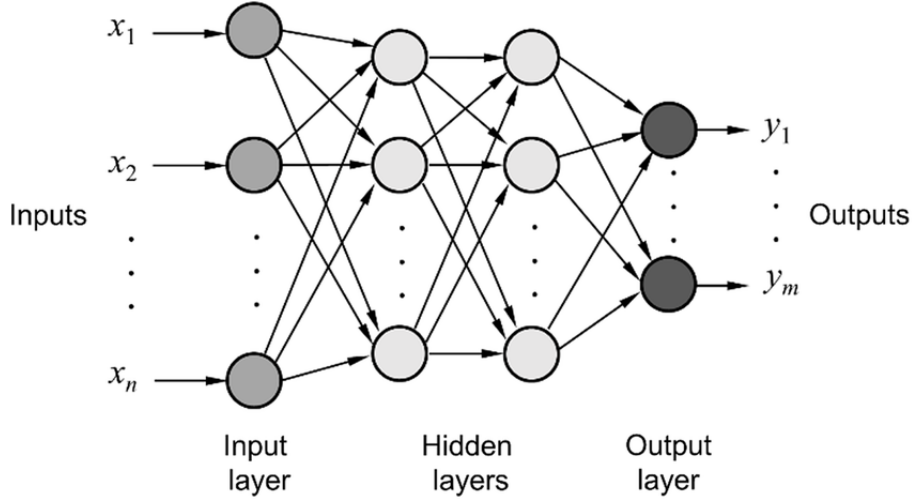


Figure 3.1: An illustration of a feed-forward neural network with multiple hidden layers [30].

activation function is chosen with other hyper-parameters of the model (for example the number of neurons or layers) and it is there to include some degree of non-linearity [8].

## 3.2 Backpropagation

Multilayer architectures, such as the one presented above, can be trained by stochastic gradient descent (SGD). This procedure is an application of the chain rule for derivatives [25]. We start by choosing a *loss* function, which is a metric quantifying the the error. There are more options that can lead to good results depending on the particular problem. In regression, the loss function is usually the *sum-of-squares error* [17]:

$$E(\boldsymbol{\theta}) = \sum_{n=1}^N E_n = \frac{1}{2} \sum_{n=1}^N \|f(\mathbf{x}_n; \boldsymbol{\theta}) - \mathbf{y}_n\|^2 \quad (3.1)$$

This is particularly convenient if the activation function of the last layer is identity, because then  $\frac{\partial E}{\partial a_k} = y_k - \bar{y}_k$ . However, a better justification of this choice is rooted in derivation of maximum likelihood function after the assumption of Gaussian noise [8].

The back=propagation can be expressed in following steps [8]:

1. Take an input vector  $\mathbf{x}_n$  and forward propagate it through the model using:

$$a_j = \sum_i w_{ji} z_i \quad (3.2)$$

where  $z_i$  is the  $i$ -th output from the previous layer and  $w_{ji}$  is the weight to  $j$ -th neuron for  $z_i$ . Let  $a_i$  be called *activation*. Output of the current neuron



with activation function  $h(\cdot)$  is:

$$o_j = h(a_j) \quad (3.3)$$

which serves as an input value  $z_j$  for neurons in the next layers.

2. Evaluate the first derivative of error function  $E$  with respect to of all activations for all outputs. For mean-square-error that can be written as:

$$\frac{\partial E_n}{\partial a_k} = o_k - y_k \quad (3.4)$$

3. If we denote  $\delta_j = \frac{\partial E_n}{\partial a_j}$ , we can write:

$$\delta_j = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad (3.5)$$

and we get the *back=propagation formula*:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad (3.6)$$

where  $k$  is the index of neuron in a layer that is higher up in the network compared to neuron  $j$  of which we are currently calculating the gradient. The equation 3.6 shows that the gradient for a particular neuron can be calculated using the gradients from the next layer.

4. We can easily see that  $\frac{\partial a_j}{\partial w_{ji}} = z_i$  from equation 3.2. Calculating the derivative of the error function with respect to particular weight  $w_{ji}$  can be then written as:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \quad (3.7)$$

This way we calculate the gradient of the error function with respect to all the weights and we can adjust the weights accordingly [8]. Real-world implementations are slightly more complicated and include other useful operations. This topic is further discussed in section 3.5.

### 3.3 Activation functions

Activation functions can differ based on the particular type of the relationship we are modeling. However, the recommended default is so-called *Rectified Linear Unit* (ReLU) defined as  $ReLU(z) = \max(0, z)$  [17]. Compared to other activation functions it has very little computational cost - it simply returns the value if the input value is positive and 0 if the input is negative. The derivatives, which are needed in the backpropagation procedure, are similar - 0 when negative and 1 when positive. The derivative is not defined in  $z = 0$ , but in practice this is solved by returning one-sided derivative - either 1 or 0 - as it does not cause any problems [17].

Before ReLU was first introduced, most neural networks used the *sigmoid* activation function  $h(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ . Its derivatives are quite small outside the range  $(-1, 1)$  and therefore it can quickly become insensitive to change. When  $|a_i|$  is large, the derivative becomes small which makes it difficult to improve the cost function effectively. This is usually called *vanishing gradient problem* and it does not occur when using for example ReLU [17].

The same problem can be present when using the hyperbolic tangent  $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$  and the arctangent function, which is not surprising as both of these have shape similar to the sigmoid.

Another alternatives are for example *softplus*( $z$ ) =  $\log(1 + e^z)$  and *Gaussian* function [23]. A quick graphical overview of the most common activation functions can be seen in figure 3.2.

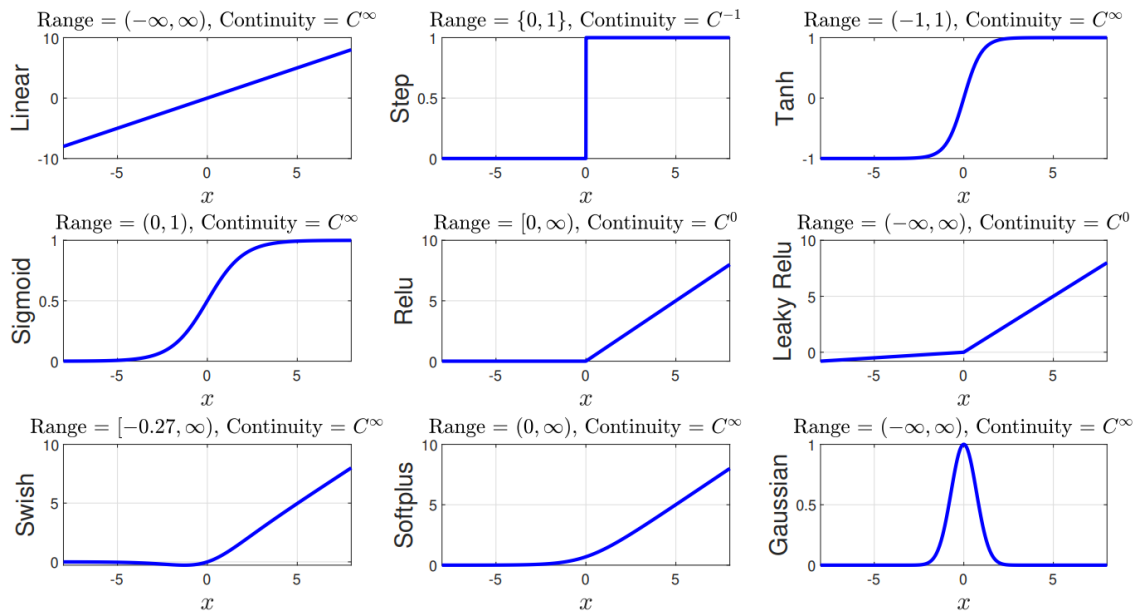


Figure 3.2: An overview of graphs of common activation functions [23].

There are lot of modifications to the simple activation functions such as *Swish*, which is a parameterized modification of the sigmoid function, *leaky ReLU*, *displaced Relu* and many others [23]. There is also a lot of time invested in researching more complex options often called *adaptive activation functions*, which are supposed to change some of their parameters during the learning process in order to optimize it. A summary of these can be found in [23].

### 3.4 Small datasets

There is a strong relationship between the size of the dataset and the predictive power of a neural network. Networks used for image classification usually start to produce reasonable predictions when the data set has tens of thousands data points. Today, it is not rare to find datasets containing as many as tens of millions of

examples and this is probably what changed this field so much in the past few years [17]. The process of collecting data for regression models might have different priorities than for classification such as noise reduction or features variability, but usually a bigger dataset is better anyway.

The standard answer to the question: *what if my dataset is small?* is *try to get a bigger one!*. However, in real world, it is sometimes either not possible or too expensive for a given research to listen to this advice. For example, in medical applications where neural network is used to detect an illness from X-ray, some illnesses or their predictors can be so rare that the dataset only contains less than 10 such observations [33]. In this section, we would like to explore and present the options for a situation like this, where obtaining additional data is simply not an option and show that having a small dataset is not something that should stop one from trying to create a functioning neural network.

## Overfitting and regularization

One of the biggest reasons why in regression we want as many data points as possible is the risk of overfitting. Even in relatively small neural networks the number of weights can get bigger than the number of available data points. This is usually called *overparametrization* and it is generally better to avoid it. However, there are some techniques that allow us to train such models so they perform well even if they are overparametrized [21]. To be more precise, it was shown that more important than the number of weights is their size (by *size* we mean a vector norm, usually  $L_1$  or  $L_2$  norm) [6]. A restrictions put on the size of the weights of a neural network is a part of what we call *regularization*.

The goal of regularization is *smoothening* of the model. This has to be implemented carefully because as we will show, it can be a trade-off for predicting power. We said before, that during the training process we are minimizing an error function defined by equation 3.1. Explicit regularization introduces a new term in this equation and changes the task to minimization of the following functional:

$$H[f] = \frac{1}{2} \sum_{n=1}^N \|f(\mathbf{x}_n; \boldsymbol{\theta}) - \mathbf{y}_n\|^2 + \lambda_R \phi(f) \quad (3.8)$$

where  $\phi(f)$  is so-called *smoothness functional* or *regularizer* and  $\lambda_R$  is a positive number usually called *regularization parameter* [18]. The parameter  $\lambda_R$  is chosen in the same way as all of the other hyperparameters during the tuning of the network. Introduction of this new term to the error slightly changes the way the gradients are calculated during the error back-propagation process, but right now, that is not important.

$L_1$  and  $L_2$  regularization are scenarios, where we set  $\phi(f) = \|\mathbf{w}\|_1$  and  $\phi(f) = \|\mathbf{w}\|_2$  respectively, where  $\mathbf{w}$  is the vector of all the weights in the network. The bigger  $\lambda_R$  the more parameters are driven to 0. This can limit the effective model complexity [8]. Tuning the network using regularization should lead to the reduction of the complexity in a way that the model performs better on a test set. If we set

$\lambda_R$  too big and the weights become too small, we might as well reduce the number of neurons or some features become irrelevant. However, as the reader might guess, that is not what we want to accomplish at this point. Needless to say, choosing the correct amount of neurons is also important and this topic is yet to be discussed.

Another technique that is quite new and has promising results is called a *dropout* method. It was only presented in 2014 and a deeper explanation can be found in the original paper [35]. We will briefly explain how it works. Dropout is not an explicit method like the first one but rather *implicit*. The main principle is to *drop out* some of the neurons out during each training step. Not all of the weights are then updated and the complexity compared to normal training is reduced. A graphic illustration can be seen in the figure 3.3.

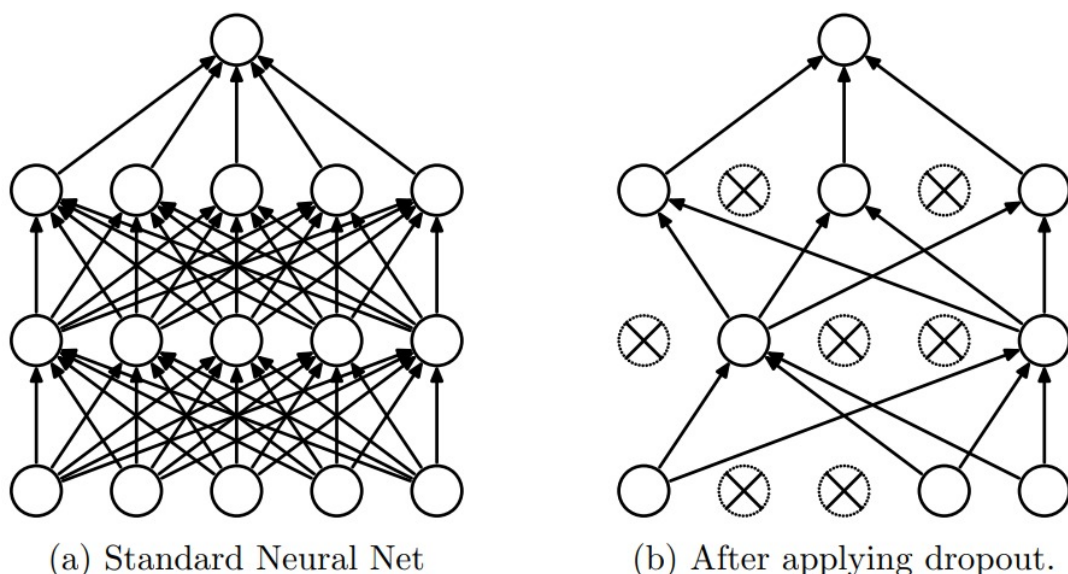


Figure 3.3: An illustration of the dropout technique [35].

Which neurons we *drop* is chosen randomly with probability  $p$ . During testing (and predicting), output of each neuron is multiplied by  $1 - p$ , which can be interpreted as an approximate averaging of all the *sub-networks* trained with reduced number of neurons. It is quite trivial and what is more important, it works well in practice [35].

Dropout can be also applied for the input layer, but there it is recommended to use smaller  $p$  [35]. We have only 3 inputs that represent parameters of the laser-plasma simulation. In this case, it would be hard to justify dropping out any one of them because the physical principles imply that the temperature of electrons should be dependent on all three of the parameters. We will therefore not use dropout on the input layer.

The last regularization technique we would like to mention is called *early stopping*. The idea is that if we have chosen the right learning-rate, we can stop the learning process just before the predicting power starts to decrease. This also is an example of *implicit regularization* and it is one of the simplest. It is based on the fact that

during training, the validation error shows a decrease at first but then tends to grow as the network starts to over-fit [8].

Although it has been demonstrated which methods tend to produce better results, the effectiveness varies from case to case. Because of that, during the fine-tuning phase one has to try a few of them and decide which settings produce the best results.

## 3.5 Tensorflow

*Tensorflow* is a large open-source cross-platform machine learning Python library. It is implemented in programming language C++ for portability and performance. It offers multiple levels of abstraction but still makes it easy to build state-of-the-art models for anyone who needs a reliable and powerful machine learning tool. The reference article describing the architecture and main properties of Tensorflow can be found in [1] and the documentation can be found in [24].

We will be using the *Keras API* which is running on top of Tensorflow. It was specifically developed for fast experimentation and for our purpose it is probably the best option. *Sequential model* allows us to build the network as a linear stack of layers where we can specify a large amount of parameters according to our needs. What is more, this library also includes regularization options like the ones we mentioned in chapter 3.4.

### Batch-size

In chapter 3.2 where we talked about back=propagation, we promised that we will elaborate on how the back=propagation is usually implemented in practice. Namely, an improvement can be made by dividing the training set into *batches*. A batch is a subset of training data that is sent through the network in one iteration of the training process [9]. Before we update the weights, we calculate the error on all of the samples in the batch. The derivative of the total error (which in the simplified approach is calculated in equation 3.7) is replaced by the sum of all derivatives across all training examples in the batch [8]:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}} \quad (3.9)$$

We mention this, because this approach is implemented in Tensorflow as it has multiple benefits. Firstly, if the process is parallelized, the forward propagation of the inputs can run simultaneously for all training samples in a batch, which saves time. Secondly, it can save memory if the datasets are large, because we can load one batch at a time. Of course, even in Tensorflow the batch technique can be suppressed with the correct choice of the batch size. This choice is also part of the fine-tuning of the network, but unless we have a very large dataset(not our case), it is reasonable

to assume that smaller batch sizes (less than 30 instances) will contribute to better results [26].

# Chapter 4

## Results

### 4.1 EPOCH simulations and electron temperatures

#### 4.1.1 Simulation results

The EPOCH simulation was run in following setting:

- The intensities of the laser:  $I = 10^{17}\text{W.cm}^{-2}$ ,  $I = 10^{18}\text{W.cm}^{-2}$  and  $I = 10^{19}\text{W.cm}^{-2}$ .
- The angle of incidence with respect to target normal direction:  
 $\alpha \in \{0^\circ, 5^\circ, 10^\circ, 20^\circ, 30^\circ, 40^\circ, 45^\circ, 50^\circ, 60^\circ\}$ .
- The characteristic scale length of the preplasma:  
 $L \in \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5\}$  in microns.
- The laser wavelength is 1 micron.
- The laser is p-polarized and focused to a Gaussian spot of size 3.2 microns.
- The density in the target ranges from about  $0.01n_c$  to  $3.\gamma_{osc}n_c$ , where  $n_c$  is critical density of laser radiation [12] and  $\gamma_{osc}$  is defined in [12].
- The initial temperature of plasma is 100 eV.
- The target is composed of electrons and protons. They are represented by 30 macro-particles per cell.
- They are represented by 30 macro-particles per cell. The resolution of spatial grid is 33nm and the time step satisfies the CFL condition [3].

The simulations have been performed on the Q3 node of the Quantum Hyperion cluster at FNSPE. The input file that used to start the EPOCH simulation can be seen in appendix A.

All combinations of mentioned options of laser intensity, angle of incidence and characteristic scale length of the preplasma give us  $3 \times 9 \times 9 = 243$  different settings. An example of electron energy distribution given by a simulation was already shown in the figure 2.1.

Before we were able to perform the fit, we had to trim the tail of the distribution because of the excess amount of empty bins. The empty bins are a sign of poor statistical significance of this part of the spectrum. To minimize its effect but to still preserve a majority of the histogram, we performed this process:

1. Start at the very end and check whether the number of empty bins is larger compared to the number the not-empty in the last twenty bins.
2. If yes, delete the last bin and repeat.
3. If no, remove all empty bins and end the loop.

Note that the fitting process does not need all energies in the fitted sample to be equidistantly spaced. Removing the remaining empty bins should therefore improve the fit performance. We also divided all the electron counts by a factor of  $10^9$  - this can be done because it has no effect on fitting the parameters in the exponent.

### 4.1.2 Exponential-sum fit results

We implemented and used the fitting technique described in section 2.1. One of the goals of this thesis was to automate running the simulations as well as approximating the hot electron temperatures. A unified way to approximate the temperatures in all histograms was used.

The quality of the fit varies, but we found by trying various scenarios that the 2-exponential fit with the constant produces the most reasonable results. Problems that can occur when using a higher number of exponential functions to fit the histograms include:

- Fit fails because of a singular matrix.
- Fit fails because of complex roots of the polynomial.
- A bad fit result because one of the exponential parameters is slightly positive, which does not correspond to the physics.

The temperature of the hot electrons  $T_{hot}$  was calculated based on the equations 2.2 and 2.5:

$$T_{hot} = -\frac{1}{b_{max}} \quad (4.1)$$

where  $b_{max} = \max(b_1, b_2)$ . The Boltzmann constant  $k_B$  from equation 2.2 is included in the units of  $T_{hot}$  - keV.



It is crucial to realize what is truly important to us - to fit the hottest temperature. It might prevent us from labeling a good fit as poor only because the graph seems to be off. A good example of this might be the fit shown in the figure 4.1.

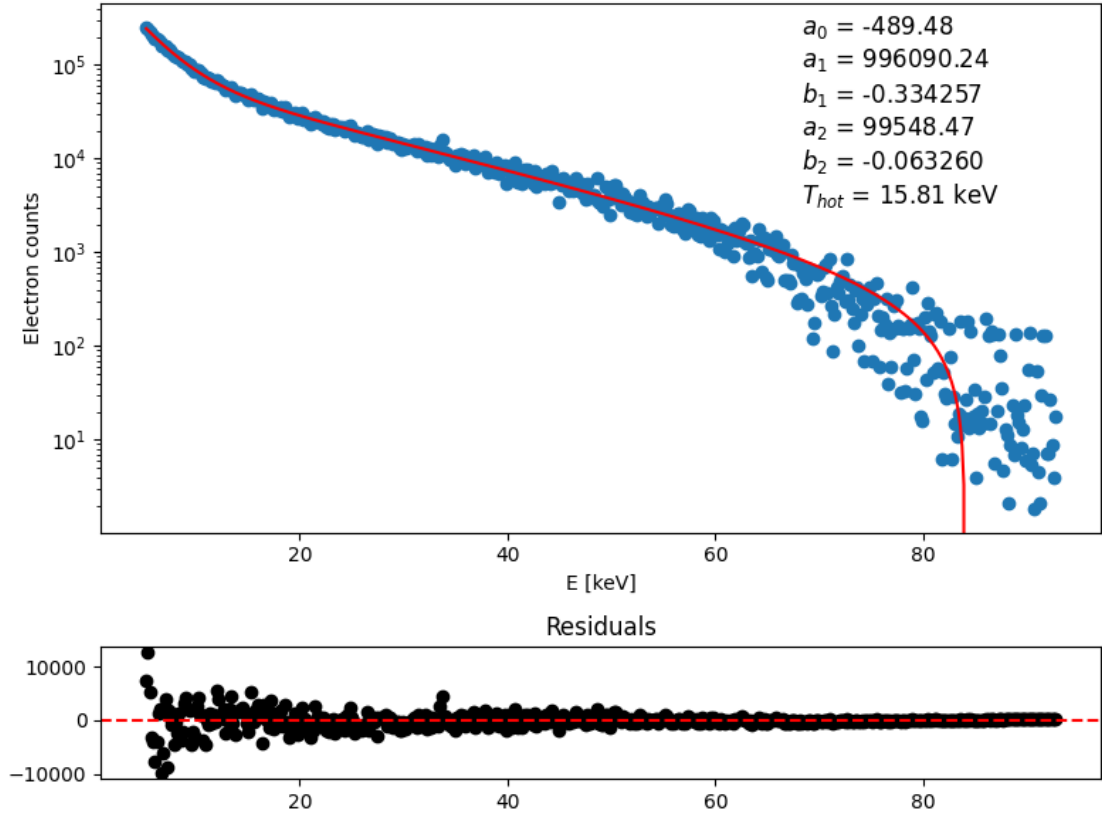


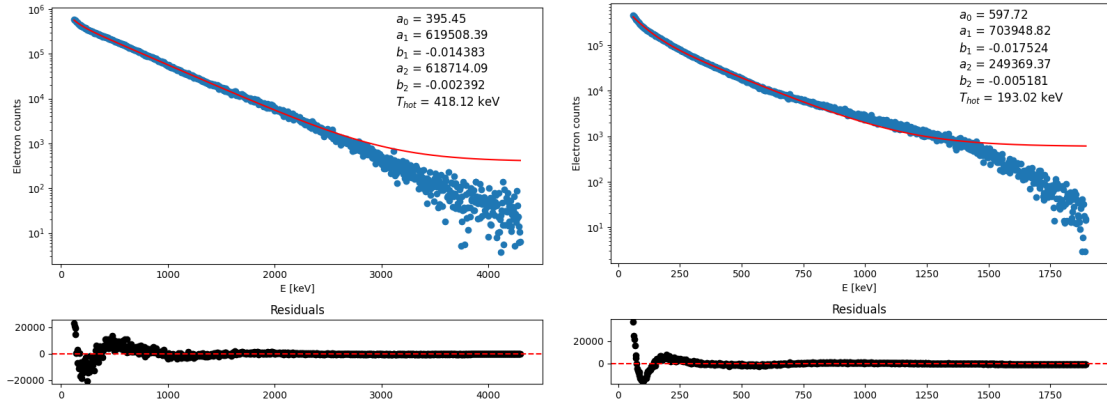
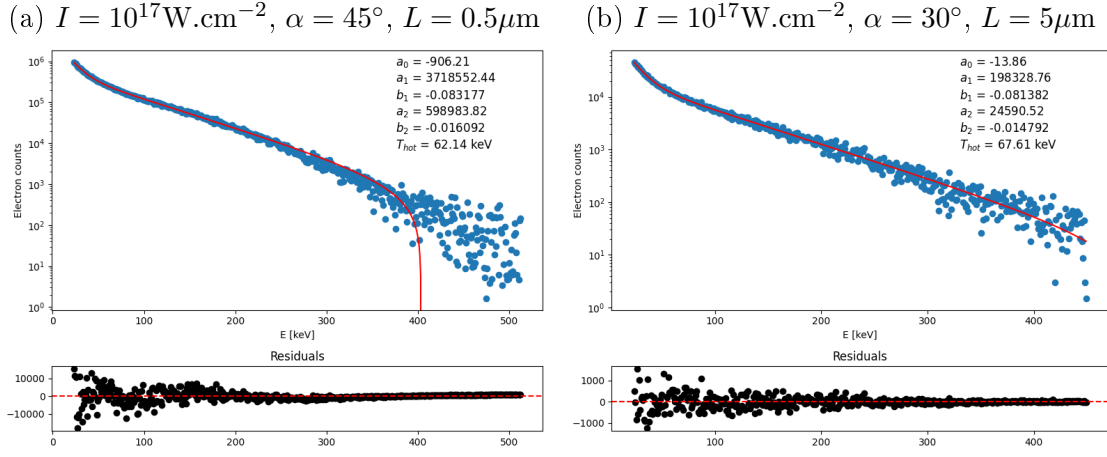
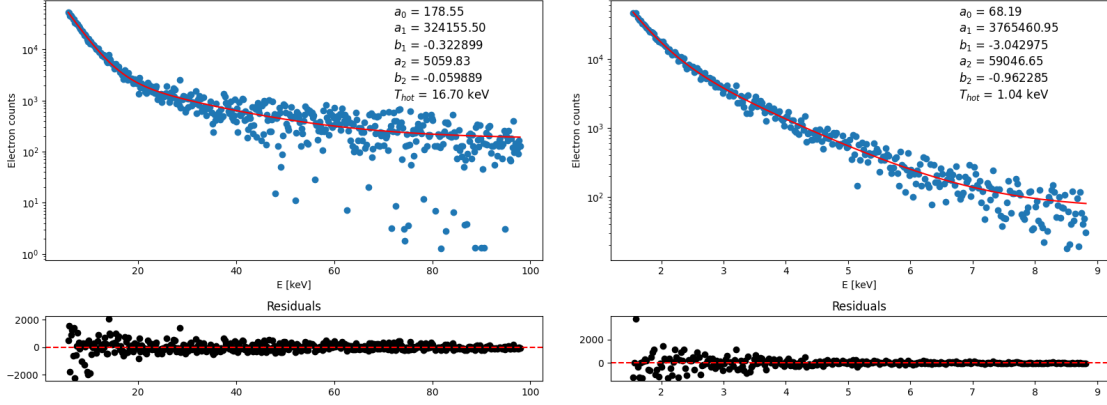
Figure 4.1: Fit of simulation results with parameters  $I = 10^{17} \text{ W.cm}^{-2}$ ,  $\alpha = 50^\circ$ ,  $L = 0.05 \mu\text{m}$ .

This fit shows a strange behavior in the tail of the distribution. The approximation of the constant  $a_0$  from the fit equation 2.5 is fitted below zero which causes the curve to head straight down. On the other hand, the part of the energy spectrum which is interesting to us is fitted quite well and therefore the value of the temperature can be considered reliable.

A few other examples of the fitted histograms can be seen in the figure 4.2. For example, in 4.2e, there is the opposite case

The variance in this graph is much larger for the lower electron energies. That can be seen in the bottom graph in figure 4.1. This is characteristic for all the histograms. In some graphs of the residuals, for example in the bottom part of figure 4.2f, there is a strong indication that the fitted function is not quite right. Results like these were exceptions, but this issue might be addressed in the future if some pattern is found. For the purpose of this thesis, the hot electron temperature is fitted with an acceptable error.

If the fitted curve does not match the data perfectly (at least in the region where we expect the hot electrons), the deviation of the parameter causes a noise in the



(e)  $I = 10^{19} \text{W.cm}^{-2}$ ,  $\alpha = 10^\circ$ ,  $L = 0.1 \mu\text{m}$  (f)  $I = 10^{19} \text{W.cm}^{-2}$ ,  $\alpha = 50^\circ$ ,  $L = 0.1 \mu\text{m}$

Figure 4.2: Examples of exponential fit to some simulation results.

dataset used for training of the neural network. To reduce the noise, one would have to improve either the fitting process or at least the selection of the fitted range. If the results of this thesis prove to be useful, it might be worthwhile to try to find an improvement in this part.

### 4.1.3 The dataset

The dataset has been created based on the results of the exponential-sum fit. For each histogram fitted with sum of two exponential functions as in the examples above. The features are the three input parameters ( $I$ ,  $L$  and  $\alpha$ ) and the target is the temperature  $T_{hot}$ .

The dataset can be plotted, but we have to do it in clever way because 3D plots are usually not very readable. On the  $y$ -axis of the plot we put the angle of incidence  $\alpha$  and on the  $x$ -axis let us put  $L$ . As we simulated the interaction for three different intensities, we can plot three different color graphs. The temperature is shown on a color palette. These plots can be seen in figures 4.3, 4.4 and 4.5 respectively.

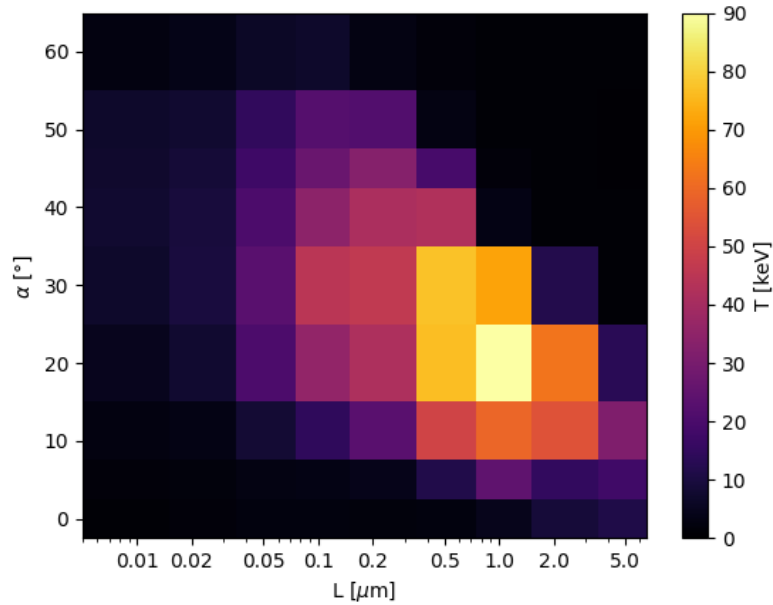


Figure 4.3: A depiction of a part of the dataset - temperatures for the intensity of the laser  $I = 10^{17} \text{W.cm}^{-2}$ .

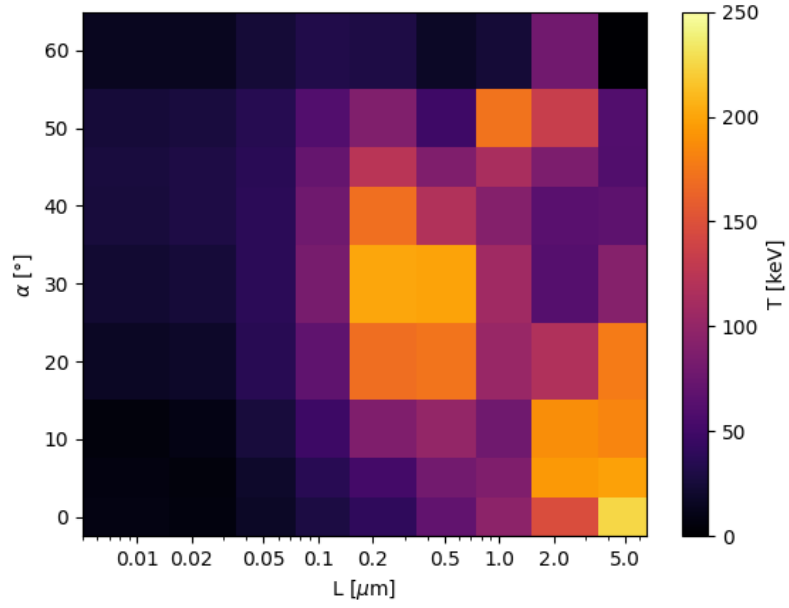


Figure 4.4: A depiction of a part of the dataset - temperatures for the intensity of the laser  $I = 10^{18} \text{W.cm}^{-2}$ .

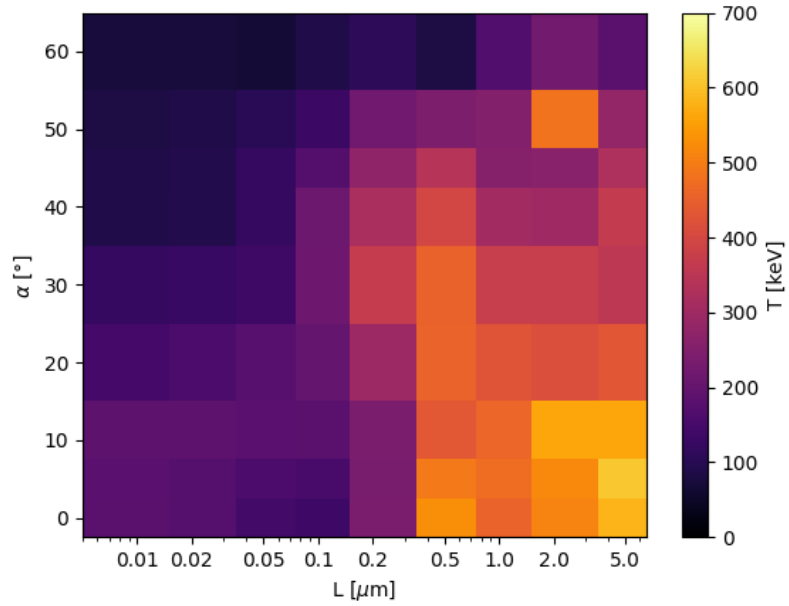


Figure 4.5: A depiction of a part of the dataset - temperatures for the intensity of the laser  $I = 10^{19} \text{W.cm}^{-2}$ .

## 4.2 Tuning the network

We divided the data into three parts. 90% is the training set, 10% of training set is used for validation when choosing the correct model and the rest is set aside for final testing.

The best architecture of the network has to be found by experimentation. We decided to experiment with architectures where all the hidden layers are built of the same amount of neurons. For 8, 16, 32, 64 and 128 neurons in the hidden layers we trained networks with 3, 5, 7, 9, 11 and 13 hidden layers. For each combination we trained 20 networks so that we reduce the bias of the results by the random weight initialization.

To sum up, we trained 600 different neural networks on the same training data and for each of them we calculated the validation loss - average loss for the validation set. The settings of these networks included using 150 epochs and a batch size of 20. We chose the ReLU activation, because it performed much better than others and as the weight initializer we chose *'he\_uniform'* because the Tensorflow documentation claims it is the best choice for ReLU [24].

Now, we can compare the average validation loss of each set of 20 networks which have the same parameters. Results can be seen in figure 4.6.

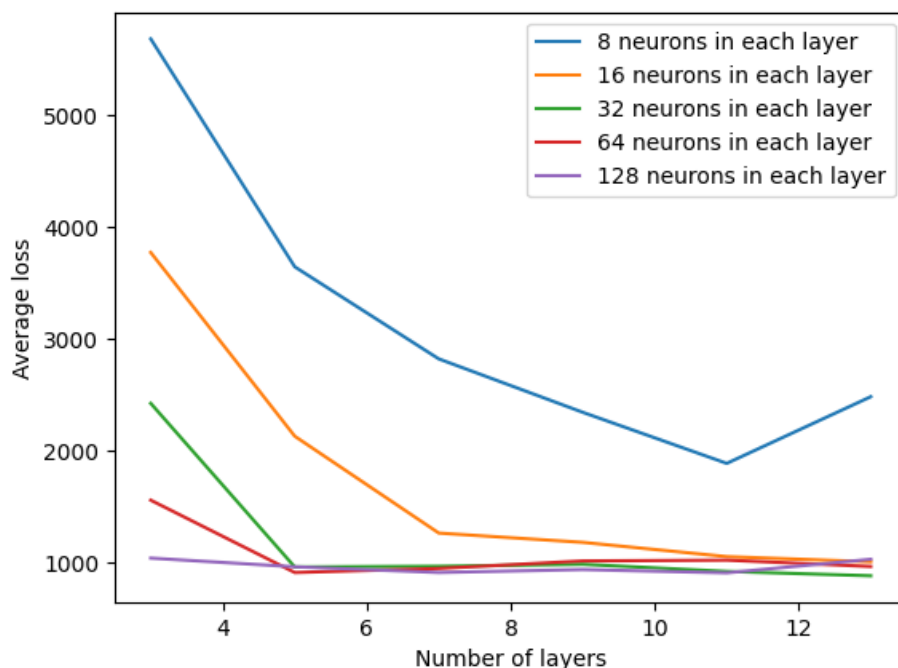


Figure 4.6: A comparison of average validation loss of different neural network architectures.

The data from figure 4.6 are written in table 4.1, where in each cell there is an average validation loss and its standard deviation.

	3 layers	5 layers	7 layers
8 neurons	$5686 \pm 762$	$3647 \pm 823$	$2823 \pm 714$
16 neurons	$3775 \pm 516$	$2132 \pm 425$	$1265 \pm 400$
32 neurons	$2426 \pm 232$	$962 \pm 148$	$968 \pm 166$
64 neurons	$1560 \pm 198$	$912 \pm 168$	$949 \pm 189$
128 neurons	$1041 \pm 118$	$964 \pm 213$	$913 \pm 191$

(a) The average validation loss with standard deviation for 3, 5 and 7 hidden layers.

	9 layers	11 layers	13 layers
8 neurons	$2345 \pm 799$	$1889 \pm 626$	$2485 \pm 989$
16 neurons	$1182 \pm 259$	$1055 \pm 245$	$1009 \pm 285$
32 neurons	$986 \pm 194$	$922 \pm 224$	$883 \pm 229$
64 neurons	$1016 \pm 258$	$1023 \pm 186$	$967 \pm 244$
128 neurons	$939 \pm 195$	$910 \pm 184$	$1031 \pm 447$

(b) The average validation loss with standard deviation for 9, 11 and 13 hidden layers.

Table 4.1: Fine-tuning of the deep neural network architecture

In general, it seems that for low number of layers more neurons in each hidden layer yield better results. However, from a certain point, having more neurons does not contribute to better performance. For 32, 64 and 128 neurons, the average validation loss is not much declining already from 5 hidden layers. The over-parameterization is significant even for the smallest of these networks, as we only have dataset of size 243. As we said, that does not have not be a problem, if we can compensate it by the correct regularization technique. That being said, when it comes to deciding between two different architectures with similar predictive ability, we always want to choose the one with smaller number of parameters as these networks are already well over-parameterized.

Looking at the results, 32 neurons seem to be a good choice and even though the overall best average loss (883) was achieved with 13 layers, we should settle for the maximum of 7 layers as it already contains more than 6 thousand parameters (weights and biases) -  $64 + 1056 \times 6 + 33 = 6433$ . Note that we did not do this comparison to find the absolute best architecture, but rather to get a rough idea where to start the tuning process.

Looking at the smoothness of plotted predictions might be much more valuable indicator of over-fitting as smoothness is much more important to us. Together with regularization and fine-tuning the other parameters it should not be surprising that a much smaller model is chosen as final.

## Fine-tuning the regularization

The next step is to choose the regularization. We said, that the goal is to create a model that generalizes the data well, but also preserves the smooth character of the laser absorption. We do not have an explicit quantifier of the smoothness. However, by *trial and error* we can converge to satisfying model quite quickly.

Starting at 32 neurons and 7 layers we quickly found that this was way too many parameters. A maximum number of layers that did not lead to strong over-fitting even after regularization was 2. An example of such unacceptable over-fit is shown in the figure 4.7.

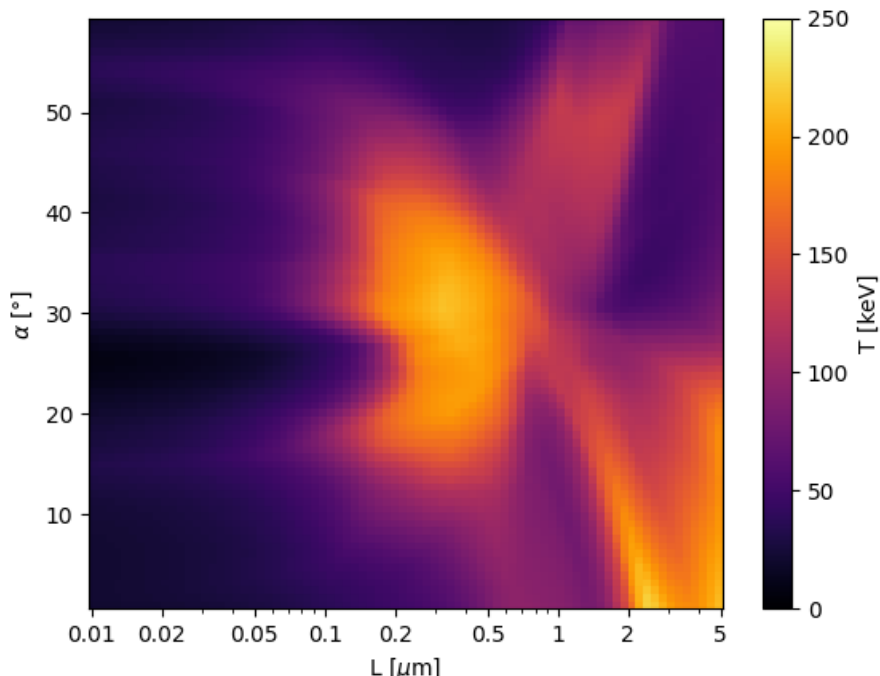


Figure 4.7: An example of over-fitting encountered during fine-tuning despite. It is a prediction for the intensity  $I = 10^{18} \text{W.cm}^{-2}$  of network with 7 layers each with 32 neurons.

This demonstrates that analysis of neural network performance can be more difficult when using a small dataset. Looking only at the validation loss is in this case misleading.

We found, that L1 and L2 regularization is better for this application compared to the *dropout* method. To be more precise, the dropout method needs more epochs to converge to the minimum and since our training set has barely 200 samples, we do not get the benefit that training for more epochs usually brings.

A fundamental issue also rose while choosing the correct  $\lambda_R$  for L1 and L2 regularization. We said, that a bigger  $\lambda_R$  ensures that less neurons will be assigned comparatively bigger weights. After looking at the figure 4.3, we can see an outlier at approximately  $\alpha = 20^\circ$ . The physics theory suggests, that somewhere in that region there actually should be a maximum of laser absorption, but mathematically it is an outlier when compared to rest of the dataset at  $I = 10^{17} \text{W.cm}^{-2}$ . Outliers like this can also be found in both of the other dataset figures 4.4 and 4.5. These values, if they represent a maximum, should not be suppressed by model regularization, but since our dataset is poor, so to speak, the regularization will probably have this effect.

Keeping  $\lambda_R$  small is therefore a necessity. The reduction of the model complexity had to be done also by reducing the number of model parameters, by trying to reduce the number of epochs and by applying dropout with small  $p$ .

### 4.2.1 The final model

These are the parameters of the final model:

- Number of hidden layers: 2
- Number of neurons in each hidden layer: 64
- Optimizer: Adam
- Batch size: 20
- Epochs: 200
- Kernel initializer: 'he\_uniform'
- Activation: ReLU
- Loss function: Mean squared error
- Regularizer: L1 with  $\lambda_R = 0.001$  and one dropout on one layer with  $p = 0.1$
- Test loss = 1042

Predictions of this model on for the same intensities as the model was trained on can be seen in the figures 4.8a, 4.8b and 4.8c.

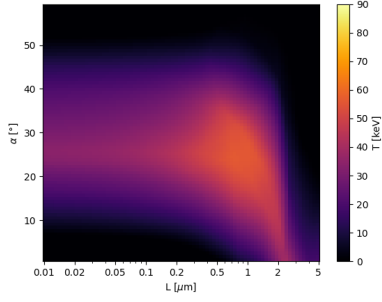
## 4.3 Discussion

As we said, the dataset was produced by running 243 simulations. A tool for automatic execution of the large set of simulations was developed and the post processing tool for automatic analysis of the results as well. The temperatures were found using exponential-sum fit. How difficult this task is has been hinted in section 2.1.

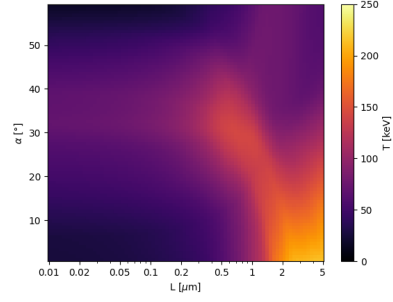
The accuracy of the chosen method can and should be questioned. A majority of the plots of the fit suggest large residuals in the region of small energies. The accuracy of fit in that region is not crucial for the objective of this thesis.

Some noise caused by inaccurate temperature estimation is expected and for good generalizing ability of a neural network it might be even wanted. In such small dataset as ours this is not desirable. Improving the exponential-sum fit could include automatically choosing the correct number of exponential terms. A function reliably evaluating the goodness of the fit could help. Such function could for example penalize larger number of exponential terms so over-fitting would be reduced. In

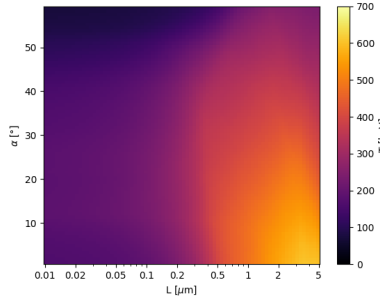




(a) A prediction of the final model - temperatures for the intensity of the laser  $I = 10^{17} \text{W.cm}^{-2}$



(b) A prediction of the final model - temperatures for the intensity of the laser  $I = 10^{18} \text{W.cm}^{-2}$



(c) A prediction of the final model - temperatures for the intensity of the laser  $I = 10^{19} \text{W.cm}^{-2}$ .

Figure 4.8: Model predictions.

our case, this could still be even more complicated because the residuals suggest a non-constant variance.

Let us now analyze the predictions. A comparison of the figures 4.3 and 4.8a tells us something important about the model. The first thing one can notice is the maximum temperature - 90 keV in dataset versus less than 60 keV in the prediction. This is somewhat expected because the maximum temperature in the dataset was one of the outliers which tend to get suppressed in a regression. Another cause contributing to this difference has the roots in the fact that in general higher intensity of laser produces hot electrons with higher temperature. Minimizing the mean square error produced bigger relative error for the small temperatures. Other loss functions taking this into account were tried, but they produced worse results for other reasons.

Now let us analyze how the predictions can be interpreted in the context of the physics. For the parameters of our simulations, the most important absorption processes responsible for generation of hot electrons are Brunel's vacuum heating, resonance absorption, and ponderomotive (or so-called  $\vec{J} \times \vec{B}$ ) heating [12].

The vacuum heating and resonance absorption are more efficient in the sub-relativistic domain with laser intensities  $I \leq 10^{18} \text{W.cm}^{-2}$  and thus we are comparing them with

the results of our model for the lowest intensity  $10^{18}\text{W.cm}^{-2}$ . While both processes contribute to the absorption, the resonance absorption process may produce hotter electrons as the resonantly driven plasma wave may accelerate them over a longer distance if the ideal conditions are fulfilled. Therefore, we expect that this process should dominate our results in the low intensity case.

The resonance absorption is most efficient when the angle of incidence and the scale length of plasma density profile satisfy the optimum condition for the resonance. The optimum angle is  $\alpha = \arcsin[0.68(2\pi L)^{-1/3}]$  where  $L$  is normalized to laser wavelength [12]. It can be seen in figure 4.9 that the results of our PIC simulations agree satisfactorily with this optimum condition.

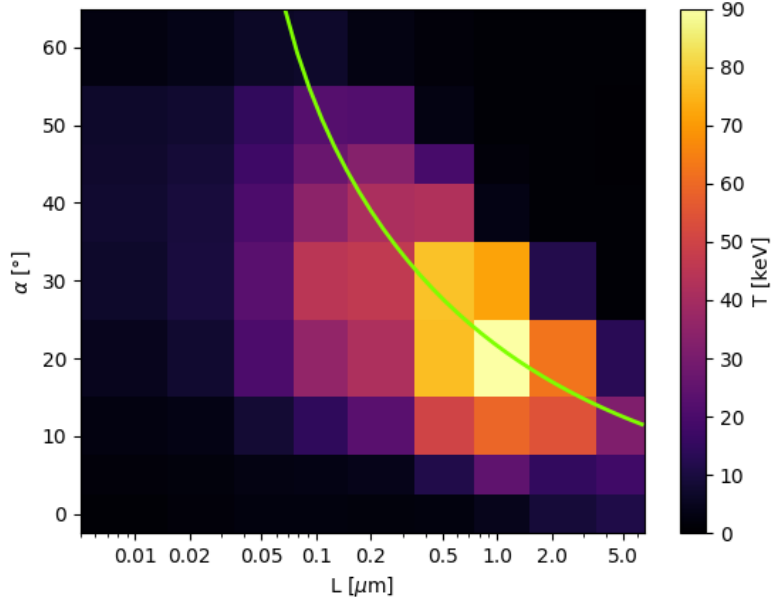


Figure 4.9: A depiction of a part of the dataset - temperatures for the intensity of the laser  $I = 10^{17}\text{W.cm}^{-2}$  with plotted curve given by  $\alpha = \arcsin[0.68(2\pi L)^{-1/3}]$  which characterizes the optimum angle for absorption.

Also, the results of the model presented in figures 4.8a, 4.8b and 4.8c are in relatively good agreement at least in the region, where the resonance is producing the hottest electrons. The vacuum heating should dominate absorption for short density profile scale length  $L$  (much smaller than laser wavelength  $\lambda$ ) and the absorption process in this range of  $L$  does not depend on  $L$  anymore. According to the simple theory, the absorption due to vacuum heating would peak at large angles of incidence close to  $90^\circ$  degrees. This is however not applicable in our simulations as the plasma expands during the process developing a larger scale density profile. In the simulations with the lower laser intensity, the hot electron temperature has a broad maximum between  $30^\circ$  and  $50^\circ$  for low  $L$ . This maximum is also observed in 4.8a however, it is shifted to smaller incidence angles and the hot electron temperature is augmented. This is likely due to influence of the results obtained at higher laser intensities to which the neural network tries to generalize to the lower intensity.

Note that the figure 4.8a has the maximum in the right place but for smaller  $L$  the prediction is larger than in the training data. This is probably caused by the network taking into account the temperatures for higher intensities which indeed are bigger even for smaller  $L$ . Even though the prediction does not copy the dataset precisely, it is not in contradiction with the theoretical curve of optimal resonance absorption.

For higher laser intensities, the resonance absorption is gradually vanishing, and the dominant absorption is shifting toward normal laser incidence as expected for the  $\vec{J} \times \vec{B}$  heating. This can be seen especially for the intensity  $10^{19}\text{W.cm}^{-2}$  in figure 4.8c. The absorption and hot electron temperature are also increased for longer density scale length as noted in section 1.2 due to stochastic heating of electrons in plasma channels, which develop due to hole boring. This behavior is also present in the neural network model.

Finally, let us compare the predictions of our model to some other results for hot electron temperature given in the literature [4], [12]. In [12] the hot electron temperature is shown in Fig. 4c in dependence on the incidence angle for various density scale length and for the laser intensity  $1.37 \times 10^{18}\text{W.cm}^{-2}$ . The temperatures obtained in this paper are significantly higher than the temperatures in our simulations. This might be due to the fact that the laser spot size and the laser pulse duration are both larger, so the laser pulse contains significantly more energy. Another important difference is that the hot electrons analyzed in our simulations are only those propagating into the target while all hot electrons present in the simulation box are included in the results in [12]. The only value given in the paper which is relevant for our comparison is in Fig. 5b,  $T_{hot} = 577\text{keV}$  for  $L$  equal to laser wavelength and incidence angle  $45^\circ$ .

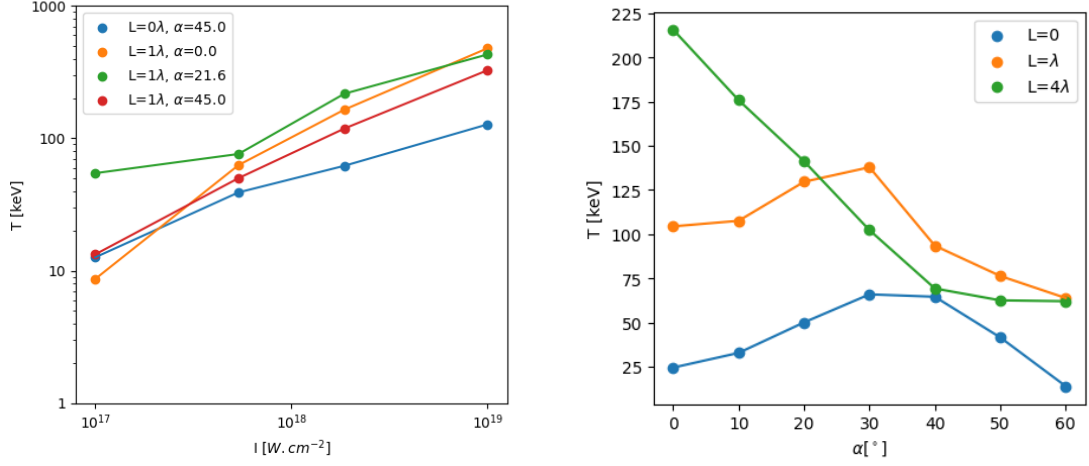
In [4], the hot electron temperatures obtained for the p-polarized laser pulse with intensity  $10^{18}\text{W.cm}^{-2}$  presented in Table I are relevant for our comparison. These are  $T_{hot} = 24\text{keV}$  for  $L = 0$  and  $T_{hot} = 250\text{keV}$  for  $L = 3$  laser wavelength. These results are in a very good agreement with our model. The comparison with the results of our model is shown in the table 4.2.

$I [\text{W.cm}^{-2}]$	$\alpha [^\circ]$	$L [\lambda]$	$T_{hot} [\text{keV}]$ [source]	$T_{hot}$ (our model)
$1.37 \times 10^{18}$	45	1	577 [12]	100
$10^{18}$	0	0	24 [4]	24
$10^{18}$	0	3	250 [4]	212

Table 4.2: A comparison of certain predictions of our model with results from other works.

Another possible comparison can be made when plotting the temperature as a function of  $\alpha$  for certain  $L$  and  $I$ , or as a function of  $I$  for certain  $\alpha$  and  $L$ . In [12] there are such graphs - Fig. 4b and Fig. 4c. Similar plots which contain predictions from our neural network are in figures 4.10a and 4.10b.

We said the the absolute values of the temperatures are expected to be lower, as they are, but there is undeniable similarity in the shape of the plotted curves. The biggest



(a) Plot of the relationship of the temperature predicted by intensity for various parameters. (b) The predicted temperature based on angle for  $L = 0$ ,  $L = \lambda$  and  $L = 4\lambda$  for intensity  $I = 10^{18} \text{W.cm}^{-2}$ .

Figure 4.10: Plots of predictions where temperature is a function of intensity and angle for various  $L$ .

difference is in figure 4.10b where for  $L = 4\lambda$  and small angles the temperature is higher relative to the rest of the plot.

Let us now present the predictions of the network as we increase the intensity from  $10^{17} \text{W.cm}^2$  to  $10^{19} \text{W.cm}^2$ . The predictions can be seen in the plots in figures 4.11a-4.11l. There is a reasonably good monotonous character of the temperature across the various intensities which is what one should expect based on the reasons mentioned above.

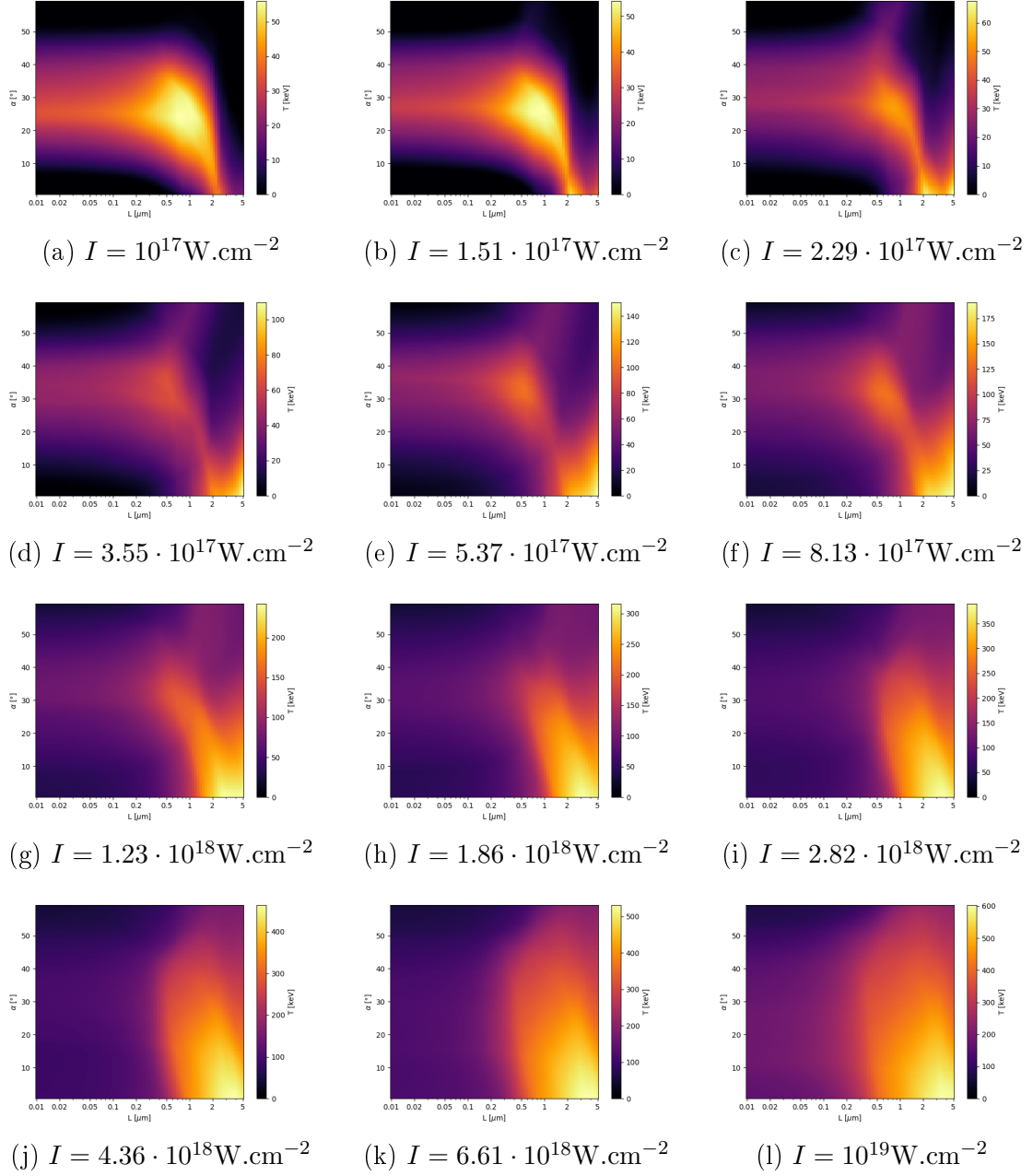


Figure 4.11: The predictions of the network for intensities not included in the dataset.

# Conclusion

Absorption of a high-intensity ultrashort laser pulse in a plasma is studied in this thesis with computer simulations. Using results from PIC simulations we trained a neural network to model the temperatures of the hot electrons based on the input parameters of the simulation. With the dataset of only 243 samples we managed to create a smooth model roughly corresponding to what was presented in other papers motivating this work.

A short summary of a majority of the topics related to this work was presented. Starting with PIC simulations and its principles, we motivated how electron energy distributions can be fitted. Then, the problem of exponential-sum fitting was introduced and besides the method of J. Jacquelin which we used, few other methods were mentioned. The possibility of using different, more robust method might be considered if one wanted to improve the dataset in the future.

In the introduction to neural networks, a few tricks to help with smaller datasets were emphasized. By providing results of a short fine-tuning experiment we demonstrated that the validation loss is not always the most important metric when deciding whether a model is or is not over-parameterized. At least in a regression, it appears to be more important to look directly at the smoothness of the predictions computed for a much denser set of parameters than the training set.

An automated way of PIC results evaluation was presented and implemented. We used an explicit method for exponential-sum fitting which was presented by J. Jacquelin [22]. Its simplicity was described in theory and its robustness was shown by fitting the electron energy spectrum. Namely, it was used to get an approximation of the temperature of the hot electrons by fitting a sum of two exponential terms with a constant. We have also shown that this method has its limits and might not be always reliable if one is interested in other parameters than the coefficients in the exponent.

If the dataset is expanded in the future, it is advised to find a way to confirm the goodness of the fit by a reliable metric. The laser intensities of  $I = 10^{19} \text{W.cm}^{-2}$  produced some spectra that suggested fitting more than two exponential terms would be more eligible. A metric that would be able to confirm this while taking into account the possibility of over-fitting might improve quality of the dataset or at least some of its parts.

In general, the data acquired this way corresponded with results other works as [12]. Simulations with  $I = 10^{17} \text{W.cm}^{-2}$  had clear signs of resonance absorption

happening in agreement with  $\alpha_m = \arcsin \left[ 0.68 (\omega L/c)^{-1/3} \right]$ . For higher intensities  $I = 10^{18} \text{W.cm}^{-2}$  and  $I = 10^{19} \text{W.cm}^{-2}$ , resonance absorption was less significant compared to the other absorption mechanisms, which is in agreement with the conclusion of [12].

The neural network trained to predict temperatures produces predictions with quite large relative error for lower intensity of the laser  $I = 10^{17} \text{W.cm}^{-2}$ . This is blamed on big difference in electron temperatures between simulations with  $I = 10^{17} \text{W.cm}^{-2}$  and  $I = 10^{18} \text{W.cm}^{-2}$  as well as the small size of the dataset. The predictions at larger intensities do not have such significant difference compared to the data in dataset.

Overall, the possibility of creating a mathematical model in a form of a neural network for modeling hot electron temperature was clearly shown. Obtaining a larger dataset than we had is desirable.

# Bibliography

- [1] M. Abadi, P. Barham, and J. Chen, *Tensorflow: A system for large-scale machine learning*, 2015. [Online]. Available: <https://www.tensorflow.org/>.
- [2] F. S. Acton, *Numerical Methods That Work*. The Mathematical Association of America, 1990, ISBN: 0-88385-450-3.
- [3] T. D. Arber, K. Bennett, C. S. Brady, *et al.*, “Contemporary particle-in-cell approach to laser-plasma modelling”, *Plasma Physics and Controlled Fusion*, vol. 57, p. 113 001, 11 2015. DOI: 10.1088/0741-3335/57/11/113001.
- [4] R. Babjak and J. Psikal, “The role of standing wave in the generation of hot electrons by femtosecond laser beams incident on dense ionized target”, *Physics of Plasmas*, vol. 28, p. 023 107, 2 2021. DOI: 10.1063/5.0031555.
- [5] M. Barbarino, “A brief history of nuclear fusion”, *Nature Physics*, vol. 16, pp. 890–893, 9 2020. DOI: 10.1038/s41567-020-0940-7.
- [6] P. L. Bartlett, “The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network”, *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 44, pp. 525–536, 2 1998. DOI: 10.1109/18.661502.
- [7] C. K. Birdsall and A. B. Langdon, *Plasma physics via computer simulation*. McGraw-Hill, 1985, ISBN: 0070053715.
- [8] C. M. Bishop, *Pattern Recognition and Machine Learning*, M. Jordan, J. Kleinberg, and B. Scholkopf, Eds. Springer, 2006, ISBN: 0-387-31073-8.
- [9] J. Brownlee, *What is the difference between a batch and an epoch in a neural network?*, 2018. [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>.
- [10] O. Buneman, “Dissipation of currents in ionized media”, *Phys. Rev.*, vol. 115, pp. 503–517, 3 1959. DOI: 10.1103/PhysRev.115.503.
- [11] M. S. Cha and R. Snoeckx, “Plasma technology—preparing for the electrified future”, *Frontiers in Mechanical Engineering*, vol. 8, 2022. DOI: 10.3389/fmech.2022.903379.
- [12] Y. Q. Cui, W. M. Wang, Z. M. Sheng, Y. T. Li, and J. Zhang, “Laser absorption and hot electron temperature scalings in laser-plasma interactions”, *Plasma Physics and Controlled Fusion*, vol. 55, p. 085 008, 8 2013. DOI: 10.1088/0741-3335/55/8/085008.
- [13] J. Dawson, “One-dimensional plasma model”, *Physics of Fluids*, vol. 5, pp. 445–459, 4 1962. DOI: 10.1063/1.1706638.



- [14] T. Esirkepov, “Exact charge conservation scheme for particle-in-cell simulation with an arbitrary form-factor”, *Computer Physics Communications*, vol. 135, pp. 144–153, 2001. DOI: 10.1016/S0010-4655(00)00228-9.
- [15] A. Ghorbani, A. Abid, and J. Zou, *Interpretation of neural networks is fragile*, Oct. 2017. [Online]. Available: <http://arxiv.org/abs/1710.10547>.
- [16] P. Gibbon, “Introduction to plasma physics”, CERN, 2014, pp. 51–65, ISBN: 9789290834250. DOI: 10.5170/CERN-2016-001.51.
- [17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ISBN: 0262035618. [Online]. Available: <http://www.deeplearningbook.org>.
- [18] S. Gupta, R. Gupta, M. Ojha, and K. P. Singh, “A comparative analysis of various regularization techniques to solve overfitting problem in artificial neural network”, *Communications in Computer and Information Science*, vol. 799, pp. 363–371, 2018. DOI: 10.1007/978-981-10-8527-7\_30.
- [19] J. M. Hokanson, *Numerically stable and statistically efficient algorithms for large scale exponential fitting*, 2013. [Online]. Available: <https://hdl.handle.net/1911/102224>.
- [20] K. Holmström and J. Petersson, “A review of the parameter estimation problem of fitting positive exponential sums to empirical data”, *Applied Mathematics and Computation*, vol. 126, pp. 31–61, 2002. DOI: 10.1016/S0096-3003(00)00138-7.
- [21] S. Ingrassia and I. Morlini, “Neural network modeling for small datasets”, *Technometrics*, vol. 47, pp. 297–311, 3 2005. DOI: 10.1198/004017005000000058.
- [22] J. Jacquelin, *Regressions et equations integrales*, 2014. [Online]. Available: <https://www.scribd.com/doc/14674814/Regressions-et-equations-integrales>.
- [23] A. D. Jagtap and G. E. Karniadakis, *How important are activation functions in regression and classification? a survey, performance comparison, and future directions*, 2022. [Online]. Available: <http://arxiv.org/abs/2209.02681>.
- [24] D. Kerabatsos, *Tensorflow documentation*. [Online]. Available: [https://www.tensorflow.org/api\\_docs/python/tf](https://www.tensorflow.org/api_docs/python/tf).
- [25] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning”, *Nature*, vol. 521, pp. 436–444, 7553 2015. DOI: 10.1038/nature14539.
- [26] D. Masters and C. Luschi, *Revisiting small batch training for deep neural networks*, Apr. 2018. [Online]. Available: <http://arxiv.org/abs/1804.07612>.
- [27] P. J. Mohr, D. B. Newell, and B. N. Taylor, “Codata recommended values of the fundamental physical constants: 2014”, *Reviews of Modern Physics*, vol. 88, 3 2016. DOI: 10.1103/RevModPhys.88.035009.
- [28] D. Montgomery, E. Peck, and G. Vinning, *Introduction to Linear Regression Analysis*, 6th ed. John Wiley & Sons, 2021, ISBN: 1119578728.
- [29] D. Potts and M. Tasche, “Parameter estimation for exponential sums by approximate prony method”, *Signal Processing*, vol. 90, pp. 1631–1642, 5 2010. DOI: 10.1016/j.sigpro.2009.11.012.

- [30] A. Pouliakis, E. Karakitsou, N. Margari, *et al.*, “Artificial neural networks as decision support tools in cytopathology: Past, present, and future”, *Biomedical Engineering and Computational Biology*, vol. 7, pp. 1–18, 2016. DOI: 10.4137/becb.s31601.
- [31] G. R. B. Prony, “Essai experimental et analytique sur les lois de la dilatabilite de fluides elastiques et sur celles de la force expansion de la vapeur de l’alcool, a differentes temperatures.”, *Journal de l’Ecole Polytechnique (Paris)*, vol. 1, pp. 24–76, 1795.
- [32] Y. Sentoku and A. J. Kemp, “Numerical methods for particle simulations at extreme densities and temperatures: Weighted particles, relativistic collisions and reduced currents”, *Journal of Computational Physics*, vol. 227, pp. 6846–6861, 14 2008. DOI: 10.1016/j.jcp.2008.03.043.
- [33] T. Shaikhina and N. Khovanova, “Handling limited datasets with neural networks in medical applications: A small-data approach”, *Artificial Intelligence in Medicine*, vol. 75, pp. 51–63, 2017. DOI: 10.1016/j.artmed.2016.12.003.
- [34] Z. M. Sheng, S. M. Weng, L. L. Yu, *et al.*, “Absorption of ultrashort intense lasers in laser-solid interactions”, *Chinese Physics B*, vol. 24, p. 015 201, 1 2015. DOI: 10.1088/1674-1056/24/1/015201.
- [35] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting”, *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 1 2014.
- [36] D. Tskhakaya, K. Matyash, R. Schneider, and F. Taccogna, “The particle-in-cell method”, *Contributions to Plasma Physics*, vol. 47, pp. 563–594, 8-9 2007. DOI: 10.1002/ctpp.200710072.
- [37] J. Wang, Z. Chen, Y. Wang, *et al.*, “Three-dimensional parallel unipic-3d code for simulations of high-power microwave devices”, *Physics of Plasmas*, vol. 17, p. 073 107, 7 2010. DOI: 10.1063/1.3454766.
- [38] B. Widrow, D. E. Rumelhart, and M. A. Lehr, “Neural networks: Applications in industry, business and science”, *Communications of the ACM*, vol. 37, pp. 93–105, 3 1994. DOI: 10.1145/175247.175257.
- [39] W. J. Wiscombe and J. W. Evans, “Exponential-sum fitting of radiative transmission functions”, *JOURNAL OF COMPUTATIONAL PHYSICS*, vol. 24, pp. 416–444, 4 1977. DOI: 10.1016/0021-9991(77)90031-6.
- [40] K. S. Yee, “Numerical solution of initial boundary value problems involving maxwell’s equations in isotropic media”, *IEEE Transactions on Antennas and Propagation*, vol. 14, pp. 302–307, 3 1966. DOI: 10.1109/TAP.1966.1138693.

# List of Figures

1.1	An illustration of a Yee grid [37] . . . . .	4
2.1	An example of electron energy distribution with intensity of laser $I = 10^{19}\text{W.cm}^{-2}$ , characteristic scale of the exponential preplasma profile $L = 0.1\mu\text{m}$ and angle of incidence with respect to target normal direction $\alpha = 10^\circ$ . . . . .	10
3.1	An illustration of a feed-forward neural network with multiple hidden layers [30]. . . . .	17
3.2	An overview of graphs of common activation functions [23]. . . . .	19
3.3	An illustration of the dropout technique [35]. . . . .	21
4.1	Fit of simulation results with parameters $I = 10^{17}\text{W.cm}^{-2}$ , $\alpha = 50^\circ$ , $L = 0.05\mu\text{m}$ . . . . .	26
4.2	Examples of exponential fit to some simulation results. . . . .	27
4.3	A depiction of a part of the dataset - temperatures for the intensity of the laser $I = 10^{17}\text{W.cm}^{-2}$ . . . . .	28
4.4	A depiction of a part of the dataset - temperatures for the intensity of the laser $I = 10^{18}\text{W.cm}^{-2}$ . . . . .	29
4.5	A depiction of a part of the dataset - temperatures for the intensity of the laser $I = 10^{19}\text{W.cm}^{-2}$ . . . . .	29
4.6	A comparison of average validation loss of different neural network architectures. . . . .	30
4.7	An example of over-fitting encountered during fine-tuning despite. It is a prediction for the intensity $I = 10^{18}\text{W.cm}^{-2}$ of network with 7 layers each with 32 neurons. . . . .	32
4.8	Model predictions. . . . .	34
4.9	A depiction of a part of the dataset - temperatures for the intensity of the laser $I = 10^{17}\text{W.cm}^{-2}$ with plotted curve given by $\alpha = \arcsin[0.68(2\pi L)^{-1/3}]$ which characterizes the optimum angle for absorption. . . . .	35

4.10	Plots of predictions where temperature is a function of intensity and angle for various $L$ . . . . .	37
4.11	The predictions of the network for intensities not included in the dataset. . . . .	38

# Appendix A

## EPOCH input file - input.deck

```
begin:constant
konstanta1 = 1e19
konstanta2 = 2.0
konstanta3 = 60

las_lambda    = 0.8 * micron
l1            = las_lambda
las_intensity_focus_w_cm2 = konstanta1
a0            = l1 / micron * sqrt(las_intensity_focus_w_cm2/1.37e18)
gamma0        = sqrt(1.0+a0^2)
pulse_length  = 30.0 * femto
spot_size     = 3.2 * micron

cpwl          = 30
simtime       = 800.0 * femto
theta         = konstanta3 / 180.0 * pi
ct            = cos(theta)
st            = sin(theta)
n_el_over_nc  = 3.0*gamma0

min_density   = 0.01
preplasma_length = konstanta2 * micron
foil_thickness = 4.0 * micron
profile_thickness = - loge(min_density) * preplasma_length
boundary_thickness = 1.0 * micron
x1 = 60 * micron - foil_thickness -
    profile_thickness - boundary_thickness
x2 = x1 + profile_thickness
x3 = x2 + foil_thickness
xfocus       = x2
yc           = - xfocus * tan(theta) * 0.5
sigma_to_fwhm = 2.0 * sqrt(log(2.0))
w0           = spot_size / sigma_to_fwhm
```

```

zR          = pi * w0^2 / ll
lfocus      = xfocus / ct
lfzr        = lfocus / zR
wl          = w0 * sqrt(1.0+lfzr^2)
intensity_fac2d = 1.0 / sqrt(1.0+lfzr^2) * ct
las_omega   = 2.0 * pi* 299792458.0 / ll
las_time     = 2.0 * pi / las_omega
wt          = pulse_length / sigma_to_fwhm
n_crit      = critical(las_omega)
n_el        = n_el_over_nc * n_crit
y_full      = 54.0 * micron
end:constant

```

```

begin:control
x_min = 0.0 * micron
x_max = 60.0 * micron
y_min = -55.0 * micron
y_max = 55.0 * micron
nx = (x_max-x_min) / ll * cpwl
ny = (y_max-y_min) / ll * cpwl
t_end = simtime
particle_tstart = x1 / ct / 299792458.0
dt_multiplier = 0.97
dlb_threshold = 0.5
use_random_seed = T
stdout_frequency = 10
smooth_currents = T
field_order = 2
end:control

```

```

begin:boundaries
cpml_thickness = 6
cpml_kappa_max = 20
cpml_a_max = 0.15
cpml_sigma_max = 0.7
bc_x_min = cpml_laser
bc_x_max_field = cpml_outflow
bc_y_min_field = periodic
bc_y_max_field = periodic
bc_x_min_particle = heat_bath
bc_x_max_particle = heat_bath
bc_y_min_particle = heat_bath
bc_y_max_particle = heat_bath
end:boundaries

```

```

begin:laser
boundary = x_min

```

```

intensity_w_cm2 = las_intensity_focus_w_cm2 * intensity_fac2d
lambda = ll
t_profile = gauss(time, pulse_length, wt)
t_end = 3.0 * pulse_length
phase = -2.0*pi*(y-yc)*st/ll+((pi/ll)*(((y-yc)*ct)^2))/
        (lfocus*(1.0+(1.0/lfzr)^2))-atan((y-yc)*st/zR+lfzr)
profile = gauss((y-yc)*ct,0.0,wl)
end:laser

begin:species
name = electron
charge = -1.0
mass = 1.0
dump = T
npart_per_cell = 30
density = if((x gt x1) and (x lt x2) and (abs(y) lt y_full),
            n_el*exp((x-x2)/preplasma_length), 0.0)
density = if((x gt x2) and (x lt x3) and (abs(y) lt y_full), n_el,
            density(electron))
temp_ev = 100.0
identify:electron
end:species

begin:species
name = proton
charge = 1.0
mass = 1836.0
dump = T
npart_per_cell = 30
density = density(electron)
temp_ev = 100.0
end:species

begin:probe
name = electron_back_probe
point = (x3-2.0*micron, 0.0)
normal = (1.0, 0.0)
ek_min = 1.602e-16
ek_max = -1.0
include_species:electron
dumpmask = always
end:probe

#begin:output
# dt_snapshot = 10.0*femto
# name = fields
# file_prefix = f

```

```

# restartable = F
# grid = always + single
# ex = always + single
# ey = always + single
# bz = always + single
#end:output

#begin:output
# dt_snapshot = 10.0*femto
# name = plasma
# file_prefix = n
# restartable = F
# grid = always
# number_density = always + single
# poynt_flux = always + single
# absorption = always
# total_energy_sum = always
#end:output

begin:output
dt_snapshot = 700.0*femto
name = prob
file_prefix = ppr
restartable = F
particle_probes = always
end:output

```