UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Bachelor Thesis

# Physics-informed neural networks for 1D plasma edge transport in fusion devices

Matti Fensch

Reviewers:
Prof. Dr. Ralf Schneider
Prof. Dr. Peter Manz

Institute of Physics
University of Greifswald
August 7, 2023

# Contents

# 1 Introduction

Plasma is often referred to as the fourth physical state of matter. Unlike the common states solid, fluid and gaseous, at least some atoms are ionized in a plasma. In nature, some examples are lightnings, sparks and northern lights [1]. The Leibniz Institute for Plasma Science and Technology (INP) in Greifswald does research on low-temperature plasmas and their applications in renewable energies, bio economy, plasma chemistry, process technology, health and hygiene [2].
Physicists also make use of high-temperature plasmas to trigger nuclear fusion reactions. A popular example is the research at the Max Planck Institute for Plasma Physics (IPP) in Greifswald with its Wendelstein 7-X experiment [3]. Since electric and magnetic fields are used to control the plasma particles in magnetic fusion experiments, the study of their behavior in such environments is of particular interest.
In this thesis, the transport in fully ionized plasmas containing electrons ($q_{\mathrm{c,e}} = -e$) and one type of ions ($q_{\mathrm{c,i}} = Ze$) is considered, specifically in the so-called scrape-off layer of fusion plasmas, which is determined by the contact with walls. This plasma is a bounded plasma in between walls. It is characterized by strong interaction with neutrals and by collisions with neutrals and between the plasma particles due to its high density. The collisions define mean-free paths for the plasma particles which are shorter than the system length. This allows a fluid description of the plasma transport, because the distribution gets Maxwellian. The emerging coupled, non-linear transport equations of fusion edge plasmas are solved numerically. Here, physics-informed neural networks (PINNs) come into play. PINNs are neural networks trained by differential equations to approximate their solution using automatic differentiation.
This thesis uses PINNs for the solution of the partial differential transport equations along the field lines in the edge of fusion plasmas to evaluate if they can be used as a numerical alternative for such systems. The reduction to the transport processes along field lines is a first approximation, because the transport in this direction is much stronger than in the other directions due to the strong magnetization of the plasma particles. This forces them to follow the field lines gyrating around them. In chapter 2 the general plasma transport equations are derived and reduced to the 1D formulation along magnetic field lines. Additionally, neural networks and the Python package PyTorch are introduced and applied to introduce the numerical method of PINNs. These PINNs are tested in chapter 3 in several simplified cases of the transport equations for which analytical solutions exist. Then, the results for a more complex version of the coupled transport equations are presented, where no analytical solutions exist. Finally, the thesis and in particular the difficulties and benefits of PINNs in the given field are summarized.

# 2 Basics

In this section, the basic description of plasma transport in the collision-dominated edge region of fusion plasmas and the basic concept of PINNs are introduced.

## 2.1 Derivation of the transport equations

The derivation of the transport equations in a fully ionized plasma follows Braginskii [4]. The starting point of describing the plasma's behavior is the kinetic Boltzmann equation

$$\frac{\partial f_\mathrm{a}}{\partial t} + \frac{\partial}{\partial x_\beta}(v_\beta f_\mathrm{a}) + \frac{\partial}{\partial v_\beta}\left(\frac{F_{\mathrm{a}\beta}}{m_\mathrm{a}} f_\mathrm{a}\right) = C_\mathrm{a} \tag{1}$$

with distribution function $f_\mathrm{a}(t, \mathbf{r}, \mathbf{v})$ at time $t$, position $\mathbf{r}$ and velocity $\mathbf{v}$, force $\mathbf{F}_\mathrm{a}$ on a particle with mass $m_\mathrm{a}$ and collision term $C_\mathrm{a}$. The index $a$ indicates the particle type, e.g., electron or ion. Terms including $\beta$ indices are written in Einstein notation. Therefore, equation (1) is equivalent to

$$\frac{\partial f_\mathrm{a}}{\partial t} + \nabla_\mathbf{r}(\mathbf{v} f_\mathrm{a}) + \nabla_\mathbf{v}\left(\frac{\mathbf{F}_\mathrm{a}}{m_\mathrm{a}} f_\mathrm{a}\right) = C_\mathrm{a} \ .$$

The distribution function $f_\mathrm{a}(t, \mathbf{r}, \mathbf{v})$ is normalized in a way so that[1]

$$n_\mathrm{a}(t, \mathbf{r}) = \int f_\mathrm{a} d^3v \tag{2}$$

yields the density of particles $n_\mathrm{a}$ of type $a$ in $\mathbf{r}$ at $t$. Integrating $n_\mathrm{a}$ over some volume $V$ would give the number of particles in $V$. To determine an expectation value $\langle \mathbf{w} \rangle = \mathbf{W}_\mathrm{a}(t, \mathbf{r})$ of some physical quantity $\mathbf{w}_\mathrm{a}$ in terms of the distribution function it is necessary to calculate

$$\langle \mathbf{w} \rangle_\mathrm{a} = \mathbf{W}_\mathrm{a}(t, \mathbf{r}) = \frac{\int \mathbf{w} f_\mathrm{a}(t, \mathbf{r}, \mathbf{v}) d^3v}{\int f_\mathrm{a}(t, \mathbf{r}, \mathbf{v}) d^3v} = \frac{1}{n_\mathrm{a}} \int \mathbf{w} f_\mathrm{a}(t, \mathbf{r}, \mathbf{v}) d^3v \ .$$

Since the plasma behavior is studied in fusion plasmas, an electric field $\mathbf{E}$ and magnetic field $\mathbf{B}$ has to be taken into account. The Lorentz force acting on a particle is

$$\mathbf{F}_\mathrm{a} = q_\mathrm{c,a}\left(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}\right) \tag{3}$$

where $q_\mathrm{c,a}$ denotes the charge of particles $a$ and $c$ is the speed of light.[2]
The term $C_\mathrm{a}$ in equation (1) carries the information about collisions between particles. Since $aa$ and $ab$ collisions are possible in the plasma considered, the term becomes

$$C_\mathrm{a} = \sum_b C_\mathrm{ab}(f_\mathrm{a}, f_\mathrm{b}) \ . \tag{4}$$

Elastic collisions produce sources and sinks. They satisfy particle, momentum and energy conservation. As stated in [5, p. 3], if $\boldsymbol{\psi}(\mathbf{v})$ is an invariant with regard to an $ab$ collision, it has to fulfill,

$$\int \boldsymbol{\psi}(\mathbf{v}) C_\mathrm{ab} d^3v = 0 \ .$$

[1]The integral is meant to be over the entire phase space. If not explicitly stated, this applies to the rest of this thesis.
[2]Here, CGS units are used. If not explicitly stated, this applies to the rest of this thesis.

Therefore,

$$\int C_{\mathrm{ab}} d^3v = 0 \tag{5}$$

$$\int m_{\mathrm{a}} \mathbf{v} C_{\mathrm{aa}} d^3v = 0 \tag{6}$$

$$\int \frac{m_{\mathrm{a}} v^2}{2} C_{\mathrm{aa}} d^3v = 0 \ . \tag{7}$$

Equation (5) holds for any general *ab* collision because the number of particles is conserved even though multiple particle types are involved. That is not the case for conservation of momentum (6) and energy (7) because those quantities can be transferred between different particle types. Hence, it is only true for *aa* collisions.

### 2.1.1 Continuity equation

To make use of (5), it is obvious to integrate both sides of equation (1) over the entire velocity space.

$$\int \frac{\partial f_{\mathrm{a}}}{\partial t} d^3v + \int \frac{\partial}{\partial x_\beta}(v_\beta f_{\mathrm{a}}) d^3v + \int \nabla_{\mathbf{v}} \left( \frac{\mathbf{F}_{\mathrm{a}}}{m_{\mathrm{a}}} f_{\mathrm{a}} \right) d^3v = \int C_{\mathrm{a}} d^3v$$

Now, the collision term can be evaluated by using equation (4) and (5) and by changing the order of summation and integration. Additionally, the order of differentiation and integration in the first two terms can be changed.

$$\frac{\partial}{\partial t} \int f_{\mathrm{a}} d^3v + \frac{\partial}{\partial x_\beta} \int v_\beta f_{\mathrm{a}} d^3v + \int \nabla_{\mathbf{v}} \left( \frac{\mathbf{F}_{\mathrm{a}}}{m_{\mathrm{a}}} f_{\mathrm{a}} \right) d^3v = \sum_b \int C_{\mathrm{ab}} d^3v = 0$$

Because of (2), the first term becomes the time derivative of $n_{\mathrm{a}}$. The second term can be interpreted as divergence of the expectation value of velocity $\mathbf{V}$ times $n_{\mathrm{a}}$. The third term vanishes by using Gauss's theorem (see equation (48) in appendix A.1) and applying that the distribution function decreases rapidly for $v \to \infty$. Additionally, the $a$ can be omitted for better readability.
Overall, this results in the continuity equation

$$\frac{\partial n}{\partial t} + \nabla(n\mathbf{V}) = 0 \ . \tag{8}$$

### 2.1.2 Momentum equation

Equation (1) can be multiplied by $m_{\mathrm{a}}\mathbf{v}$ and integrated again over the entire velocity space. Already omitting the $a$ and writing the $\alpha$th component of the equation gives[1]

$$\int \frac{\partial}{\partial t}(f m v_\alpha) d^3v + \int \frac{\partial}{\partial x_\beta}(m v_\alpha v_\beta f) d^3v + \int \nabla_{\mathbf{v}} (\mathbf{F} f)\, v_\alpha d^3v = \int m v_\alpha C d^3v \ .$$

It must be noted that $C$ only covers collisions between different particles because the $C_{\mathrm{aa}}$ integral vanishes due to (6).
After changing the order of derivative and integration like before, the first term reduces to $\frac{\partial}{\partial t}(mnV_\alpha)$ and the second one to $\frac{\partial}{\partial x_\beta}(mn\langle v_\alpha v_\beta \rangle)$. The third term can be calculated using

[1]Inserting the velocity into the time and spatial derivatives is allowed due to the independence of t, $\mathbf{r}$ and $\mathbf{v}$ among each other.

higher dimensional integration by parts (see equation (49) in appendix A.1) and applying that $f$ decreases rapidly. Hence,

$$\begin{aligned}\int \nabla_{\mathbf{v}} (\mathbf{F} f)\, v_\alpha d^3v &= -\int \mathbf{F} f \nabla_{\mathbf{v}}(v_\alpha) d^3v = -\int \mathbf{F} f \hat{e}_\alpha d^3v \\ &= -\int \mathbf{F}_\alpha f d^3v = -qn\left(\mathbf{E} + \frac{\mathbf{V}}{c} \times \mathbf{B}\right)_\alpha\end{aligned}$$

is valid for a force like (3). This results in

$$\frac{\partial}{\partial t}(mnV_\alpha) + \frac{\partial}{\partial x_\beta}(mn\langle v_\alpha v_\beta\rangle) - q_c n\left(\mathbf{E} + \frac{\mathbf{V}}{c} \times \mathbf{B}\right)_\alpha = \int m v_\alpha C d^3v \ . \tag{9}$$

By introducing the following quantities and abbreviations, this can be converted into a more useful form.

$$\begin{aligned} T(t,\mathbf{r}) &\coloneqq \frac{m}{3}\left\langle(\mathbf{v}-\mathbf{V})^2\right\rangle \eqqcolon \frac{m}{3}\langle\boldsymbol{\eta}^2\rangle \\ \pi_{\alpha\beta} &\coloneqq nm\left\langle \eta_\alpha\eta_\beta - \frac{\boldsymbol{\eta}^2}{2}\delta_{\alpha\beta}\right\rangle \\ \mathbf{R} &\coloneqq \int m\boldsymbol{\eta} C d^3v \end{aligned} \tag{10}$$

Here $\pi_{\alpha\beta}$ is the stress tensor and $\mathbf{R}$ is the change in momentum due to collisions. The temperature $T$ is defined as an energy.[1] The exact definition is justified by the ansatz $\frac{1}{2}m\langle\boldsymbol{\eta}^2\rangle = \frac{3}{2}T$, where $\boldsymbol{\eta}$ is the random deviation between velocity $\mathbf{v}$ and its expectation value $\mathbf{V}$. Thus, $\langle\boldsymbol{\eta}\rangle = \langle\mathbf{v}-\mathbf{V}\rangle = \langle\mathbf{v}\rangle - \langle\mathbf{V}\rangle = \mathbf{V}-\mathbf{V} = 0$ and $\langle v_\alpha v_\beta\rangle$ in (9) can also be written as

$$\begin{aligned}\langle v_\alpha v_\beta\rangle &= \langle(V_\alpha+\eta_\alpha)(V_\beta+\eta_\beta)\rangle = \langle V_\alpha V_\beta + V_\alpha\eta_\beta + V_\beta\eta_\alpha + \eta_\alpha\eta_\beta\rangle \\ &= V_\alpha V_\beta + V_\alpha\langle\eta_\beta\rangle + V_\beta\langle\eta_\alpha\rangle + \langle\eta_\alpha\eta_\beta\rangle \\ &= V_\alpha V_\beta + \langle\eta_\alpha\eta_\beta\rangle \ .\end{aligned} \tag{11}$$

and the integral in (9) as

$$\int m v_\alpha C d^3v = \int mV_\alpha C d^3v + \int m\eta_\alpha C d^3v = mV_\alpha\int C d^3v + R_\alpha = R_\alpha$$

because of the already used conservation of momentum (5). Using these, (9) becomes

$$\frac{\partial}{\partial t}(mnV_\alpha) + \frac{\partial}{\partial x_\beta}(mnV_\alpha V_\beta + nT\delta_{\alpha\beta} + \pi_{\alpha\beta}) = q_c n\left(\mathbf{E} + \frac{\mathbf{V}}{c}\times\mathbf{B}\right)_\alpha + R_\alpha \ . \tag{12}$$

This equation is already a momentum equation. Using the material derivative

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + V_\beta\frac{\partial}{\partial x_\beta} = \frac{\partial}{\partial t} + (\mathbf{V}\cdot\nabla_{\mathbf{r}})$$

and rewriting the partial time derivative using the continuity equation (8), it can also be expressed as

$$mn\frac{DV_\alpha}{Dt} = -\frac{\partial nT}{\partial x_\alpha} - \frac{\partial \pi_{\alpha\beta}}{\partial x_\beta} + q_c n\left(\mathbf{E} + \frac{\mathbf{V}}{c}\times\mathbf{B}\right)_\alpha + R_\alpha \ . \tag{13}$$

[1] In principle, the energy $k_BT$ is replaced by the symbol $T$.

### 2.1.3 Energy equation

Multiplying equation (1) by $m_\mathrm{a}v^2/2$ and integrating yields

$$\int \frac{\partial f_\mathrm{a}}{\partial t}\frac{m_\mathrm{a}v^2}{2}d^3v + \int \frac{\partial}{\partial x_\beta}(v_\beta f_\mathrm{a})\frac{m_\mathrm{a}v^2}{2}d^3v + \int \nabla_\mathbf{v}\left(\frac{\mathbf{F}_\mathrm{a}}{m_\mathrm{a}}f_\mathrm{a}\right)\frac{m_\mathrm{a}v^2}{2}d^3v = \int C_\mathrm{a}\frac{m_\mathrm{a}v^2}{2}d^3v\ .$$

As before, the first two terms can be transformed and then interpreted as expectation values. Additionally, the $a$ is dropped.

$$\frac{\partial}{\partial t}\left(\frac{mn}{2}\langle v^2\rangle\right) + \frac{\partial}{\partial x_\beta}\left(\frac{mn}{2}\langle v_\beta v^2\rangle\right) + \int \nabla_\mathbf{v}\left(\frac{\mathbf{F}}{m}f\right)\frac{mv^2}{2}d^3v = \int \frac{mv^2}{2}Cd^3v \tag{14}$$

Again, higher dimensional integration by parts (see equation (49) in appendix A.1) and equation (3) turns the third term into

$$\begin{aligned}\int \nabla_\mathbf{v}\left(\frac{\mathbf{F}}{m}f\right)\frac{mv^2}{2}d^3v &= -\int \mathbf{F}\nabla_\mathbf{v}\left(\frac{v^2}{2}\right)fd^3v = -\int \mathbf{F}\mathbf{v}fd^3v\\ &= -q_\mathrm{c}\int \mathbf{E}\mathbf{v}fd^3v = -q_\mathrm{c}n\mathbf{E}\mathbf{V},\end{aligned}$$

because $f$ decreases rapidly and $(\mathbf{v}\times\mathbf{B})\mathbf{v} = 0$. Hence, (14) becomes

$$\frac{\partial}{\partial t}\left(\frac{mn}{2}\langle v^2\rangle\right) + \nabla_\mathbf{r}\left(\frac{mn}{2}\langle \mathbf{v}v^2\rangle\right) - q_\mathrm{c}n\mathbf{E}\mathbf{V} = \int \frac{mv^2}{2}Cd^3v\ . \tag{15}$$

Now, $\mathbf{v} = \mathbf{V} + \boldsymbol{\eta}$ can be used again to rewrite the first two and the last term. Reusing (11) yields

$$\left\langle v^2\right\rangle = V^2 + \left\langle \eta^2\right\rangle\ . \tag{16}$$

The expectation value of the second term is calculated similarly.

$$\begin{aligned}\left\langle \frac{v^2}{2}v_\beta\right\rangle &= \left\langle \frac{1}{2}\left(\mathbf{V}+\boldsymbol{\eta}\right)^2\left(V_\beta+\eta_\beta\right)\right\rangle\\ &= \left\langle \frac{1}{2}\left(V^2V_\beta + 2\mathbf{V}\boldsymbol{\eta}V_\beta + \eta^2V_\beta + V^2\eta_\beta + 2\mathbf{V}\boldsymbol{\eta}\eta_\beta + \eta^2\eta_\beta\right)\right\rangle\\ &= \frac{1}{2}\left(V^2V_\beta + \langle\eta^2\rangle V_\beta + 2\mathbf{V}\langle\boldsymbol{\eta}\eta_\beta\rangle + \langle\eta^2\eta_\beta\rangle\right)\\ &= \left(\frac{1}{2}V^2 + \frac{3}{2}\frac{T}{m}\right)V_\beta + V_\alpha\langle\eta_\alpha\eta_\beta\rangle + \frac{q_\beta}{mn}\end{aligned} \tag{17}$$

Here, the heat flux density $\mathbf{q} := \left\langle n\frac{m\eta^2}{2}\boldsymbol{\eta}\right\rangle$ and the temperature from (10) are used. By considering

$$\begin{aligned}\pi_{\alpha\beta}V_\alpha = nm\left\langle \eta_\alpha\eta_\beta - \frac{\eta^2}{3}\delta_{\alpha\beta}\right\rangle V_\alpha &= nm\langle\eta_\alpha\eta_\beta\rangle V_\alpha - nm\left\langle\frac{\eta^2}{3}\delta_{\alpha\beta}\right\rangle V_\alpha\\ &= nm\langle\eta_\alpha\eta_\beta\rangle V_\alpha - n\left\langle\frac{m\eta^2}{3}\right\rangle V_\beta = nm\langle\eta_\alpha\eta_\beta\rangle V_\alpha - nTV_\beta\end{aligned}$$

equation (17) becomes

$$\left\langle \frac{v^2}{2}v_\beta\right\rangle = \left(\frac{1}{2}V^2 + \frac{5}{2}\frac{T}{m}\right)V_\beta + \frac{1}{mn}\pi_{\alpha\beta}V_\alpha + \frac{q_\beta}{mn}\ . \tag{18}$$

Additionally, it is reasonable to rewrite the integral in (15) as

$$\begin{aligned}\int \frac{mv^2}{2} C d^3v &= \int \frac{mV^2}{2} C d^3v + \int m\mathbf{V}\boldsymbol{\eta} C d^3v + \int \frac{m\eta^2}{2} C d^3v \\ &= \frac{mV^2}{2} \int C d^3v + \mathbf{V} \cdot \int m\boldsymbol{\eta} C d^3v + \int \frac{m\eta^2}{2} C d^3v \\ &= \mathbf{VR} + Q \end{aligned} \tag{19}$$

by using (5), (10) and the heat $Q := \int \frac{m\eta^2}{2} C d^3v$ generated due to collisions with particles of the other type. Combining all the findings from (16), (18) and (19), equation (15) becomes the so-called energy transport equation

$$\frac{\partial}{\partial t}\left(\frac{nm}{2}V^2 + \frac{3}{2}nT\right) + \frac{\partial}{\partial x_\beta}\left[\left(\frac{nm}{2}V^2 + \frac{5}{2}nT\right)V_\beta + (\pi_{\alpha\beta}V_\alpha) + q_\beta\right] = q_c n\mathbf{EV} + \mathbf{RV} + Q \ . \tag{20}$$

To get the heat-balance equation, there are still some modifications to be carried out. Writing the three equations of motion (13) as a vector equation and multiplying it by $\mathbf{V}$ results in

$$mn\frac{\partial \mathbf{V}}{\partial t}\mathbf{V} + mn\left[(\mathbf{V}\cdot\nabla_\mathbf{r})\cdot\mathbf{V}\right]\cdot\mathbf{V} = -\nabla_\mathbf{r}(nT)\mathbf{V} - \frac{\partial \pi_{\alpha\beta}}{\partial x_\beta}V_\alpha + q_c n\mathbf{EV} + \mathbf{RV} \ .$$

The first, third and fourth term can be modified by reverse chain and product rules. According to equation (50) in appendix A.2, the second term is equal to $\frac{nm}{2}\mathbf{V}\nabla_\mathbf{r}(V^2)$. Applying the reverse product rule again and using the equation of continuity (8) yields

$$\begin{aligned}\frac{mn}{2}\frac{\partial V^2}{\partial t} + \frac{mn}{2}\nabla_\mathbf{r}(\mathbf{V}V^2) + \frac{m}{2}V^2\frac{\partial n}{\partial t} =& -\nabla_\mathbf{r}(nT\mathbf{V}) + nT\nabla_\mathbf{r}(\mathbf{V}) - \frac{\partial \pi_{\alpha\beta}V_\alpha}{\partial x_\beta} \\ &+ \pi_{\alpha\beta}\frac{\partial V_\alpha}{\partial x_\beta} + q_c n\mathbf{EV} + \mathbf{RV} \ .\end{aligned}$$

Using the reverse product rule one last time for the first and third term and then subtracting by the equation of energy transport (20), the heat-balance equation is derived

$$\frac{3}{2}\frac{\partial nT}{\partial t} + \nabla_\mathbf{r}\left(\frac{3}{2}nT\mathbf{V}\right) + nT\nabla_\mathbf{r}\mathbf{V} + \pi_{\alpha\beta}\frac{\partial V_\alpha}{\partial x_\beta} + \nabla_\mathbf{r}\mathbf{q} = Q \ . \tag{21}$$

## 2.2 1D transport equations along magnetic field lines

The coupled continuity (8), motion (12) and energy transport equation (20) can be re-written for 1D plasma movement along magnetic field lines and additional source terms [6] as

$$
\begin{aligned}
&\frac{\partial n}{\partial t} + \frac{\partial nV}{\partial x} = S \\
&\frac{\partial m_i nV}{\partial t} + \frac{\partial}{\partial x}\left(m_i nV^2 + nT_i - \frac{4}{3}\eta_0^i \frac{\partial V}{\partial x}\right) = -eEn + 0.71n\frac{\partial T_e}{\partial x} \\
&\frac{\partial m_e nV}{\partial t} + \frac{\partial}{\partial x}\left(m_e nV^2 + nT_e - \frac{4}{3}\eta_0^e \frac{\partial V}{\partial x}\right) = eEn - 0.71n\frac{\partial T_e}{\partial x} \\
&\frac{\partial}{\partial t}\left(\frac{m_i nV^2}{2} + \frac{3}{2}nT_i\right) + \frac{\partial}{\partial x}\left(\frac{m_i nV^3}{2} + \frac{5}{2}nVT_i - \kappa_i\frac{\partial T_i}{\partial x}\right) \\
&= \frac{3}{2}ST_s^i + \frac{4}{3}\eta_0^i\left(\frac{\partial V}{\partial x}\right)^2 + \frac{3m_e n}{m_i \tau_e}(T_e - T_i) + eEnV \\
&\frac{\partial}{\partial t}\left(\frac{m_e nV^2}{2} + \frac{3}{2}nT_e\right) + \frac{\partial}{\partial x}\left(\frac{m_e nV^3}{2} + \frac{5}{2}nVT_e - \kappa_e\frac{\partial T_e}{\partial x}\right) \\
&= \frac{3}{2}ST_s^e + \frac{3m_e n}{m_i \tau_e}(T_i - T_e) - eEnV
\end{aligned}
\tag{22}
$$

with nuclear charge number $Z$, densities $n := n_i = n_e/Z$, velocities $V := V_i = V_e$, temperatures $T_{i,e}$, electric field $E$ and masses $m_{i,e}$. Indices $i$ and $e$ indicate the ion and electron quantities, respectively. $S$ represents a constant particle source with temperatures $T_s^{i,e}$ along the interval $[-L, L]$. The boundary conditions are specified as $V(\pm L) = \pm\sqrt{(T_i(\pm L) + T_e(\pm L))/m_i}$ and $q_{e,i}(\pm L) = \gamma_{e,i} n(\pm L)V(\pm L)T_{e,i}(\pm L)$. Here, $q_{e,i}$ are the thermal heat fluxes and can be calculated by $q_{e,i} = -\kappa_{e,i}\partial T_{e,i}/\partial x$. The so-called energy transmission coefficients of the sheath as given in [7, p. 4] are $\gamma_i = 3.5$ and $\gamma_e = 5.0$. The ion and electron viscosity coefficients $\eta_{i,e}$ are given in [4] as

$$\eta_0^i = 0.96 n_i T_i \tau_i \qquad \text{and} \qquad \eta_0^e = 0.73 n_e T_e \tau_e \ .$$

The thermal conductivities as given in [7] are

$$\kappa_i = 3.9\frac{n_i T_i \tau_i}{m_i} \qquad \text{and} \qquad \kappa_e = 3.2\frac{n_e T_e \tau_e}{m_e} \ .$$

And the collision times due to Coulomb interactions are also given in [4] as

$$\tau_i = \frac{3\sqrt{m_i}}{4\sqrt{\pi}\lambda e^4 Z^4 n_i}\cdot T_i^{3/2} \qquad \text{and} \qquad \tau_e = \frac{3\sqrt{m_e}}{4\sqrt{2\pi}\lambda e^4 Z^2 n_i}\cdot T_e^{3/2}$$

with electron charge $e$ and Coulomb logarithm $\lambda$. Hence, viscosities and thermal conductivities are proportional to $T_{i,e}^{5/2}$. This leads to strong non-linearities in equations (22).

## 2.3 Neural networks

Now that the transport equations are derived, they have to be solved. Even in one dimension, analytical solutions exist only for special cases. Therefore, numerical methods are needed. Well-known examples are finite differences, finite elements, finite volumes and Monte-Carlo methods (e.g., [8], [9], [10], [11]). In this thesis, physics-informed neural networks are used. These are a special type of neural networks for solving differential equations. They are introduced in the following.
Neural networks (NNs), often referred to as artificial neural networks (ANNs), belong to the field of machine learning (ML). All they do is to transform a given input vector into an output vector in a way that is either very complex to code or not even feasible as a common algorithm.
Figure 1 shows the basic structure of a multilayer perceptron. Technically, a multilayer perceptron is a special kind of neural network. Hereafter, both terms might be used synonymous because it will be the only NN type considered in this thesis.

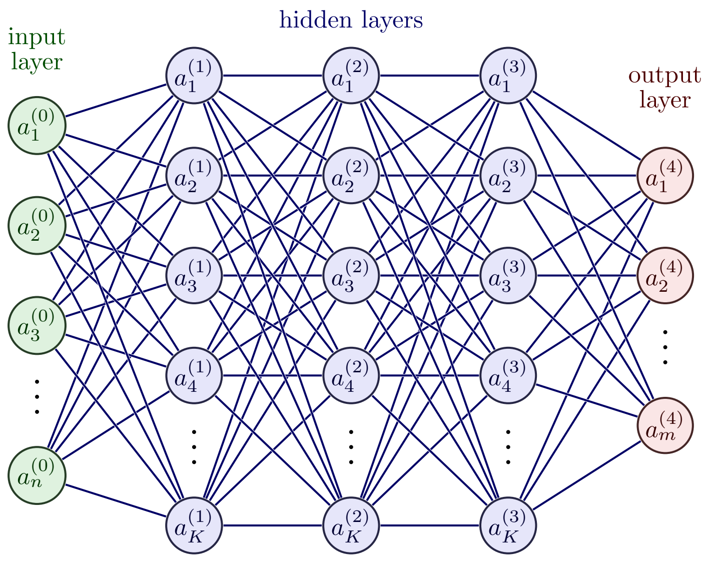


**Figure 1:** Structure of an exemplary neural network with three hidden layers (adapted from Neutelings [12]).

The building blocks of a neural network are its neurons, illustrated by circles in the figure. Each neuron carries a value $a_k^{(l)}$ that often but not necessarily lies between 0 and 1. This value is called activation of the respective neuron. Each column of neurons $l$ is called layer. The input layer $\mathbf{a}^{(0)}$ (left) and output layer $\mathbf{a}^{(4)}$ (right) consist directly of the input vector $\mathbf{x}$ and output vector $\mathbf{y}$ values, respectively. The layers in between are known as hidden layers, because usually only the input and output layers are accessed by users. A neural network with more than one hidden layer is called deep neural network (DNN) [13].
A neuron of a hidden or output layer is connected to every neuron of the previous layer. Here, connected refers to how the activation is calculated. It is done by

$$a_j^{(l)} = f\left(\sum_{k=1}^{K} w_{jk}^{(l)} \cdot a_k^{(l-1)} + b_j^{(l)}\right), \tag{23}$$

where $w_{jk}^{(l)}$ and $b_j^{(l)}$ are the weights and biases, respectively [13]. $f$ is called activation function. There are many functions used as activation function. But for each, $f$ is non-linear and often fulfills $f(x) \in [0,1]$. Chapter 2.3.1 will give more details on activation

functions.
When calculating all the activations in a layer, it is useful to rewrite equation (23) in matrix notation

$$\mathbf{a}^{(l)} = f\left(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right). \tag{24}$$

Here, $f$ acts element-wise onto the vector and $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector, respectively. Thus, the forward calculation, often referred to as feed-forward, in a neural network of $L+1$ layers is a composition of such functions

$$\mathbf{a}^{(L)} = g\left(\mathbf{W}^{(L)} f\left(\mathbf{W}^{(L-1)} \ldots\, f\left(\mathbf{W}^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)}\right) \ldots + \mathbf{b}^{(L-1)}\right) + \mathbf{b}^{(L)}\right). \tag{25}$$

Here, the last activation function $g$ might differ from case to case. All the knowledge of neural networks is encoded in the values of weights and biases. From that perspective, neural networks appear to be less intelligent than claimed to be.
When initializing a neural network, the weights and biases are randomly chosen.[1] Usually after the initialization, a given input vector will most probably not result in the wanted output vector. For this reason, a neural network has to be trained. In the training process, the weights and biases are adjusted in a way that output vectors satisfy the expectations better and better. An example could be the image of a leaf encoded as a vector containing all the pixel's normalized RGB values. When defining the output vector interpretation by how it is done in figure 2 the wanted output for a leaf image is the unit vector $\widehat{\mathbf{e}}_2$.

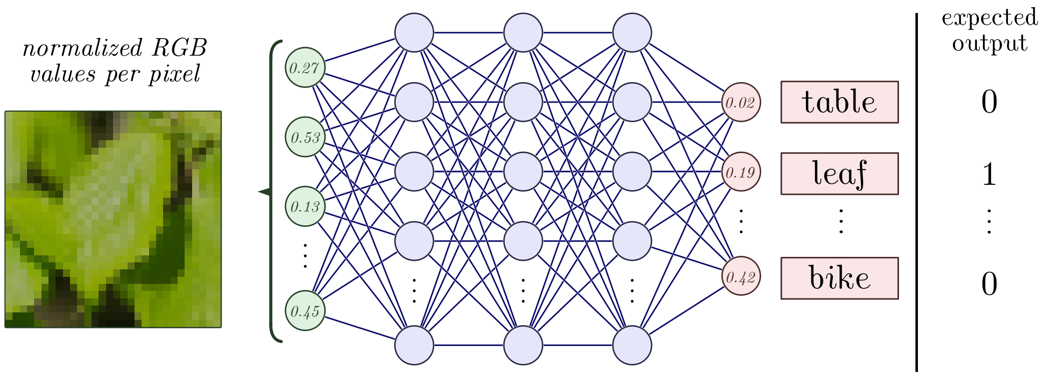


**Figure 2:** Exemplary feed-forward to classify an image. The input vector contains normalized RGB values per pixel. The output vector contains probabilities for every class. The expected output differs because the NN was not trained yet. Image recognition is commonly done by convolutional neural networks (CNNs, see [15, 5.4 p. 90]) instead and only serves the purpose of exemplification here (NN adapted from Neutelings [12]).

The deviation between the output $\mathbf{y}$ and the wanted vector $\widehat{\mathbf{e}}_2$ is called loss and can be written as

$$\|\mathbf{y} - \widehat{\mathbf{e}}_2\|\,,$$

where $\|\cdot\|$ is the Euclidean norm. The obvious goal is to decrease this deviation. But even if it gets zero for this particular image, there might be deviations for different images of leafs or other objects. That is why a vast training data set is needed. The mean of quadratic deviations of all training examples from that set

$$C(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{y}_n(\boldsymbol{\theta}) - \widehat{\mathbf{y}}_n\|^2 \tag{26}$$

[1]There are many ways to randomly initialize NN parameters. An overview for how it is done in the Python package PyTorch can be found in [14].

is called cost.[1] In principle, different cost functions could be used. The mean squared error function (MSE) is common. $\boldsymbol{\theta}$ contains the network parameters, e.g., weights and biases. That is why loss $C$ and output vectors $\mathbf{y}_n$ are functions of $\boldsymbol{\theta}$. The expected output vectors $\widehat{\mathbf{y}}_n$ do not depend on $\boldsymbol{\theta}$, since they are given in the data set of length $N$.
Now, the goal is to minimize the cost function with respect to network parameters $\boldsymbol{\theta}$. Doing this analytically is not feasible due to the huge number of parameters and data examples. Therefore, a method known as gradient descent is used. Chapter 2.3.3 will cover more details on that. The only important fact right now is the need of calculating the gradient of $C$ with respect to $\boldsymbol{\theta}$. Knowing the gradient at a given point in parameter space allows a good guess of where a local minimum might be. Doing this step by step and adjusting the weights and biases accordingly will yield smaller costs and therefore satisfy the expectations better and better. Hence, differentiation methods of calculations within neural networks are needed to get the gradient. Chapter 2.3.2 covers more information on that.
In general, neural networks are used in two cases. These are classification and regression [16]. The example above is a typical classification scenario. In this case, output values should lie between 0 and 1. Additionally, all the output values should add up to 1. Those conditions yield reasonable probabilities. They are easily implemented by using the softmax function in the last layer. In contrast, expressing some mathematical function like temperature distribution with respect to position $x$ by neural networks is referred to as regression. In principle, normalization in the last layer is not needed for that.

### 2.3.1 Activation functions

As mentioned earlier, the results of matrix multiplication and vector addition are evaluated by activation functions $f$ and $g$ in every layer. This is no unnecessary complexity but a crucial step for the feed-forward of equation (25). Omitting $f$ and $g$ or choosing them as the identity functions would result in

$$\begin{aligned}\mathbf{a}^{(L)} &= \mathbf{W}^{(L)}\left(\mathbf{W}^{(L-1)}\dots\left(\mathbf{W}^{(1)}\mathbf{a}^{(0)}+\mathbf{b}^{(1)}\right)\dots+\mathbf{b}^{(L-1)}\right)+\mathbf{b}^{(L)}\\ &= \mathbf{W}^{(L)}\mathbf{W}^{(L-1)}\dots\mathbf{W}^{(1)}\mathbf{a}^{(0)}+\left(\mathbf{W}^{(L)}\dots\left(\left(\mathbf{W}^{(2)}\mathbf{b}^{(1)}\right)+\mathbf{b}^{(2)}\right)\dots\right)+\mathbf{b}^{(L)}\\ &=: \tilde{\mathbf{W}}\mathbf{a}^{(0)}+\tilde{\mathbf{b}}\ .\end{aligned}$$

Hence, the entire neural network collapses and only a linear problem remains. This would look similar if $f$ was chosen linear. The many weights and biases would combine to a matrix $\tilde{\mathbf{W}}$ and vector $\tilde{\mathbf{b}}$ of fixed sizes. Changing the number of layers or neurons per layer would not affect the model complexity at all. This phenomenon is called linear degeneracy [17]. Therefore, the activation function and its non-linearity in particular is a neural network's defining element.
Many activation functions can be used, but some are more common. One example is the sigmoid function $\sigma$ already mentioned. It is defined by

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

Its derivative is

$$\begin{aligned}\nabla\sigma(x) := \frac{\partial\sigma}{\partial x} &= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2}\\ &= \sigma(x)\cdot(1-\sigma(x))\ .\end{aligned}$$

[1]The terms loss, cost and error often are used synonymous. In any case, something like the deviation of the expected output vector and received one is meant.

Derivatives of activation functions are important for automatic differentiation. Both the sigmoid function and its derivative can be seen in figure 3. Sigmoid returns values between 0 and 1. This can be shown by considering the limits $x \to \infty$ and $x \to -\infty$.

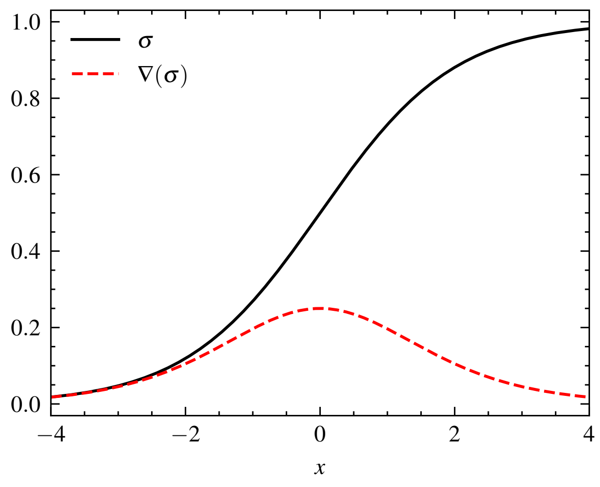


**Figure 3:** Sigmoid activation function $\sigma$ and its derivative $\nabla(\sigma)$.

The rectified linear unit function known as ReLU (see figure 4) is defined by

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} .$$

Obviously, this function is not bounded for the $x \to \infty$ limit. Hence, ReLU maps into $[0, \infty)$. Even though it is often used, its derivative

$$\nabla(\mathrm{ReLU})(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$

is not defined in $x = 0$. This might cause problems when evaluating ReLU's derivative in automatic differentiation.

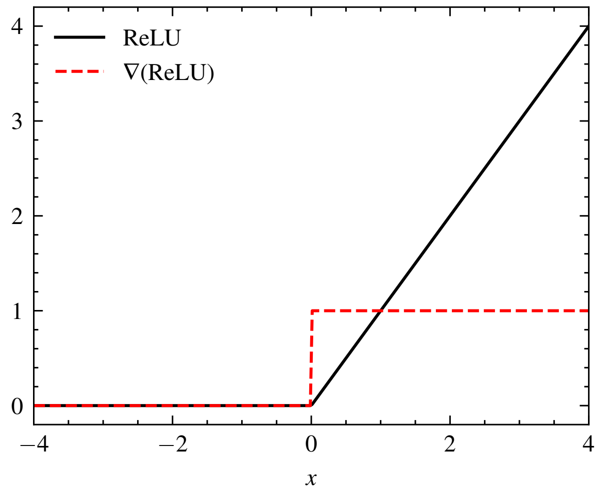


**Figure 4:** ReLU activation function and its derivative $\nabla$(ReLU).

The last activation function considered in this thesis is the hyperbolic tangent (tanh) function. It is defined by

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \; .$$

Hence, its derivative can be written as

$$\nabla(\tanh)(x) = \frac{\cosh(x)\cosh(x) - \sinh(x)\sinh(x)}{\cosh^2(x)} = 1 - \tanh^2(x) \; .$$

It is related to sigmoid by the following expression [18, p. 191].

$$\tanh(x) = 2 \cdot \sigma(2x) - 1$$

Again, both tanh and its derivative are shown in figure 5.

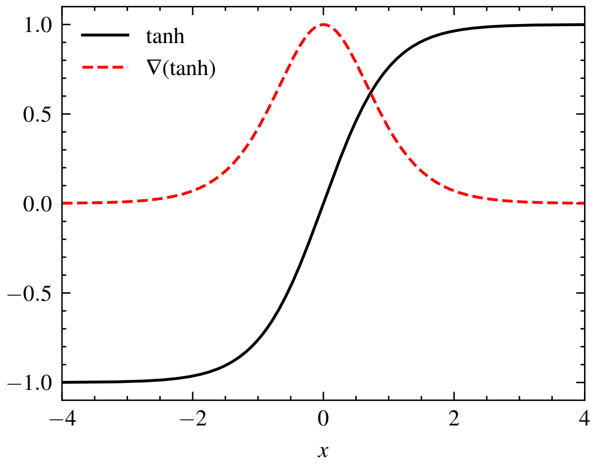


**Figure 5:** Tanh activation function and its derivative $\nabla(\tanh)$.

Unlike the previous functions, tanh returns values between $-1$ and 1 antisymmetrically around $x = 0$. The derivative is well-defined in $(-\infty, \infty)$. It can be noted that the gradient of tanh is much larger around 0 than the gradient of sigmoid.

### 2.3.2 Automatic differentiation

As mentioned earlier, calculating the gradient of the cost function with respect to network parameters $\boldsymbol{\theta}$ is needed for training the neural network. Additionally, differentiating the NN's output with respect to input variables like position $x$ is needed to calculate the loss function of a physics-informed neural network (see chapter 2.4). A differentiation method is required in both cases.
In [19] several differentiation methods like finite differences, symbolic differentiation and automatic differentiation (AD) are compared. It is stated that using the reverse mode of automatic differentiation applies best for NNs. One reason is the small numerical error in comparison with finite differences. Moreover, the large number of variables for which the derivatives have to be calculated do not affect the computational effort too much. The latter is an advantage over finite differences but also symbolic and forward automatic differentiation. The following explanation follows [19, p. 12f].
For deriving a function $f$ by reverse mode AD, at first $f$ has to be evaluated at a specific point. While executing all the elementary computations, that when composited lead to

the whole function $f$, all the intermediate variables $v_\mathrm{i}$ have to be stored. Additionally, the dependencies between these $v_\mathrm{i}$ have to be saved in a way such that a computational graph as shown in figure 6 can be created. The actual deriving process starts at the end of that graph. For every node, the so-called adjoint

$$\bar{v}_\mathrm{i} = \frac{\partial f}{\partial v_\mathrm{i}}$$

can be calculated. The last node fulfills

$$\bar{v}_5 = \frac{\partial f}{\partial v_5} = 1 \ .$$

As stated in [20, p. 4], all the nodes in front can now be determined using the following chain rule where $\mathrm{Pa}(j)$ denotes the set of parent nodes of child node $j$. When using forward computation, the intermediate value of the parent is directly used to calculate the child's one.

$$\begin{aligned} \bar{v}_\mathrm{i} = \frac{\partial f}{\partial v_\mathrm{i}} &= \sum_{j:\ i \in \mathrm{Pa}(j)} \frac{\partial f}{\partial v_\mathrm{j}} \frac{\partial v_\mathrm{j}}{\partial v_\mathrm{i}} \\ &= \sum_{j:\ i \in \mathrm{Pa}(j)} \bar{v}_\mathrm{j} \frac{\partial v_\mathrm{j}}{\partial v_\mathrm{i}} \end{aligned} \tag{27}$$

Hence, the name reverse mode refers to the step-by-step calculation of the adjoints from the end to the beginning of the computational graph. Since the partial derivatives at the end of (27) may depend on the $v_\mathrm{i}$, it is necessary to store them in the forward calculation process. Obviously, the partial derivatives of any intermediate variable with respect to its parents have to be known and implemented. This is viable since only elementary computations have to be considered.

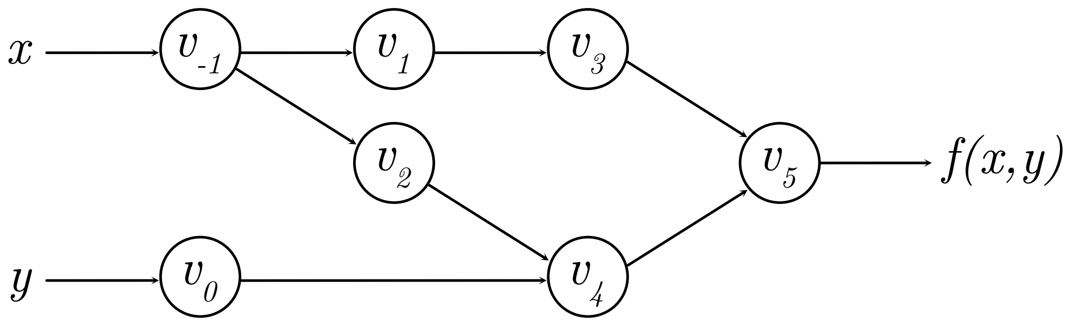


**Figure 6:** Computational graph, referring to the exemplary function $f(x, y) = \sin^2(x) + y \cdot e^x$ and intermediate variables of equation (28) (adapted from [19, figure 4]).

The computational graph in figure 6 refers to the exemplary function $f(x, y) = \sin^2(x) + y \cdot e^x$. The $v_\mathrm{i}$ are calculated by

$$\begin{aligned} v_{-1} &= x \ , \qquad v_0 = y \\ v_1 &= \sin(v_{-1}) \\ v_2 &= e^{v_{-1}} \\ v_3 &= v_1^2 \\ v_4 &= v_2 \cdot v_0 \\ v_5 &= v_3 + v_4 \ . \end{aligned} \tag{28}$$

Now the adjoints can be calculated by

$$
\begin{aligned}
\bar{v}_5 &= 1 \\
\bar{v}_4 &= \bar{v}_5 \frac{\partial v_5}{\partial v_4} = \bar{v}_5 \cdot 1 \\
\bar{v}_3 &= \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot 1 \\
\bar{v}_2 &= \bar{v}_4 \frac{\partial v_4}{\partial v_2} = \bar{v}_4 \cdot v_0 \\
\bar{v}_1 &= \bar{v}_3 \frac{\partial v_3}{\partial v_1} = \bar{v}_3 \cdot 2v_1 \\
\bar{v}_0 &= \bar{v}_4 \frac{\partial v_4}{\partial v_0} = \bar{v}_4 \cdot \bar{v}_2 = \frac{\partial f}{\partial y} \\
\bar{v}_{-1} &= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}} = \bar{v}_2 \cdot e^{v_{-1}} + \bar{v}_1 \cdot \cos(v_{-1}) = \frac{\partial f}{\partial x}
\end{aligned}
$$

beginning at the end of the computational graph and step by step moving towards the variable of interest. At the end of this particular recursion, the wanted derivatives of $f$ with respect to $x$ and $y$ are determined.
In the context of neural networks, the calculation of derivatives of the cost function with respect to network parameters by slightly adjusted reverse mode AD is called back-propagation. Here, the derivatives of the activation functions (see 2.3.1), matrix multiplications and vector additions are already known. In case of ReLU, the derivative at $x = 0$ would have to be chosen.

### 2.3.3 Gradient descent and optimizers

Now that the gradient can be calculated, the cost function needs to be minimized. Since gradients point in the direction of greatest increase of the considered function, the negative gradient points in the direction of greatest decrease. Therefore, moving in the direction of $-\nabla C$ in parameter space should reduce the cost. Typically, a learning rate $\gamma$ is used to prevent from jumping over minima. This method is referred to as gradient descent [13]. The corresponding equation for updating network parameters $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \nabla C(\boldsymbol{\theta}_t) \ . \tag{29}$$

Figure 7 illustrates the idea behind (29) in the simplified case of only one parameter $\theta$, cost function $C(\theta) = \theta^3 - \theta^2 + 0.2$ and learning rate $\gamma = 0.2$. Starting at $\theta_0 = 1$, the gradient is negative and relatively large. Therefore, the gradient descent step is relatively large, too. At $\theta_1$, the gradient is much smaller. Hence, $\theta_2$ lies close to it.
Overall, the training process consists of the following steps. Firstly, all the training data inputs feed-forward through the neural network and yield some resulting outputs. These outputs are compared with the expected ones to calculate the cost. The cost is derived with respect to all network parameters to get the gradient. Then the gradient is used in equation (29) to calculate the improved parameters. This procedure repeats until the loss converges. Every such iteration through the entire training data set is called an epoch.
There are several advanced variations of this procedure. For example, splitting the training data set in random subsets (batches) leads to smaller computational effort of calculating the cost. The downside is the inaccuracy of the received gradients. Stochastic gradient descent (SGD) is a special case in which every mini batch only contains one training example [21].

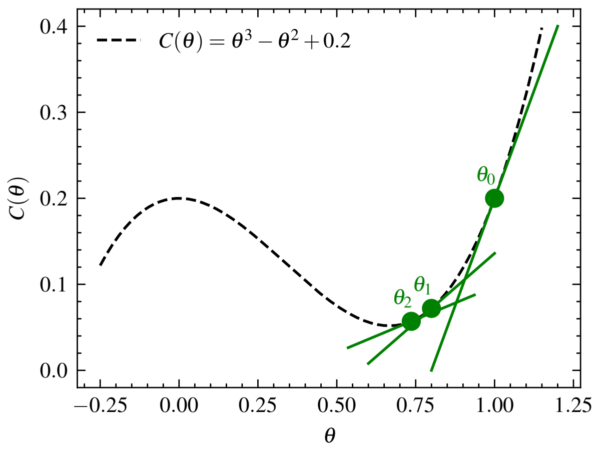


**Figure 7:** Two gradient descent iterations in case of only one parameter $\theta$, cost function $C(\theta) = \theta^3 - \theta^2 + 0.2$ and learning rate $\gamma = 0.2$. Initially, the parameter is set to $\theta_0 = 1$.

There are also improvements of equation (29). As described in [22], another hyperparameter $\mu \in [0, 1]$ can be added to calculate the exponentially weighted average $\mathbf{b}_\mathrm{t}$ by

$$\mathbf{b}_\mathrm{t} = \mu \cdot \mathbf{b}_{\mathrm{t}-1} + \nabla C(\boldsymbol{\theta}_t) \ , \quad \mathbf{b}_0 = 0$$

and then using it to update the parameters by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \mathbf{b}_\mathrm{t} \ .$$

This is called momentum method. Because the $\mathbf{b}_\mathrm{t}$ contain gradients of previous iterations, the optimization path is smoothed in parameter space. Therefore, fewer oscillations in parameter space are expected for arriving at the minimum. Additionally, when reaching a local minimum ($\nabla C(\boldsymbol{\theta}_t) \approx 0$) there is still momentum to pass it and eventually find a better one. Standard gradient descent is the special case $\mu = 0$.
Another optimization algorithm is the so-called RMSprop (root-mean-square propagation). According to [23], the moving average of the element-wise squared gradient $\mathbf{s}_\mathrm{t}$ is calculated by

$$\mathbf{s}_\mathrm{t} = \beta \cdot \mathbf{s}_{\mathrm{t}-1} + (1 - \beta) \cdot (\nabla C(\boldsymbol{\theta}_t))^2 \ , \quad \mathbf{s}_0 = 0$$

and then adjusts the parameters by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma}{\sqrt{\mathbf{s}_\mathrm{t}}} \cdot \nabla C(\boldsymbol{\theta}_t) \ .$$

The square root acts element-wise on $\mathbf{s}_\mathrm{t}$. The main idea of the resulting learning rate $\frac{\gamma}{\sqrt{\mathbf{s}_\mathrm{t}}}$ is that oscillations of a parameter result in large moving averages due to the squaring, and therefore lead to less movement in this direction. This dampens the oscillations. The element-wise operations ensure the consideration for each individual parameter.
The adaptive moment estimation optimizer, also known as Adam optimizer, was introduced in [24]. Adam combines momentum and RMSprop algorithms. At first, the moving averages $\mathbf{m}_\mathrm{t}$ and $\mathbf{v}_\mathrm{t}$ are calculated by

$$\mathbf{m}_\mathrm{t} = \beta_1 \cdot \mathbf{m}_{\mathrm{t}-1} + (1 - \beta_1) \cdot \nabla C(\boldsymbol{\theta}_t) \ , \quad \mathbf{m}_0 = 0$$
$$\mathbf{v}_\mathrm{t} = \beta_2 \cdot \mathbf{v}_{\mathrm{t}-1} + (1 - \beta_2) \cdot (\nabla C(\boldsymbol{\theta}_t))^2 \ , \quad \mathbf{v}_0 = 0$$

where $\beta_1$ and $\beta_2$ are the decay rates. Additionally, these averages are corrected by

$$\widehat{\mathbf{m}}_\mathrm{t} = \frac{\mathbf{m}_\mathrm{t}}{1-\beta_1^t}$$
$$\widehat{\mathbf{v}}_\mathrm{t} = \frac{\mathbf{v}_\mathrm{t}}{1-\beta_2^t}$$

to ensure reasonable averages in the first few iterations. The effect of this correction reduces by increasing number of iterations $t$. At the end, the parameters are updated by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \cdot \frac{\widehat{\mathbf{m}}_\mathrm{t}}{\sqrt{\widehat{\mathbf{v}}_\mathrm{t}} + \varepsilon}$$

where $\varepsilon$ is a small number to avoid division by zero.
There are several more optimizers. Some examples are Adadelta, Adagrad, AdamW, SparseAdam, Adamax, ASGD, LBFGS, NAdam, RAdam, RMSprop, Rprop. In [25] they are listed and their underlying algorithms are explained. Often there are only minor modifications between them. There does not exist a general best optimizer, it depends on the case.

### 2.3.4 Overfitting and regularization

When training a neural network with more parameters than training examples, overfitting might occur. That is the case when the model predicts the training outputs very well, but fails at validation data which was separated from the training data set before. Figure 8 shows an exemplary training curve with overfitting. The training loss decreases rapidly, while the validation loss remains constant or even increases. Hence, the neural network does not generalize the problem well.

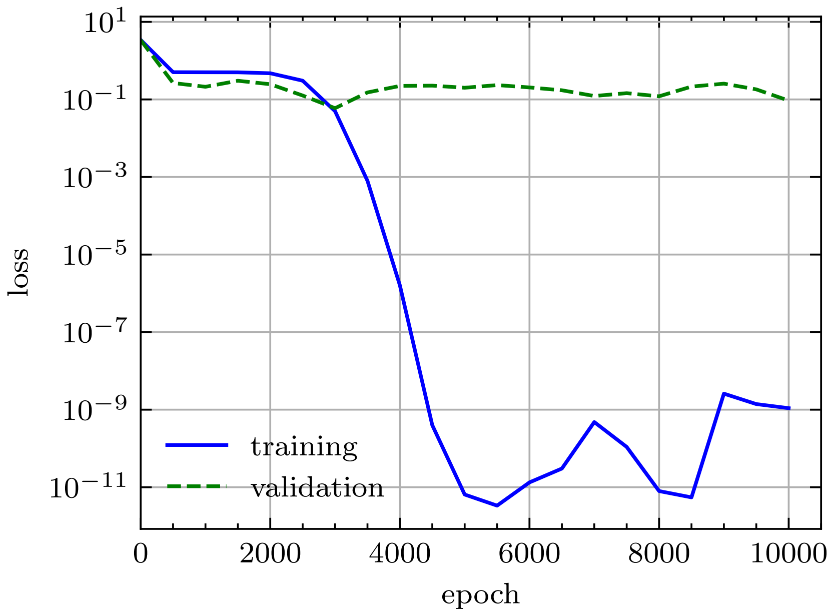


**Figure 8:** Loss versus epochs in case of overfitting in logarithmic scale. The training curve decreases while the validation loss stays constant.

To avoid such learning behavior, it is best to use more training data or reduce the number of parameters. If more data is not available, the method of regularization can be used.

For this, the regularization loss

$$L_{reg} = \lambda_{reg} \sum_{w \in \boldsymbol{\theta}} w^2$$

as sum over the weights $w$ is added to the previous loss.[1] This leads to smaller weights. Therefore, only fewer weights influence the network output and the risk of overfitting is reduced.
Another possibility is weight decay, as explained in [26]. Here, the update rule for the weights $\mathbf{w}$ is given by

$$\mathbf{w}_{t+1} = (1 - \lambda_{wd})\mathbf{w}_t - \gamma \nabla C(\mathbf{w}_t)$$

with weight decay parameter $\lambda_{wd}$, which reduces the weights in every iteration. For the SGD optimizer, both methods are equivalent.

## 2.4 Physics-informed neural networks

Most of the work is already done to approximate a function $u : U \to \mathbb{R}$ with $U \subset \mathbb{R}^n$ by the function $u_{\boldsymbol{\theta}} : U \to \mathbb{R}$ corresponding to the neural network with network parameters $\boldsymbol{\theta}$ such that

$$u_{\boldsymbol{\theta}}(\mathbf{x}) \approx u(\mathbf{x})$$

for every $\mathbf{x} \in \mathbb{R}^n$. In principle, the neural network could be trained by data of measurements or analytical solutions and the common loss function given in equation (26). But if there is no experimental data and the underlying differential equation is not analytically solvable, physics-informed neural networks (PINNs) might be useful. The given explanation follows [27].
A partial differential equation of the form

$$\mathrm{D}[u](\mathbf{x}) = f(\mathbf{x}) \quad \forall \mathbf{x} \in U, \qquad \mathrm{B}[u](\mathbf{x}) = g(\mathbf{x}) \quad \forall \mathbf{x} \in V \subset \partial U \tag{30}$$

is considered, where D is a partial differential operator acting on $u$, and $f : U \to \mathbb{R}$ and $g : V \to \mathbb{R}$ are not dependent on $u$. The second equation describes the boundary condition (BC) using an operator B on the boundary $\partial U$ of $U$. The goal is to approximate the solution $u(\mathbf{x})$ by a NN. For this purpose the so-called residual $r$ is introduced as

$$r(\mathbf{x}) = \mathrm{D}[u](\mathbf{x}) - f(\mathbf{x}).$$

Therefore, if $u$ is a solution of (30), the residual will be equal to 0. Thus, it is reasonable to minimize the residual loss

$$L_r = \frac{1}{R} \sum_{i=1}^{R} r_{\boldsymbol{\theta}}^2(\mathbf{x}_i) = \frac{1}{R} \sum_{i=1}^{R} \left(\mathrm{D}[u_{\boldsymbol{\theta}}](\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2 \tag{31}$$

for chosen $\mathbf{x}_i \in U$. To take the boundary condition into account, the boundary loss

$$L_{bc} = \frac{1}{B} \sum_{j=1}^{B} \left(\mathrm{B}[u_{\boldsymbol{\theta}}](\mathbf{x}_j) - g(\mathbf{x}_j)\right)^2$$

[1]Similarly, the sum over the absolute values of the weights can be used.

for chosen $\mathbf{x}_j \in V$ can be defined. When using Dirichlet BC the loss

$$L_{bc} = \frac{1}{B} \sum_{j=1}^{B} (u_{\boldsymbol{\theta}}(\mathbf{x}_j) - u(\mathbf{x}_j))^2$$

is used. Here, the values of $u$ are given at the respective points. It works analogous for other types of boundary conditions. If $u_{\boldsymbol{\theta}}$ outputs the same values at these points as $u$, $L_{bc}$ would be equal to 0. Hence, the boundary loss as well as the residual loss needs to be minimized. This can be realized by considering the overall loss function

$$L = \lambda_r L_r + \lambda_{bc} L_{bc} \tag{32}$$

with scaling parameters $\lambda_r$ and $\lambda_{bc}$. If needed, regularization terms can be added as well.

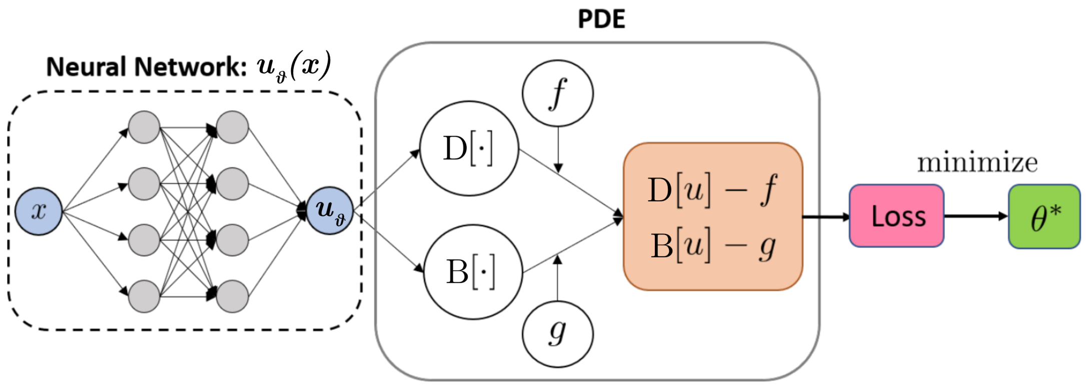


**Figure 9:** Structure of a physics-informed neural network to approximate the function $u$ that fulfills problem (30). The network computes $u_{\boldsymbol{\theta}}$ for chosen points $x$ and then uses automatic differentiation to calculate the residual and boundary loss. At the end, the network parameters $\theta$ are updated to $\theta^*$. (adapted from [27, fig. 1])

Figure 9 illustrates the structure of a physics-informed neural network. To calculate the derivatives in the residual loss, the neural network function $u_{\boldsymbol{\theta}}$ has to be derived with respect to input variables and not to weights or biases. Since automatic differentiation can proceed in front of the first NN layer as well, this does not lead to any new problems. When keeping track of the calculations needed to get the derivative, even the computation of higher order derivatives in the loss function can be done without problems. However, the computational graph becomes even more complex and therefore more memory is needed to store the intermediate variables and their dependencies. Since the network generates its own loss with no need of prepared training data physics-informed neural networks learn unsupervised.
As explained in [28, remark 2.5], the PINN function $u_{\boldsymbol{\theta}}$ has to be differentiable depending on the differential operator D. The used activation functions have to be sufficiently smooth. They state that the sigmoid and tanh functions should be preferred over ReLU in PINNs for this reason. As stated in [29, p. 8], since ReLU is a combination of two linear functions, its second derivative (and higher) is always zero. This prevents the PINN from learning, even if one completes the gradient of ReLU to ensure differentiability.
The given PINN model can be extended to $C$ coupled PDEs by introducing one residual loss $L_r^{(k)}$ for every equation, respectively. The overall loss is the sum of all these residual losses and the boundary loss.

$$L_{\text{coupled}} = \sum_{k=1}^{C} \lambda_r^{(k)} L_r^{(k)} + \lambda_{bc} L_{bc}$$

Vector-valued functions $\mathbf{u}$ can also be realized by summing individual losses of the respective components. If the scaling parameters $\lambda_r^{(k)}$ of these terms are equal, it would be equivalent to taking the square of the Euclidean norm of the vector-valued functions in equation (31).

## 2.5 PyTorch

Implementing all the parts of physics-informed neural networks, the network structure, activation functions, automatic differentiation and gradient based optimization algorithms, by oneself would be a big effort. Fortunately, software packages like PyTorch for Python were developed to simplify this process.
As explained in its repository [30], PyTorch makes calculations with tensors on the CPU and also on the GPU possible. This speeds up the calculations considerably. In this regard, it is a competitor to the popular Numpy package. But PyTorch also provides reverse mode automatic differentiation by creating computational graphs if desired to calculate gradients for the calculations done in the PyTorch environment. PyTorch is a great help to implement neural networks and especially PINNs simply but effectively.

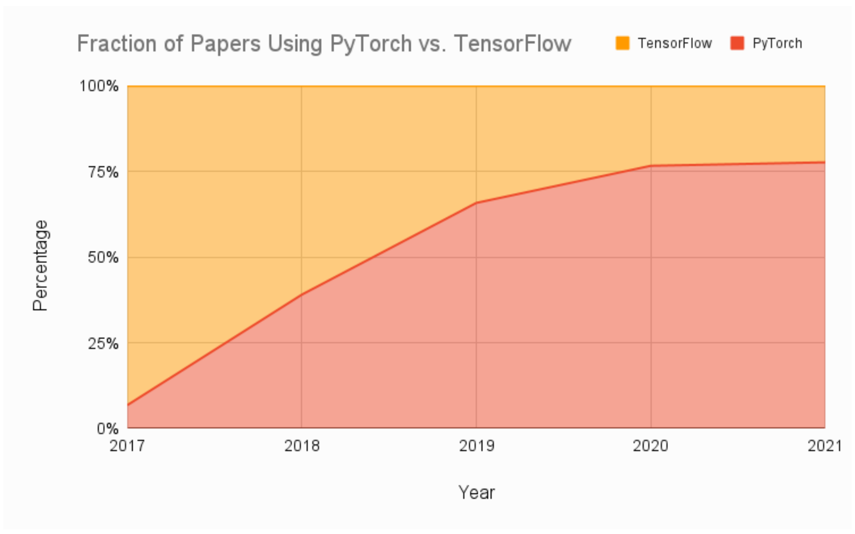


**Figure 10:** Fraction of papers in eight research journals using PyTorch vs. TensorFlow until 2021 (taken from [31]).

A similar and also popular python package for creating neural networks is TensorFlow. But as shown in figure 10, most of the authors of papers in research journals until 2021 were using PyTorch.

# 3 Numerical results for 1D problems

After introducing the basics of the transport equations and the PINNs, in this chapter different validation problems are formulated to test the PINNs. The first five test cases are taken from [11]. Here, analytical solutions are derived to compare the PINN's results with. Finally, an attempt is made to solve a simplified version of the full transport equations (22). The implementation is done using PyTorch. The codes can be accessed in [32].

## 3.1 Pure heat conduction problem

The first considered problem is the simplified heat-balance equation from (21) for the temperature $T$

$$\frac{\partial}{\partial t}\left(\frac{3}{2}nT\right)+\frac{\partial}{\partial x}\left(-\kappa\frac{\partial T}{\partial x}\right)=0\ ,\quad n=n_0\ ,\ \kappa=\kappa_0$$

with constant density $n$, linear thermal conductivity $\kappa$ and boundary conditions $T(0)=T_0,\ T(L)=T_L$. Since only the steady state is of interest, it is reasonable to neglect the derivative with respect to time by $\frac{\partial}{\partial t}\left(\frac{3}{2}nT\right)\to 0$. Hence,

$$\frac{\partial^2 T}{\partial x^2}=0$$

remains. Integrating two times with respect to $x$ yields

$$T(x)=c_1x+c_2\ ,$$

where the constants $c_1$ and $c_2$ can be identified by the boundary conditions

$$c_2=T_0\ ,\ c_1=\frac{T_L-T_0}{L}$$

and therefore the temperature distribution in the steady state becomes

$$T(x)=\frac{T_L-T_0}{L}\cdot x+T_0\ .$$

By replacing the variables using $\widetilde{x}=\frac{x}{L}$, $\widetilde{T}=\frac{T}{T_0}$ and $\widetilde{T}_L=\frac{T_L}{T_0}$ the differential equation and its solution become

$$\frac{\partial^2\widetilde{T}}{\partial\widetilde{x}^2}=0\ ,\quad \widetilde{T}(\widetilde{x})=\left(\widetilde{T}_L-1\right)\cdot\widetilde{x}+1\ . \tag{33}$$

The PINN was trained by this equation to approximate the temperature distribution. By definition of the new variables, the boundary condition on the left is $\widetilde{T}(0)=1$. On the right, $\widetilde{T}(1)=\widetilde{T}_L=3$ was chosen. The grid for computing the residual loss is given by 500 equidistant points in $[0,1]$. To compute a validation loss, a uniformly distributed random grid of 500 points is used. This grid is regenerated after every epoch. These grid structures are reused in the following test cases, since increasing the number of grid points (e.g., 5000) never lead to better results. The used NN consists of five hidden layers, each with five neurons, to handle the complexity of the problem but also to not require an unnecessary amount of resources. The input and output layer only have one neuron, respectively (position $\widetilde{x}$, temperature $\widetilde{T}$). Hence, the overall number of network parameters is 136. By comparing the number of parameters and of grid points, overfitting is not expected. The intermediate values in every layer except for the last are evaluated by the sigmoid activation function. The last one is not evaluated by any activation function[1], since the output should be the non-normalized, dimensionless temperature. Boundary and residual loss are added without additional scaling ($\lambda_{bc} = \lambda_r = 1$) to build the overall loss. No regularization terms are used. Figure 11 shows the results for these conditions after 40000 epochs when using the Adam optimizer.

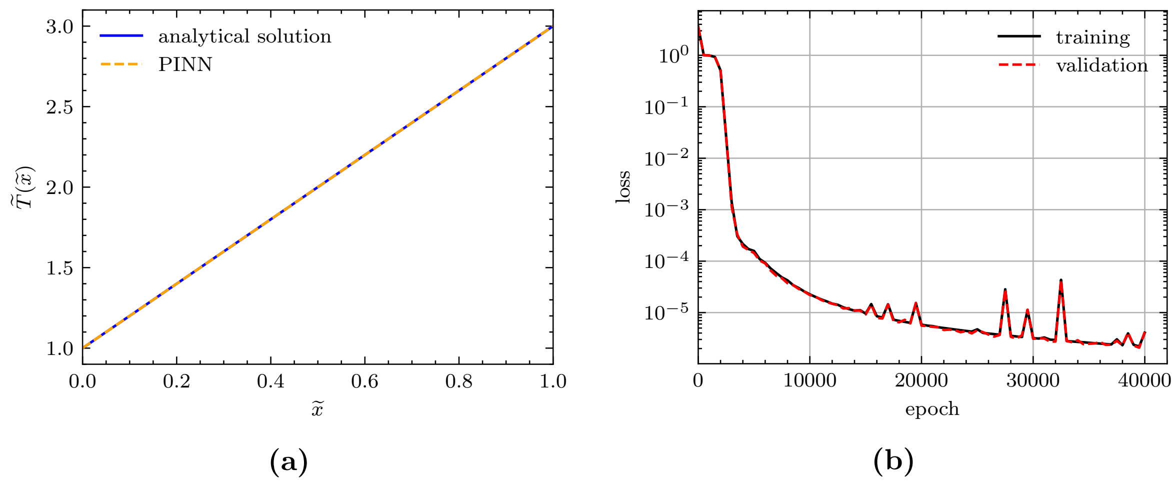


**Figure 11:** Numerical results of problem (33) solved by the PINN. (a) analytical solution and approximation by PINN of the temperature distribution $\widetilde{T}(\widetilde{x})$. (b) training and validation loss curves.

The PINN's and the analytical solution fit perfectly. The loss converges, even though some local peaks can be seen. The training and validation loss curves are mainly the same. Hence, there is no suspicion of overfitting.
To compare the different optimizers the training process was done by SGD (lr=0.001), SGD (lr=0.001 and $\mu$=0.9), Adam, AdamW, Adagrad, Adadelta and RMSprop as well. The respective loss training curves are given in figure 12.

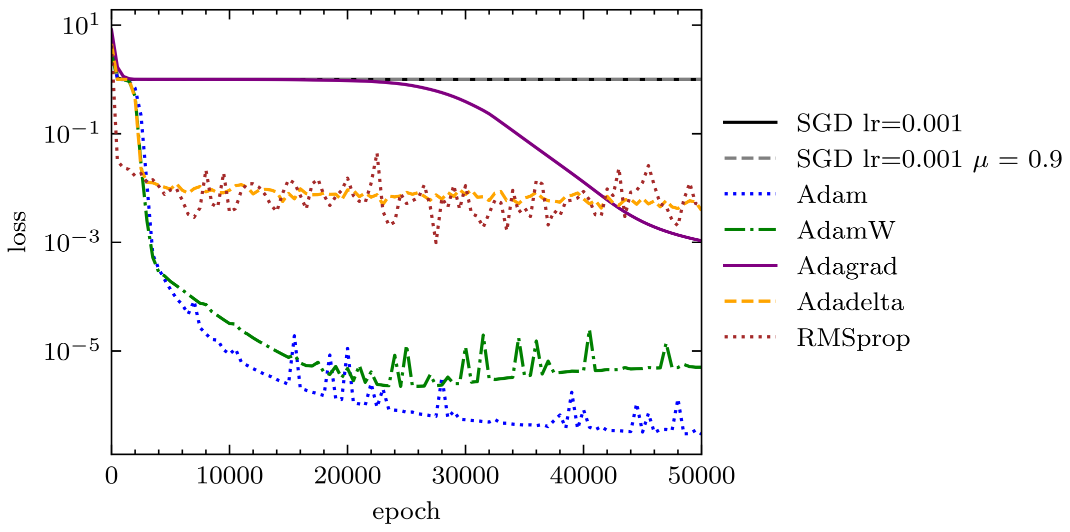


**Figure 12:** Comparison of training curves for the shown optimizers.

With and without momentum, the SGD optimizer does not converge at all or both found the same unsatisfactory local minimum. Adadelta and RMSprop seem to stuck in a

[1]In principle, it is the identity activation function.

different local minimum since both oscillate around the same loss. Adagrad reduces the loss very slowly but continuously. Adam and AdamW show the best performance. Both reduce the loss quickly, but after about 25000 epochs AdamW increases again. Despite some peaks mentioned, Adam continues to converge.
Some of these results could have been expected because all the optimizers besides SGD adapt their learning rates automatically and therefore find better minima. But Adagrad's monotonic decreasing learning rate drops too rapidly with epochs to reach the minimum found by Adam. Adam and AdamW probably do better because they combine the two advanced methods momentum and RMSprop. The increasing training curve of AdamW might be caused by the additional weight decay in comparison with the basic Adam algorithm. [25]
Using these results as benchmark, Adam seems to be the best optimizer here and is used in the next cases.

## 3.2 Convection-conduction coupling

Now a convection term with constant velocity $V = V_0$ is added to the previous test problem. The equation modifies to

$$\frac{\partial}{\partial t}\left(\frac{3}{2}nT\right) + \frac{\partial}{\partial x}\left(\frac{3}{2}nVT - \kappa\frac{\partial T}{\partial x}\right) = 0 \ , \quad n = n_0 \ , \ \kappa = \kappa_0 \ , \ V = V_0$$

with boundary conditions $T(0) = T_0$, $T(L) = T_L$. In the presence of a steady state, the time derivative can be neglected. The remaining equation is

$$\frac{\partial}{\partial x}\left(\frac{3}{2}n_0V_0T - \kappa_0\frac{\partial T}{\partial x}\right) = 0 \ .$$

Hence, the constant $c_3$ can be introduced to get the ordinary differential equation

$$\frac{3}{2}n_0V_0T - \kappa_0\frac{\partial T}{\partial x} = c_3 \ .$$

The homogeneous equation can be solved by an exponential ansatz leading to

$$T_{\text{hom}}(x) = c_4 \cdot e^{\frac{3}{2}\frac{n_0V_0}{\kappa_0}\cdot x} \ .$$

A particular solution of the inhomogeneous equation is

$$T_{\text{part}}(x) = \frac{2}{3}\frac{c_3}{n_0V_0} \ .$$

The general solution as sum of homogeneous and particular solution is

$$T(x) = c_4 \cdot e^{\frac{3}{2}\frac{n_0V_0}{\kappa_0}\cdot x} + \frac{2}{3}\frac{c_3}{n_0V_0} \ .$$

Hence, the constants $c_3$ and $c_4$ can be determined by using the boundary conditions.

$$c_4 = \frac{T_0 - T_L}{1 - e^{\frac{3}{2}\frac{n_0V_0}{\kappa_0}\cdot L}} \ , \quad c_3 = \frac{3}{2}n_0V_0 \cdot \left(T_0 - \frac{T_0 - T_L}{1 - e^{\frac{3}{2}\frac{n_0V_0}{\kappa_0}\cdot L}}\right)$$

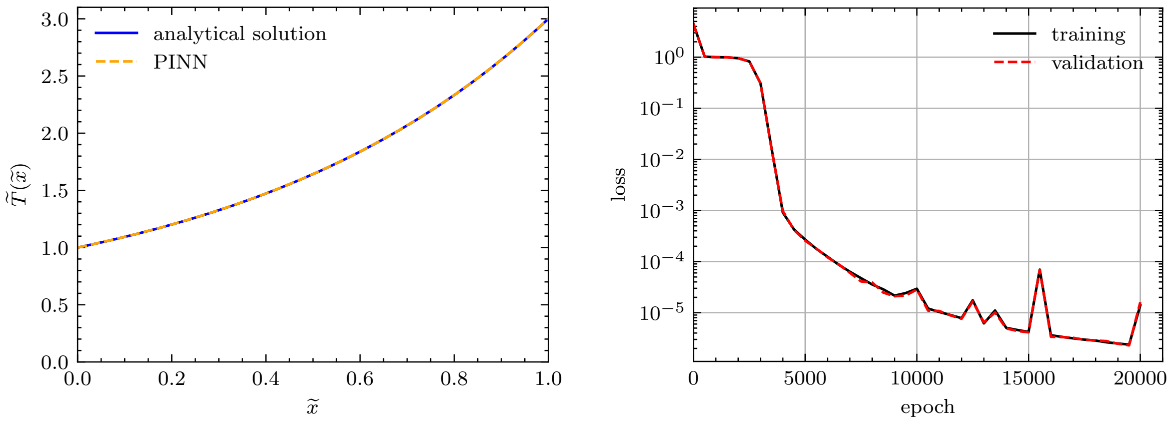
Therefore, the overall solution becomes

$$T(x) = T_0 + \frac{T_L - T_0}{e^{\frac{3}{2}\frac{n_0V_0}{\kappa_0}\cdot L} - 1} \cdot \left(e^{\frac{3}{2}\frac{n_0V_0}{\kappa_0}\cdot x} - 1\right) \ .$$

Introducing $\widetilde{x} = \frac{x}{L}$, $\widetilde{T} = \frac{T}{T_0}$ and $\widetilde{T}_L = \frac{T_L}{T_0}$ again, and choosing the dimensionless constant $\frac{n_0 V_0}{\kappa_0} = 1$ yields the following differential equation and its solution.

$$\frac{\partial}{\partial \widetilde{x}}\left(\frac{3}{2}\widetilde{T} - \frac{\partial \widetilde{T}}{\partial \widetilde{x}}\right) = 0 \ , \quad \widetilde{T}(\widetilde{x}) = 1 + \frac{\widetilde{T}_L - 1}{e^{\frac{3}{2}} - 1} \cdot \left(e^{\frac{3}{2}(\widetilde{x}-1)} - 1\right) \tag{34}$$

A PINN with the same structure as in 3.1 is used to approximate the temperature distribution by this equation. The boundary conditions are again chosen to be $\widetilde{T}(0) = 1$ and $\widetilde{T}(1) = \widetilde{T}_L = 3$. The Adam optimizer is used. Again the loss factors are chosen to be $\lambda_{bc} = \lambda_r = 1$. The results after 20000 epochs can be seen in figure 13.



**(a)** **(b)**

**Figure 13:** Numerical results of problem (34) solved by the PINN. (a) analytical solution and approximation by PINN of the temperature distribution $\widetilde{T}(\widetilde{x})$. (b) training and validation loss curves.

Again, the analytical and PINN's solution match perfectly. The loss decreases rapidly during the first 5000 epochs. Thereafter, it seems to converge, nevertheless there are small peaks. Training and validation loss curves are the same, even at these peaks. Hence, overfitting is not a problem here.

## 3.3 Conduction with non-linearity

The third problem extends the pure-conduction problem of chapter 3.1 by introducing the non-linearity $\kappa = \kappa_0 \cdot T^{\frac{5}{2}}$. The equation becomes

$$\frac{\partial}{\partial t}\left(\frac{3}{2}nT\right) + \frac{\partial}{\partial x}\left(-\kappa_0 \cdot T^{\frac{5}{2}}\frac{\partial T}{\partial x}\right) = 0 \ , \quad n = n_0$$

with boundary conditions $T(0) = T_0$, $T(L) = T_L$. Neglecting the time derivative and introducing the constant $c_5$ leads to

$$-\kappa_0 T^{\frac{5}{2}}\frac{\partial T}{\partial x} = c_5 \ .$$

Integrating both sides of the equation from the left boundary $x' = 0$ to a general $x$ yields

$$\int_{T_0}^{T(x)} -\kappa_0 T'^{\frac{5}{2}} dT' = \int_0^x c_5 dx'$$

$$\Rightarrow \qquad -\frac{2\kappa_0}{7}\left(T^{\frac{7}{2}} - T_0^{\frac{7}{2}}\right) = c_5 \cdot x \ .$$

The constant $c_5$ can be determined by considering the right boundary $x = L$.

$$c_5 = -\frac{2\kappa_0}{7L}\left(T_L^{\frac{7}{2}} - T_0^{\frac{7}{2}}\right)$$

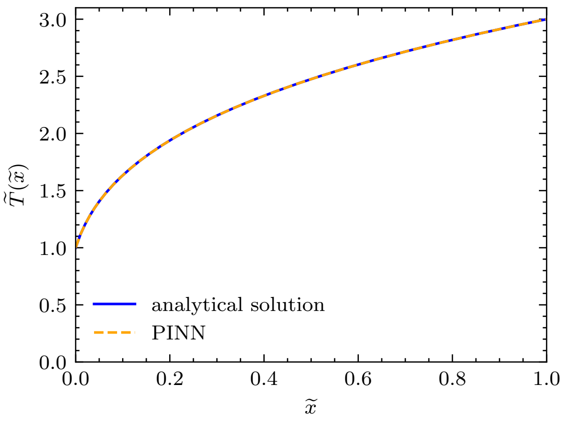
Therefore, the overall solution becomes

$$T(x) = \left[\left(T_L^{\frac{7}{2}} - T_0^{\frac{7}{2}}\right) \cdot \frac{x}{L} + T_0^{\frac{7}{2}}\right]^{\frac{2}{7}} .$$

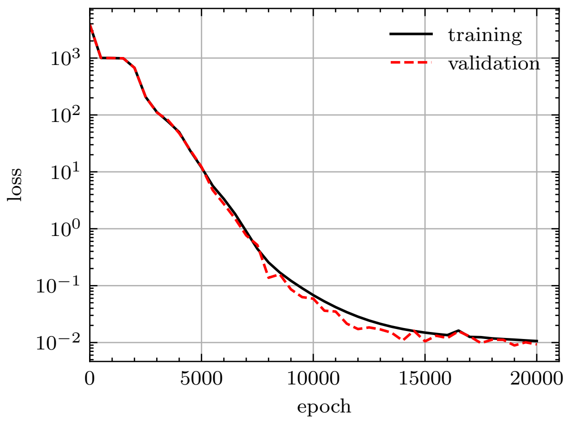
Introducing $\widetilde{x} = \frac{x}{L}$, $\widetilde{T} = \frac{T}{T_0}$ and $\widetilde{T}_L = \frac{T_L}{T_0}$ again, leads to

$$\frac{\partial}{\partial \widetilde{x}}\left(\widetilde{T}^{\frac{5}{2}} \frac{\partial \widetilde{T}}{\partial \widetilde{x}}\right) = 0 , \qquad \widetilde{T}(x) = \left[\left(\widetilde{T}_L - 1\right) \cdot \widetilde{x} + 1\right]^{\frac{2}{7}} . \tag{35}$$

A PINN of the same structure as in 3.1 is used to approximate the temperature distribution by this equation. Again, the boundary conditions are chosen to be $\widetilde{T}(0) = 1$ and $\widetilde{T}(1) = \widetilde{T}_L = 3$, and the Adam optimizer is used. Since the previously used loss factors $\lambda_{bc} = \lambda_r = 1$ yield deviations from the boundary conditions, the boundary loss is scaled by $\lambda_{bc} = 1000$. The residual loss factor is kept at $\lambda_r = 1$. When not using an activation function at the end which maps into the positive real numbers, there might occur numerical errors when calculating the residual loss due to $\widetilde{T}^{5/2}$ depending on the random chosen initial weights and biases. Initializing the NN until it works with the respective parameters is an acceptable method for small problems like this. Even if it is not the case here, it should be noted that derivatives which are contained in roots, e.g., $(\partial T/\partial x)^{5/2}$, would lead to further problems. That is because changing the initial monotonic behavior of the PINN's function is not trivial. The results without additional activation function are shown in figure 14.



**(a)**



**(b)**

**Figure 14:** Numerical results of problem (35) solved by the PINN. (a) analytical solution and approximation by PINN of the temperature distribution $\widetilde{T}(\widetilde{x})$. (b) training and validation loss curves.

By changing the boundary loss factor, the analytical and PINN's curve fit perfectly again. An unexpected behavior can be seen in the loss curves. The training curve decreases and seems to converge while the validation curve does the same, but is even smaller from the 8000th epoch onwards.

## 3.4 Coupled convection dominant problem

In this chapter, coupled transport equations are considered. But here, only the continuity and momentum equations are used. The problem is formulated as

$$\begin{aligned} \frac{\partial n}{\partial t} + \frac{\partial}{\partial x}(nV) &= 0 \\ \frac{\partial mnV}{\partial t} + \frac{\partial}{\partial x}\left(mnV^2\right) + \frac{\partial nT}{\partial x} &= 0 \end{aligned} \tag{36}$$

with boundary conditions $n(0) = n_0$ and $V(0) = V_0$. The temperature distribution $T(x)$ is given. The two test cases

$$T_1(x) = \frac{T_L - T_0}{L} \cdot x + T_0 \qquad \text{and} \qquad T_2(x) = \left[\left(T_L^{\frac{7}{2}} - T_0^{\frac{7}{2}}\right) \cdot \frac{x}{L} + T_0^{\frac{7}{2}}\right]^{\frac{2}{7}}$$

are distinguished.
Both time derivative terms can be neglected again due to the wanted steady state. Hence, the continuity equation leads to a constant which can be described by $n_0$ and $V_0$. Therefore, $n$ and $V$ can be expressed by

$$n = \frac{n_0 V_0}{V} \;, \quad V = \frac{n_0 V_0}{n} \;. \tag{37}$$

The second equation in (36) leads to a constant as well. This constant can be described by $n_0$, $V_0$ and $T_0 = T(0)$. $n$ and $V$ from (37) can be inserted into the received equation. Doing it with $n$ yields

$$\begin{aligned} m\frac{n_0V_0}{V}V^2 + \frac{n_0V_0}{V}T &= mn_0V_0^2 + n_0T_0 \\ \Rightarrow \qquad V^2 - \left(V_0 + \frac{T_0}{mV_0}\right) \cdot V + \frac{T}{m} &= 0 \;. \end{aligned}$$

This is a reduced quadratic equation with respect to $V$. Thus, $V$ can be written as

$$V(x) = \frac{V_0}{2} + \frac{T_0}{2mV_0} \pm \sqrt{\left(\frac{V_0}{2} + \frac{T_0}{2mV_0}\right)^2 - \frac{T}{m}} \;,$$

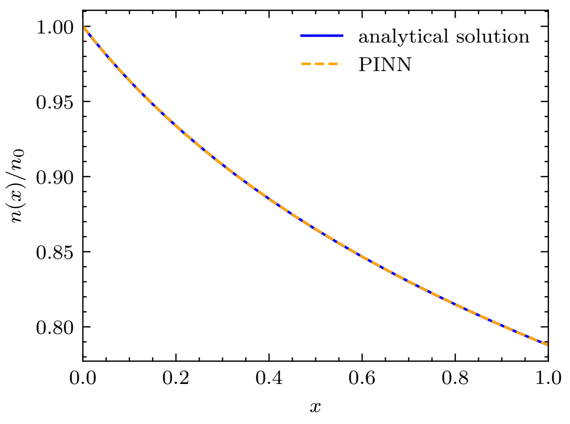
but only the case with positive sign fulfills the boundary condition $V(0) = V_0$. The same procedure can be repeated by inserting $V$ from (37). This yields

$$\begin{aligned} mn\frac{n_0^2V_0^2}{n^2} + nT &= mn_0V_0^2 + n_0T_0 \\ \Rightarrow \quad n^2 - \frac{mn_0V_0^2 + n_0T_0}{T} \cdot n + \frac{mn_0^2V_0^2}{T} &= 0 \;. \end{aligned}$$

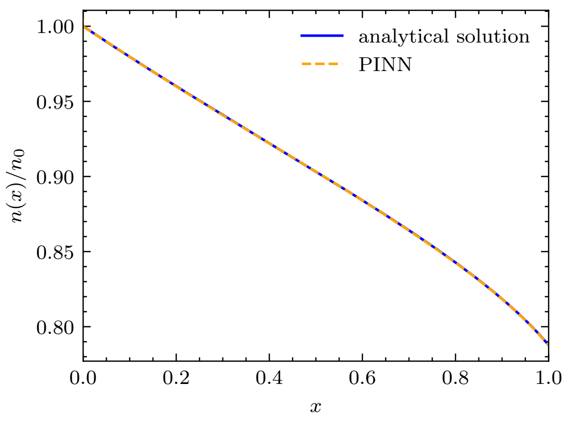
Analogously, $n$ can be written as

$$n(x) = \frac{mn_0V_0^2 + n_0T_0}{2T} \pm \sqrt{\left(\frac{mn_0V_0^2 + n_0T_0}{2T}\right)^2 - \frac{mn_0^2V_0^2}{T}} \;.$$

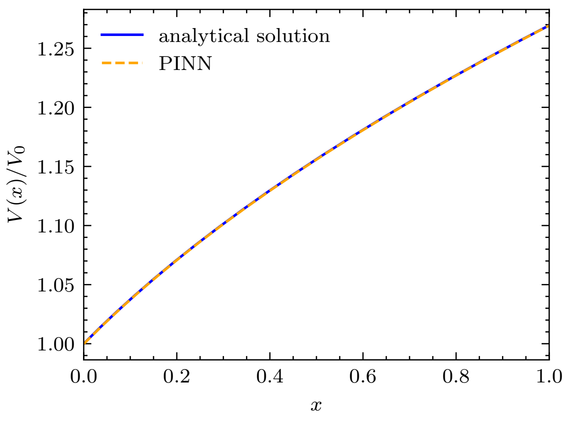
Now, only the solution with negative sign fulfills $n(0) = n_0$.



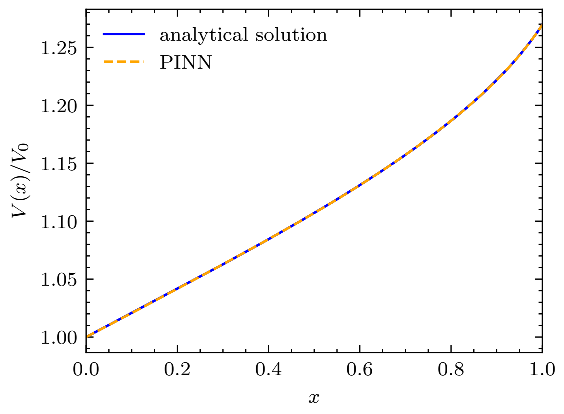
**(a)**



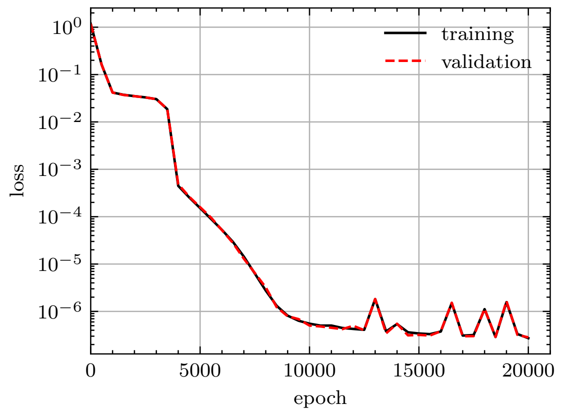
**(b)**



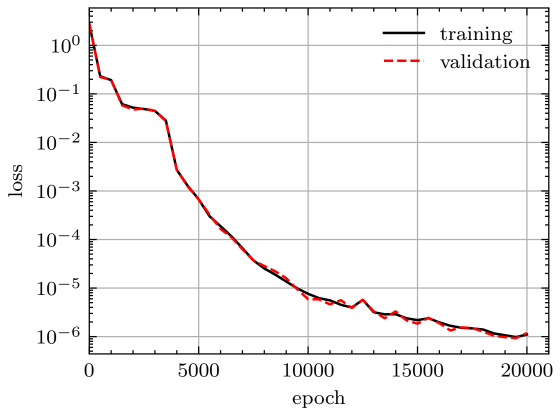
**(c)**



**(d)**



**(e)**



**(f)**

**Figure 15:** Numerical results of problem (36) with linear temperature distribution $T_1(x)$ (left, (a), (c), (e)) and non-linear temperature distribution $T_2(x)$ (right, (b), (d), (f)) solved by the PINN. (a), (b), (c), (d) analytical solutions and approximations by PINN of the normalized density $n(x)/n_0$ and normalized velocity $V(x)/V_0$, respectively. (e), (f) training and validation loss curves.

To train a PINN by the coupled equations (36) the NN structure of chapter 3.1 can not be used again, since not one function should be approximated but two, namely the density $n(x)$ and the velocity $V(x)$. Hence, the last layer has to be changed. Now, the last layer reduces the number of nodes from 5 to only 2. In this way, the PINN approximates not only one function but two at the same time. Additionally, the loss gets a second residual term due to the two coupled equations considered in this problem. The loss factors are chosen to be $\lambda_{bc} = \lambda_r^{(1)} = \lambda_r^{(2)} = 1$, and the boundary conditions to be $n_0 = 1$, $V_0 = 1.5$,

$m = 1$, $T_0 = 1$ and $T_L = 0.5$. The Adam optimizer is used again. Figure 15 shows the results after 20000 epochs. The left column corresponds to the temperature distribution $T_1(x)$ and the right column to $T_2(x)$.
In both temperature cases, the analytical and PINN's solution for the densities $n(x)$ and velocities $V(x)$ match perfectly. Even though the training curve for $T_1$ includes some peaks, the losses converge and no sign of overfitting can be seen.

## 3.5 Coupled problem with energy equation

Now the energy equation is added as well. Thus, the three coupled equations are

$$\begin{aligned} \frac{\partial n}{\partial t} + \frac{\partial}{\partial x}(nV) &= 0 \\ \frac{\partial mnV}{\partial t} + \frac{\partial}{\partial x}\left(mnV^2\right) + \frac{\partial nT}{\partial x} &= 0 \\ \frac{\partial}{\partial t}\left(\frac{3}{2}nT\right) + \frac{\partial}{\partial x}\left(\frac{3}{2}nVT\right) + nT\frac{\partial V}{\partial x} &= 0 \end{aligned} \tag{38}$$

with boundary conditions $n(0) = n_0$, $V(0) = V_0$, $T(0) = T_0$. Like before, all the time derivatives can be neglected due to the considered steady state. Hence, the continuity equation yields the constant

$$nV = n_0V_0 \ . \tag{39}$$

The momentum equation leads to

$$mnV^2 + nT = c_5 = mn_0V_0^2 + n_0T_0 \tag{40}$$

$$\Rightarrow \qquad \frac{3}{2}nVT = \frac{3}{2}\left(c_5 - mnV^3\right) . \tag{41}$$

The energy equation becomes

$$\frac{\partial}{\partial x}\left(\frac{3}{2}nVT\right) + nT\frac{\partial V}{\partial x} = 0 \ ,$$

where the term in the first derivative can be expressed by (41). Thus,

$$-\frac{3}{2}m\frac{\partial}{\partial x}\left(nV^3\right) + nT\frac{\partial V}{\partial x} = 0 \ .$$

The first term can be rewritten using (39) to

$$\begin{aligned} -\frac{3}{2}m\frac{\partial}{\partial x}\left(nV^3\right) &= -\frac{3}{2}m\frac{\partial}{\partial x}\left(nV \cdot V^2\right) = -\frac{3}{2}m\left(\underbrace{\frac{\partial nV}{\partial x}}_{=0} \cdot V^2 + \underbrace{nV}_{=n_0V_0} \cdot 2V \cdot \frac{\partial V}{\partial x}\right) \\ &= -3m\left(n_0V_0V\frac{\partial V}{\partial x}\right) \ . \end{aligned}$$

This yields

$$\frac{\partial V}{\partial x} \cdot \left(nT - 3mn_0V_0V\right) = 0 \ .$$

The bracket term can not be equal to zero for arbitrarily chosen boundary conditions. Therefore, the derivative of $V$ has to be equal to zero. Hence, $V$ is constant with respect to $x$. Finally, using the boundary conditions, (39) and (40) the solutions are the constants

$$n(x) = n_0 \ , \quad V(x) = V_0 \ , \quad T(x) = T_0 \ .$$

The NN from chapter 3.1 has to be adjusted again to solve the system of coupled equations (38) since now the three functions should be approximated, namely the density $n(x)$, velocity $V(x)$ and temperature $T(x)$. Thus, it is reasonable to raise the number of overall nodes by changing the number of nodes per hidden layer from 5 to 10, and reduce the number of nodes from 5 to only 3 in the last layer.[1] Additionally, a third loss term for the third equation has to be added. The loss factors are chosen to be $\lambda_{bc} = \lambda_r^{(1)} = \lambda_r^{(2)} = \lambda_r^{(3)} = 1$, and the boundary conditions to be $n_0 = m = 1$, $V_0 = 1.5$ and $T_0 = 0.5$. The analytical solutions and PINN's approximations can be seen in figure 16. The corresponding training and validation curves can be seen in figure 17.

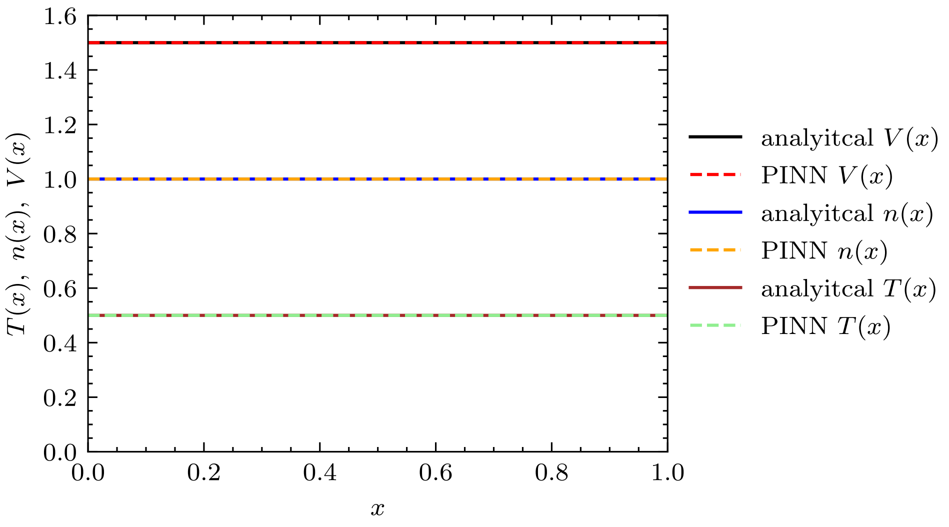


**Figure 16:** Analytical solutions of coupled equations (38) and approximations by PINN of the velocity $V(x)$, density $n(x)$ and temperature $T(x)$, respectively.

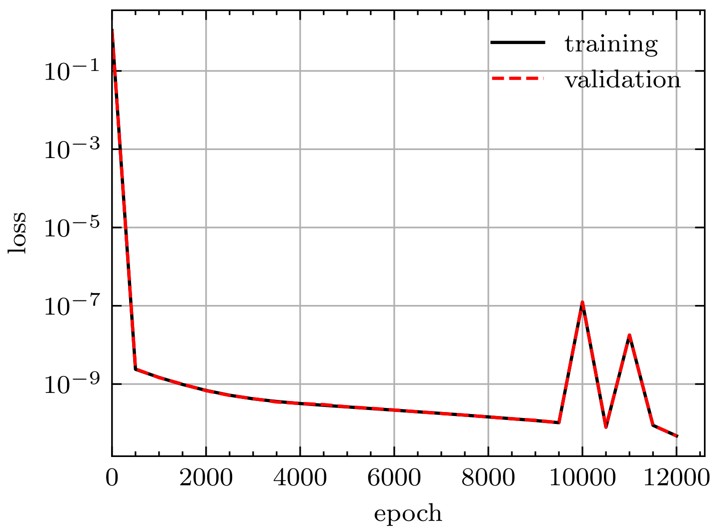


**Figure 17:** Training and validation curves corresponding to the PINN of figure 16.

[1]Each output node for every function $n(x)$, $V(x)$ and $T(x)$, respectively.

All the approximated functions fit the analytical solutions of constants perfectly, and overfitting is not an issue. The training curve is special since the main loss reduction is already done after 1000 epochs. This is much quicker than the previous test cases. This is certainly the case due to the simple structure of the analytical solutions. Even though, two major peaks can be seen around $10^4$ epochs. But the Adam optimizer finds its way back to the previous minimum.

## 3.6 Full coupled equations

Now, a more complete version of equations (22) should be solved. To generate another comparative test case with analytical solutions, the following steady state problem from [6, p. 1] for all $x \in [-L, L]$ is considered.

$$\begin{aligned} \frac{\partial nV}{\partial x} &= S \\ \frac{\partial}{\partial x}\left(m_i n V^2 + nT_i\right) &= 0 \\ \frac{\partial}{\partial x}\left(\frac{m_i n V^3}{2} + \frac{5}{2} n V T_i\right) &= \frac{3}{2} S T_s^i \end{aligned} \tag{42}$$

The boundary conditions are reduced to $V(\pm L) = \pm\sqrt{2T_i(\pm L)/m_i}$. As given in [6], simple integration and application of the boundary conditions yield the analytical solutions of velocity $V$ and density $n$ dependent on the temperature $T_i$.

$$\begin{aligned} n(x) &= Sx \cdot \sqrt{\frac{m_i}{3T_s - 5T_i(x)}} \\ V(x) &= \sqrt{\frac{3T_s - 5T_i(x)}{m_i}} \end{aligned} \tag{43}$$

The temperature distribution is the solution of the reduced quadratic equation

$$T_i^2 - \left(\frac{3}{2}T_s + \frac{5}{16}\left(\frac{P_0}{Sx}\right)^2\right) \cdot T_i + \frac{9}{16}T_s^2 - \frac{3}{16}\left(\frac{P_0}{Sx}\right)^2 \frac{T_s}{m_i} = 0 \tag{44}$$

with constants

$$P_0 = m_i SL\sqrt{\frac{3T_s - 5T_i(L)}{m_i}} + SLT_i(L)\sqrt{\frac{m_i}{3T_s - 5T_i(L)}} \quad \text{and} \quad T_i(\pm L) = \frac{3}{2}\frac{T_s}{c_0^2 + \frac{5}{2}} \ .$$

To solve (42) with a PINN it is reasonable to make it dimensionless. For this, the previous variables are replaced by the following.

$$x \to \xi \cdot L \ , \quad t \to \tau \cdot t_s \ , \quad T_{i,e} \to T_{i,e} \cdot T_s^i \ , \quad V \to V \cdot v_s \ , \quad n \to n \cdot n_s \tag{45}$$

The used characteristic quantities are defined as

$$v_s := \sqrt{\frac{2T_s^i}{m_i}} \ , \quad t_s := \frac{L}{v_s} \ , \quad n_s := St_s \ .$$

Thus, equations (42) apply for all $\xi \in [-1, 1]$ and become

$$\begin{aligned} \frac{\partial nV}{\partial \xi} &= 1 \\ \frac{\partial}{\partial \xi}\left(2nV^2 + nT_i\right) &= 0 \\ \frac{\partial}{\partial \xi}\left(nV^3 + \frac{5}{2}nT_iV\right) &= \frac{3}{2} \end{aligned} \tag{46}$$

with boundary conditions $V(\pm 1) = \pm\sqrt{T_i(\pm 1)}$.
Due to the high complexity of this problem with three coupled equations, three functions to approximate and two boundary conditions, the number of neurons per hidden layer must be increased. Five hidden layers of 50 neurons each are used here. For the calculations, deuterium ($Z = 1$ and $m_i \approx 2 \cdot m_p$) is assumed. The particle source and its temperature are chosen $S = 3 \cdot 10^{-17}\mathrm{s}^{-1}\mathrm{cm}^{-1}$ and $T_s = T_s^i = 30\mathrm{eV}$. All the loss factors are chosen to be $\lambda_r^k = \lambda_{bc} = 1$, $k = 1, 2, 3$. To ensure positive valued densities and temperatures, both are evaluated by the exponential function in the last layer. Figure 18 shows the numerical results achieved with sigmoid activation function after $10^5$ epochs.

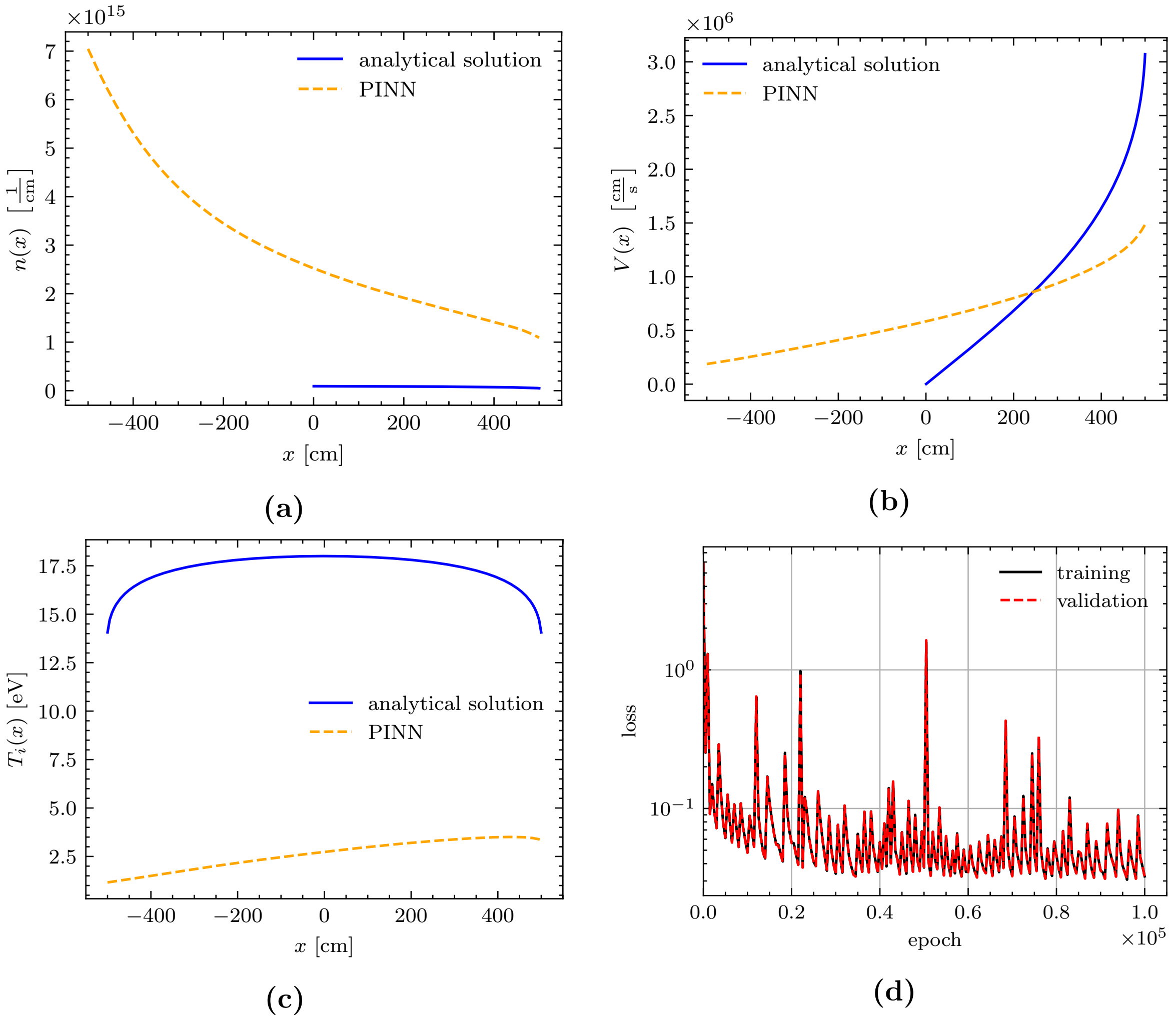


**Figure 18:** Numerical results of problem (46) solved by the PINN with sigmoid activation function, and analytical solutions (43) and (44). (a) density $n(x)$, (b) velocity $V(x)$, (c) ion temperature $T_i(x)$, (d) training and validation losses.

Obviously, these results do not satisfy the expectations at all. Additionally, the loss curve is very noisy. Changing the number of neurons or layers slightly does not have an impact on this outcome. Significant improvement can be made by using the tanh activation function. The results for that can be seen in figure 19.

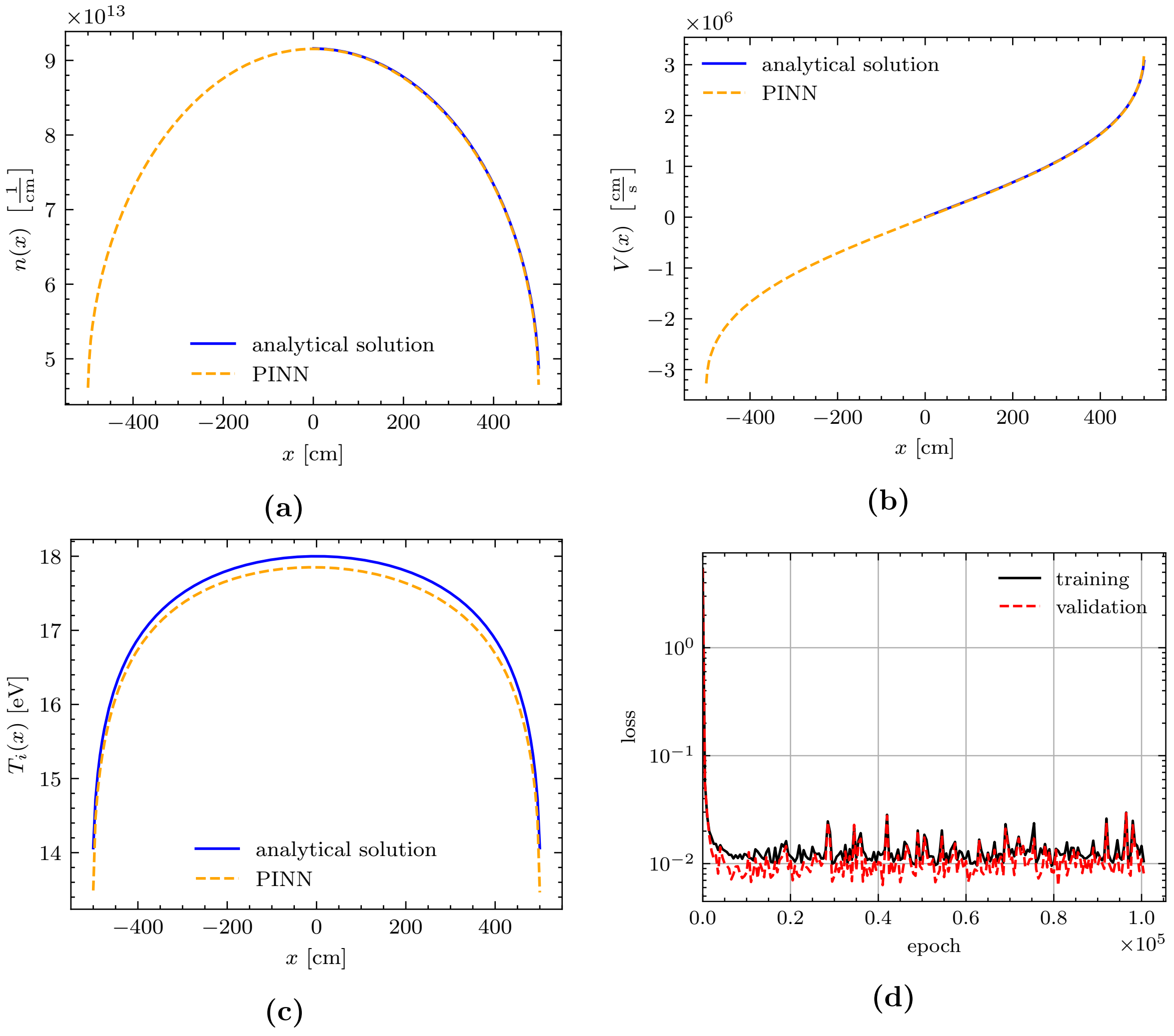


**Figure 19:** Numerical results of problem (46) solved by the PINN with tanh activation function, and analytical solutions (43) and (44). (a) density $n(x)$, (b) velocity $V(x)$, (c) ion temperature $T_i(x)$, (d) training and validation losses.

Even though, the temperature distribution does not fit the analytical curve perfectly, the density and velocity functions are approximated very well by the PINN. Furthermore, the loss does not fluctuate that strongly.

It is well known that tanh often performs better than sigmoid [33, p. 3]. But the reason seems not to be as clear. One reason might be that tanh is more similar to the identity function around 0 and therefore reduces the complexity for training [18, p. 191]. Additionally, as shown in chapter 2.3.1 the gradient of tanh is larger around 0 than the gradient of sigmoid. This might lead to greater optimizer steps and thus faster convergence [34]. But this argument should be viewed with skepticism since larger optimizer steps can also cause missing a minimum.

Now, the full problem (22) in dimensionless form by the use of (45) is studied. Additionally, $m_e/m_i \ll 1$ and source Knudsen number $K_s^i \ll 1$ are assumed so that terms proportional to these factors can be neglected. The remaining steady state equations are

$$\begin{aligned}
&\frac{\partial nV}{\partial \xi} = 1 \\
&\frac{\partial}{\partial \xi}\left(nV^2 + \frac{n(T_i+T_e)}{2}\right) = 0 \\
&\frac{\partial}{\partial \xi}\left(nV^3 + \frac{5}{2}nVT_i\right) = \frac{3}{2} + \sqrt{\frac{2m_e}{m_i}}\frac{3n^2}{K_s^iT_e^{\frac{3}{2}}}(T_e - T_i) + 2F_EnV \\
&\frac{\partial}{\partial \xi}\left(\frac{5}{2}nVT_e - \frac{3.2}{2}\sqrt{\frac{m_i}{2m_e}}K_s^iT_e^{\frac{5}{2}}\frac{\partial T_e}{\partial \xi}\right) = \frac{3T_s^e}{2T_s^i} + \sqrt{\frac{2m_e}{m_i}}\frac{3n^2}{K_s^iT_e^{\frac{3}{2}}}(T_i - T_e) - 2F_EnV
\end{aligned} \tag{47}$$

like in [6, eq. (16)] with electric force and source Knudsen number

$$F_E = \frac{1}{2n}\frac{\partial nT_e}{\partial \xi} + \frac{0.71}{2}\frac{\partial T_e}{\partial \xi}\,, \qquad K_s^i = \frac{3}{2\sqrt{\pi}}\frac{(T_s^i)^{\frac{5}{2}}}{\sqrt{m_i}e^4\lambda L^2}\,,$$

and boundary conditions

$$\begin{aligned}
q_i(\pm 1) &= -\frac{3.9}{2}K_s^iT_i^{\frac{5}{2}}\frac{\partial T_i}{\partial \xi}\bigg|_{\xi=\pm 1} = \gamma_i \cdot nVT_i|_{\xi=\pm 1} \\
q_e(\pm 1) &= -\frac{3.2}{2}\sqrt{\frac{m_i}{2m_e}}K_s^iT_e^{\frac{5}{2}}\frac{\partial T_e}{\partial \xi}\bigg|_{\xi=\pm 1} = \gamma_e \cdot nVT_e|_{\xi=\pm 1} \\
V(\pm 1) &= \pm\sqrt{\frac{T_i(\pm 1) + T_e(\pm 1)}{2}}\,.
\end{aligned}$$

Since the number of functions to approximate by the PINN is increased by the electron temperature the number of neurons is raised to 100. Both temperatures and the density are evaluated by the exponential function in the last layer again. This is of particular importance here because of the non-linear thermal conductivities and boundary conditions. In addition to the previous model, the electron source temperature is set to $T_s^e = 30\text{eV}$ and the source Knudsen number to $K_s^i = 0.15$. As before, using the sigmoid activation function leads to insufficient results, as can be seen in figure 21 in chapter A.3.
Figure 20 shows the results for the tanh activation function. Interestingly, the loss curve suggests even worse results. But at least the shapes of the approximated $n$, $V$ and $T_i$ correspond to their analytical counterpart. Since the analytical solution applies for (46) and not (47) perfect agreement of these curves is not expected anyway. But PINN's approximations differ from the finite volume results in [6, p. 6] as well. Increasing the number of grid points to 5000, or multiplying the strongly non-linear thermal heat flux BCs by 100 or 1000 to make them more relevant does not change the outcome. Even the advanced method of gPINNs[1] and the self-scalable tanh activation function[2] do not yield better results. The main reason seems to be the sheath boundary conditions, which do not contain any fixed numerical values but only dependencies between the functions $n$, $V$, $T_i$ and $T_e$ at the boundary. This is also the essential difference to the test cases of the previous chapters. Apparently, this leads to even more local minima the PINNs can be trapped in. This also makes good results less reproducible. Additionally, it is

[1]More information on gPINNs can be found in [35]. In short, gPINNs consider not only the PDE residual but also its derivative, which has to be equal to 0 as well.

[2]The self-scalable tanh activation function is introduced in [33] as $\text{Stan}(x) = \tanh(x) + \beta x \cdot \tanh(x)$. Here, the scalar $\beta$ is a new trainable NN parameter. Inter alia, Stan can be unbounded depending on $\beta$.

unfortunate that one can not tell from the loss curves themselves which of the PINNs solves the problem best in terms of similarity to the shapes of the analytical solutions.

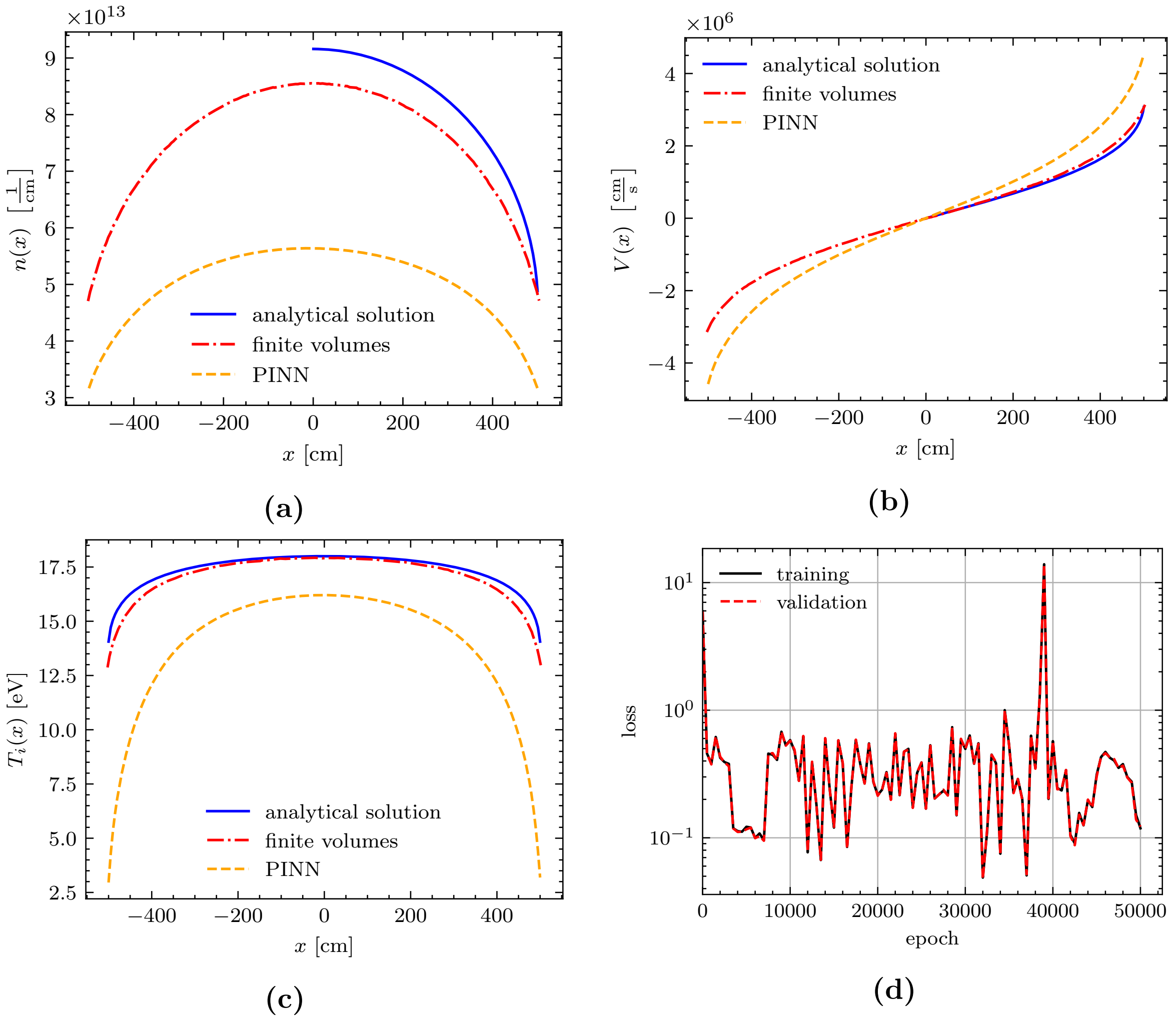


**Figure 20:** Numerical results of problem (47) solved by the PINN with tanh activation function, finite volumes results from [6, p. 6], and analytical solutions (43) and (44). (a) density $n(x)$, (b) velocity $V(x)$, (c) ion temperature $T_i(x)$, (d) training and validation losses.

## 3.7 Numerical efficiency

Finally, to consider PINNs in terms of numerical efficiency, table 1 shows all the previous cases with their corresponding times per $10^4$ epochs and their resulting total time until convergence can be seen. As expected, $t_{10^4}$ increases with cases since they become more complex. The complexity refers to the network structure (i.e., more neurons per hidden layer) but also the number of computations needed to get the residual losses (i.e., more terms per equation and more equations). For the same reason, an increase in RAM usage could be presumed. But interestingly, it remained constant at $\approx 2.6$GB.
The pattern in the change of the total running time $t_{tot}$ is not as clear. At least the cases 3.4 $T_1$ and 3.5 prevent monotonic increasing due to their particular small $N_{conv}$. Additionally, since the last problem does not show convergence at all, no statement can be made. In general, only the orders of magnitude of $N_{conv}$ and $t_{tot}$ should be considered because of the ambiguity of the occurrence of convergence.
In general, numerical methods using finite differences, elements or volumes require only a

few seconds of computation time and therefore are significantly quicker than the PINNs used here. [8] [36]

| Case | $t_{10^4}$ [s] | $N_{conv}$ [$10^3$] | $t_{tot}$ [min] |
|---|---|---|---|
| 3.1 | 70 | 20 | 2.33 |
| 3.2 | 75 | 20 | 2.50 |
| 3.3 | 80 | 20 | 2.67 |
| 3.4 $T_1$ | 85 | 10 | 1.42 |
| 3.4 $T_2$ | 90 | 20 | 3.00 |
| 3.5 | 110 | 1 | 0.18 |
| 3.6 test | 150 | 20 | 5.00 |
| 3.6 full | 350 | - | - |

**Table 1:** For every case the time per $10^4$ epochs $t_{10^4}$ (rounded to nearest 5s), the approximate number of epochs $N_{conv}$ until convergence can be seen, and the resulting total time for these epochs $t_{tot}$ are shown. The training was done on a Nvidia GeForce RTX 3090. In each case, an equal amount of $\approx 2.6$GB of RAM was required.

# 4 Conclusions and outlook

This thesis studied the use of PINNs for the solution of 1D plasma edge transport equations in fusion devices. Firstly, the coupled plasma transport equations namely the continuity, momentum and energy equation were derived. These equations were reduced to the 1D case along magnetic field lines. Non-linearities in the heat conduction and viscosity terms occurred.
Afterward, general neural networks were introduced. The basic structure and functionality of a multilayer perceptron, including the feedforward algorithm and the concept of loss, were described. The different activation functions sigmoid, ReLU and tanh were compared and their particular importance by evaluating every layer's output non-linearly was stated. The reverse mode of AD was explained. Gradient descent and the improved optimization algorithms momentum, RMSprop and Adam were discussed. NNs were applied to receive PINNs. The loss function was build using the residuals of the coupled differential equations and its boundary conditions. At the end, the python package PyTorch and its benefits in NN training due to GPU usage and flexible AD implementation were pointed out.
This was then used to discuss the numerical results for 1D test cases based on the full system of coupled transport equations along magnetic field lines. The first three cases only considered the heat-balance equation but with increasing complexity. The first one assumed linear thermal conductivity and no convection. A convection term with constant velocity was added in the second problem. The third case contained the known non-linear heat conduction. For each of these problems, the PINNs, consisting of five layers of five respective neurons and the sigmoid activation function, approximated the derived analytical solutions perfectly. The performance of the Adam optimizer stood out. The losses converged noticeably, and overfitting never was seen. Nevertheless, a difficulty occurred when trying to ensure the boundary condition in the third problem. It was needed to increase the loss factor of the corresponding loss term to $10^3$. Additionally, multiple network initializations were necessary to guarantee positive temperatures, which were needed due to the non-linear evaluation in the heat conduction. These problems would get harder to solve when derivatives of the PINN's functions would be evaluated non-linearly. Ensuring monotonic increasing is not done in practice by adding an activation function or initializing several times.
The coupled convection dominant problem consisting of continuity and simplified momentum equation, and the problem with additional energy equation yielded perfect agreement with the derived analytical solutions without any difficulties. Both problems were formulated using fixed boundary values. The second case even clearly converged after less than 1000 epochs. However, the constant solutions were particularly simple to approximate.
The last two coupled problems of this thesis studied the transport equations with constant particle source and sheath boundary condition, which lead to greater difficulties. At first, only three equations were considered. Heat conductivity and the electron temperature were neglected. To ensure positive temperatures and densities, these outputs were additionally evaluated by the exponential function in the last layer. However, the PINN of five hidden layers with 50 respective neurons and sigmoid activation function could not approximate the analytical solutions satisfactorily. Replacing sigmoid by tanh lead to significantly better results. But still the ion temperature did not match completely, while density and velocity did. The last coupled problem including non-linear conduction term lead to similar results. The sigmoid function failed and tanh did better, but far from perfect. Only the expected shapes were recognizable. Interestingly, the loss curve of tanh

did not look more promising than the one of sigmoid. Additionally, the results were not as reproducible as before. Many local minima were entered, which would be hard to spot when not having access to analytical solutions. The main problem seems to be the sheath boundary condition, since no fixed values are given but only dependencies between the functions at the boundary. Increasing the number of grid points to 5000, or multiplying the strongly non-linear thermal heat flux BCs by 100 or 1000 did not change the outcome. Even the advanced method of gPINNs and the self-scalable tanh activation function did not yield better results.
In conclusion, PINNs for solving the plasma transport equations can be implemented easily due to flexible packages like PyTorch. However, there is still the need of further tests and new strategies to overcome the remaining problems like the running time compared to finite differences, elements and volumes. Additionally, the cases of strongly coupled equations with non-linearities and particularly the problem of the strongly non-linear sheath boundary conditions has to be addressed in the future. A possible approach might be further linearization by using one NN for the density, velocity and temperature, respectively. These individual NNs would then be optimized one after the other iteratively.

# A Appendix

## A.1 Higher dimensional integration by parts

As explained by Griffiths in [37, p. 37], the product rule for the divergence of the product of a scalar field $f$ and vector field $\mathbf{A}$

$$\nabla(f \cdot \mathbf{A}) = f \cdot (\nabla \mathbf{A}) + \mathbf{A} \cdot (\nabla f) \ ,$$

can be used to receive higher dimensional integration by parts. Integrating the identity over a volume $V$ leads to

$$\int_V \nabla(f \cdot \mathbf{A}) \ d^3r = \int_V f \cdot (\nabla \mathbf{A}) \ d^3r + \int_V \mathbf{A} \cdot (\nabla f) \ d^3r \ .$$

The integral on the left can be expressed as an integral over the boundary $\partial V$ by Gauss's theorem

$$\int_V \nabla(f \cdot \mathbf{A}) \ d^3r = \oint_{\partial V} f \cdot \mathbf{A} \ d^3r \ . \tag{48}$$

Inserting this and rearranging the equation yields

$$\int_V f \cdot (\nabla \mathbf{A}) \ d^3r = \oint_{\partial V} f \cdot \mathbf{A} \ d^3r - \int_V \mathbf{A} \cdot (\nabla f) \ d^3r \ . \tag{49}$$

## A.2 Dot product rule for divergence

In [37, p. 21] Griffiths stated that the divergence of a dot product of two vector fields $\mathbf{A}$ and $\mathbf{B}$ can be written as

$$\nabla(\mathbf{A} \cdot \mathbf{B}) = (\mathbf{A} \cdot \nabla) \cdot \mathbf{B} + (\mathbf{B} \cdot \nabla) \cdot \mathbf{A} + \mathbf{A} \times (\nabla \times \mathbf{B}) + \mathbf{B} \times (\nabla \times \mathbf{A}) \ .$$

Therefore $\mathbf{B} = \mathbf{A}$ yields

$$\nabla \left(A^2\right) = 2 \cdot (\mathbf{A} \cdot \nabla) \cdot \mathbf{A} + 2 \cdot \mathbf{A} \times (\nabla \times \mathbf{A}) \ .$$

Hence,

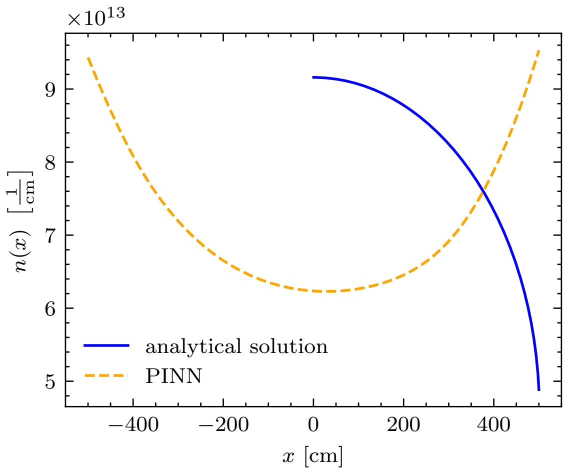
$$(\mathbf{A} \cdot \nabla) \cdot \mathbf{A} = \frac{1}{2} \cdot \nabla \left(A^2\right) - \mathbf{A} \times (\nabla \times \mathbf{A})$$

and multiplying both sides by $\mathbf{A}$ leads to the scalar equation

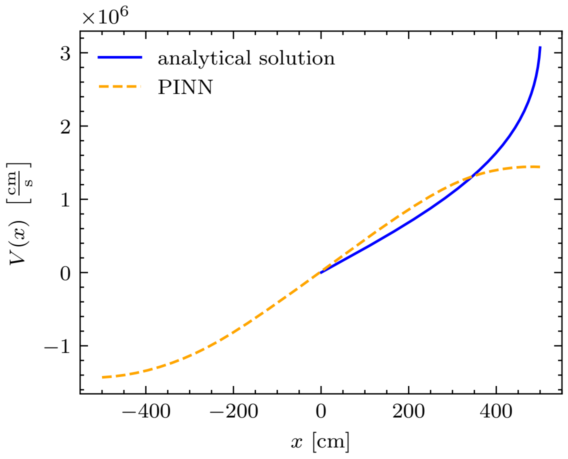
$$[(\mathbf{A} \cdot \nabla) \cdot \mathbf{A}] \cdot \mathbf{A} = \frac{1}{2} \mathbf{A} \cdot \nabla \left(A^2\right) \tag{50}$$

because $[\mathbf{A} \times (\nabla \times \mathbf{A})] \cdot \mathbf{A} = 0$ .

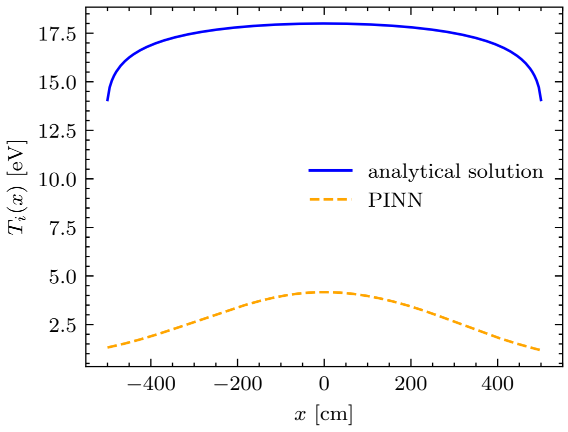
## A.3 Auxiliary sigmoid results



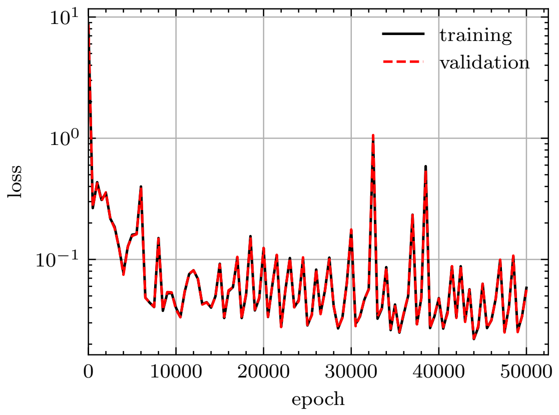
**(a)**



**(b)**



**(c)**



**(d)**

**Figure 21:** Numerical results of problem (47) solved by the PINN with sigmoid activation function, and analytical solutions (43) and (44). (a) density $n(x)$, (b) velocity $V(x)$, (c) ion temperature $T_i(x)$, (d) training and validation losses.

## References

[1] U. Stroth. *Plasmaphysik*. Springer Spektrum Berlin, Heidelberg, 2017.

[2] Leibniz Institute for Plasma Science and Technology (INP). *Research*. URL: `https://www.inp-greifswald.de/en/research/` (visited on 05/08/2023).

[3] Max Planck Institute for Plasma Physics. *Wendelstein 7-X*. 2023. URL: `https://www.ipp.mpg.de/w7x` (visited on 05/08/2023).

[4] S.I. Braginskii. 'Transport Processes in a Plasma'. In: *Reviews of Plasma Physics* 1 (1965). Ed. by Acad.M.A. Leontovich.

[5] A. Szoke and E. Brooks. *The Boltzmann equation in the difference formulation*. URL: `https://www.osti.gov/servlets/purl/1184176` (visited on 05/08/2023).

[6] Alex Runox. private communication. Max Planck Institute for Plasma Physics, Greifswald.

[7] E Havlívcková et al. 'Steady-state and time-dependent modelling of parallel transport in the scrape-off layer'. In: *Plasma Physics and Controlled Fusion* 53.6 (2011). DOI: `10.1088/0741-3335/53/6/065004`.

[8] Kahnfeld D. et al. 'Solutioin of Poisson's Equation in Electrostatic Particle-on-cell Simulation'. In: *Plasma Physics and Technology Journal* 3 (2016). DOI: `10.14311/ppt.2016.2.66`.

[9] R. Schneider et al. 'B2-solps5.0: SOL transport code with drifts and currents'. In: *Contributions to Plasma Physics* 40.3-4 (2000). DOI: `https://doi.org/10.1002/1521-3986(200006)40:3/4<328::AID-CTPP328>3.0.CO;2-Q`.

[10] Y Feng, F Sardei and J Kisslinger. '3D fluid modelling of the edge plasma by means of a Monte Carlo technique'. In: *Journal of Nuclear Materials* 266-269 (1999). DOI: `10.1016/S0022-3115(98)00844-7`.

[11] Ryoko Tatsumi. 'Development of a New Lagrange-Monte Carlo Scheme for Fluid Modeling of SOL/Divertor Plasmas'. MA thesis. Keio University, 2017.

[12] I. Neutelings. *Neural networks*. URL: `https://tikz.net/neural_networks/` (visited on 05/08/2023).

[13] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: `http://neuralnetworksanddeeplearning.com/index.html` (visited on 05/08/2023).

[14] PyTorch Contributors. *TORCH.NN.INIT*. URL: `https://pytorch.org/docs/stable/nn.init.html` (visited on 05/08/2023).

[15] Jürgen Schmidhuber. 'Deep learning in neural networks: An overview'. In: *Neural Networks* 61 (2015). URL: `https://www.sciencedirect.com/science/article/pii/S0893608014002135`.

[16] Aston Zhang et al. 'Dive into Deep Learning'. In: *CoRR* abs/2106.11342 (2021). arXiv: `2106.11342`.

[17] Bruno Després. *Neural Networks and Numerical Analysis*. Berlin, Boston: De Gruyter, 2022. DOI: `doi:10.1515/9783110783186`.

[18] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[19] Atilim Gunes Baydin et al. ‘Automatic Differentiation in Machine Learning: a Survey’. In: *Journal of Machine Learning Research* 18.153 (2018). URL: http://jmlr.org/papers/v18/17-468.html.

[20] Justin Domke. *Statistical Machine Learning.* Automatic Differentiation and Neural Networks. 2010. URL: https://people.cs.umass.edu/~domke/courses/sml2010/07autodiff_nnets.pdf (visited on 05/08/2023).

[21] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction.* MIT Press, 2022. URL: https://probml.github.io/pml-book/book1.html.

[22] T. Lee, G. Gasswint and E. Henning. *Momentum.* URL: https://optimization.cbe.cornell.edu/index.php?title=Momentum (visited on 05/08/2023).

[23] Jason Huang. *RMSProp.* URL: https://optimization.cbe.cornell.edu/index.php?title=RMSProp (visited on 05/08/2023).

[24] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG].

[25] PyTorch Contributors. *TORCH.OPTIM.* URL: https://pytorch.org/docs/stable/optim.html (visited on 05/08/2023).

[26] Ilya Loshchilov and Frank Hutter. ‘Fixing Weight Decay Regularization in Adam’. In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101.

[27] Yeonjong Shin. ‘On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs’. In: *Communications in Computational Physics* 28.5 (2020). DOI: 10.4208/cicp.oa-2020-0193.

[28] Siddhartha Mishra and Roberto Molinaro. ‘Estimates on the generalization error of physics-informed neural networks for approximating PDEs’. In: *IMA Journal of Numerical Analysis* 43 (2022). DOI: 10.1093/imanum/drab093.

[29] Paweł Maczuga and Maciej Paszyński. ‘Influence of Activation Functions on the Convergence of Physics-Informed Neural Networks for 1D Wave Equation’. In: *Computational Science – ICCS 2023.* 2023. DOI: 10.1007/978-3-031-35995-8_6.

[30] PyTorch Contributors. *PyTorch.* URL: https://github.com/pytorch/pytorch (visited on 05/08/2023).

[31] Ryan O’Connor. *PyTorch vs TensorFlow in 2023.* URL: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/ (visited on 05/08/2023).

[32] Matti Fensch. *s-mafens/bachelor-thesis-pinns.* URL: https://git.rz.uni-greifswald.de/s-mafens/bachelor-thesis-pinns (visited on 05/08/2023).

[33] Raghav Gnanasambandam et al. *Self-scalable Tanh (Stan): Faster Convergence and Better Generalization in Physics-informed Neural Networks.* 2022. arXiv: 2204.12589 [cs.LG].

[34] Panagiotis Antoniadis. *Activation Functions: Sigmoid vs Tanh.* URL: https://www.baeldung.com/cs/sigmoid-vs-tanh-functions/ (visited on 31/07/2023).

[35] Jeremy Yu et al. ‘Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems’. In: *Computer Methods in Applied Mechanics and Engineering* 393 (2022), p. 114823. DOI: 10.1016/j.cma.2022.114823.

[36] Ralf Schneider. private communication. Institute of Physics, University of Greifswald.

[37] D.J. Griffiths. *Introduction to Electrodynamics.* Cambridge University Press, 2018.

# Acknowledgement

I want to thank my supervisor Prof. Dr. Ralf Schneider for letting me chose a topic that really fits my interests and for guiding me through the whole thesis. In particular, I am very grateful for his eagerness to help at any time. Additionally, I would like to thank Prof. Dr. Peter Manz for being my secondary examiner.
Finally, I want to express my sincere gratitude to my family and friends for their great support throughout my bachelor years.