Karlsruher Institut für Technologie

Master Thesis

# Simulation of lithium ions batteries using physics informed neural networks

Nilo Schwencke

September 8. 2023

Supervisors: Dr. Mehdi Elasmi and Prof. Willy Dörfler

Fakultät für Mathematik

Karlsruher Institut für Technologie

# Acknowledgements

I would like to thank Professor Willy Dörfler for the supervision of this master thesis and the help he could provide to me. I would also like to specially thank Doctor Mehdi Elasmi for the excellent supervision and his precious help, as well as the numerous discussions we could have on PINNs and beyond, looking forward to to pursuing our work together. Thank you also to the TAU team of the LISN laboratory in the Paris-Saclay University for hosting me during the writing of this manuscript, and for lending me their computation servers so that I could produce the experimental results reported here. I look forward to joining them for my PhD. I also wish to thank the Baden-Württemberg High Performance Computing service, which enabled me to carry out my experimental research. Finally, I would like to express my gratitude to my family and closest friends, who have made my life so much easier in recent weeks. I am particularly thankful to my mother Alheli, my partner Julie and my very good friends Maria and Alicia.

This master thesis is dedicated to my two grandparents, Maria-Nieve and Jaime, who instilled in me the love of science from an early age.

# Contents

# List of Abbreviations

(S)GD (Stochastic) Gradient Descent

BMS Battery Management System

DAE Differential-Algebraic system of Equations

FVM Finite Volume Methods

LHS Latin Hypercube Sampling

MAE Mean Absolute Error

MLP Multiple Layers Perceptron

MRE Mean Relative Error

MSE Mean Squared Error

MSRE Mean Squared Relative Error

NN Neural Network

ODE Ordinary Differential Equation

P2D Pseudo-Two-Dimensional model for lithium ions cells

PDE Partial Differential Equation

PINN Physics Informed Neural Network

# Abstract

In this work, we present an approach to approximating a solution to the Pseudo-Two-dimensional (P2D) electrochemical model for lithium ions cells. In doing so, we will use Physics-Informed Neural Networks (PINNs). PINNs is a training technique for artificial neural networks, which takes into account the PDE of interest. To do so, the PDE is evaluated by the means of automatic differentiation mechanism in selected discrete points of the domain, before being inserted inside the loss function, which in turn can be used to train the neural network by back-propagation. We show that the approach of using PINNs to approximate the solution of the P2D model is valid by exhibiting promising results, paving the way to the use of hybrid methods (both empirical and model based) in Battery Management Systems.



# Kurzfassung

In dieser Arbeit stellen wir einen Ansatz zur Annäherung an eine Lösung für das Pseudo-Zweidimensionale (P2D) elektrochemische Modell für Lithium-Ionen-Zellen vor. Dabei werden wir Physics-Informed Neural Networks (PINNs) verwenden. PINNs ist eine Trainingstechnik für künstliche neuronale Netze, die die PDE von Interesse berücksichtigt. Zu diesem Zweck wird die PDE über automatisches Differenzierenmechanismus in ausgewählten diskreten Punkten der Domäne berechnet, bevor sie in die Verlustfunktion eingefügt wird, die wiederum zum Trainieren des neuronalen Netzes durch Back-Propagation verwendet werden kann. Wir zeigen, dass der Ansatz der Verwendung von PINNs zur Annäherung an die Lösung des P2D-Modells gültig ist, indem wir vielversprechende Ergebnisse zeigen, die den Weg für den Einsatz von hybriden Methoden (sowohl empirisch als auch modellbasiert) in Batteriemanagementsystemen ebnen.

# 1. Introduction

Against the backdrop of the need to transform the economy in order to achieve carbon neutrality, and in particular the growing popularity of electric vehicles, lithium-ion batteries are proving to be essential energy storage allies. Nevertheless, major challenges remain in terms of predictability and optimal control of the behavior of these technologies. In addition to materials research, a major part of battery research is thus focused on the designation of algorithmic monitors and controllers, known as Battery Management Systems (BMS). More specifically (See et al., 2022; Ramkumar et al., 2022), a BMS measures a number of characteristic quantities of the battery (temperature, state of charge, state of health, *etc.*) from which it will control the battery's operation (charge balance between cells, applied charge/discharge currents, *etc.*) in order to best satisfy the user's needs, while striving to guarantee safety of use and to preserve the battery as far as possible from the various problems it may face (overheating, calendar or cycling aging, *etc.*). Beyond hardware controllers and sensors, different algorithm approaches exist, some relying on completely empirical models, using the history of data produced by the battery to predict its future behavior, others using simplified physical models to assess the collected data and extract useful information.

In this master thesis, we propose to pave a first step into providing a unified approach, drawing on the benefits of both empirical and model based approaches via the use of deep neural networks (LeCun et al., 2015), combined with a set of promising training techniques, the so called "Physics-Informed Neural Networks" (PINNs : Raissi et al. 2019; Karniadakis et al. 2021).

In Section 2, we describe the Pseudo-Two-Dimensional (P2D) equations, modelling the electrochemical reactions taking place inside inside a lithium ions battery Cell, by mentioning the main physical laws from which these equations are derived.

In Section 3, we introduce Neural Networks (NNs) and Physics Informed Neural Networks (PINNs), first in an abstract mathematical framework, before deriving an approximating framework, and mentioning some algorithmic and practical insights.

In Section 4, we depict an effective implementation of the PINN framework applied to the P2D model, and show that the solution is correctly approximated with Dirichlet boundary conditions compared to the baseline from Torchio et al. (2016). We then show that solutions with actual Neumann and interface's boundary conditions do not work as such, even for short timescales, indicating that new research directions need to be investigated.

# 2. The Pseudo-Two-Dimensional (P2D) model of lithium ions batteries

In this section, we introduce the physical model that will be at the heart of this work. It models the behavior of smallest production unit inside a lithium ions battery, the cell. We start by briefly reviewing the working principles of lithium ions batteries.

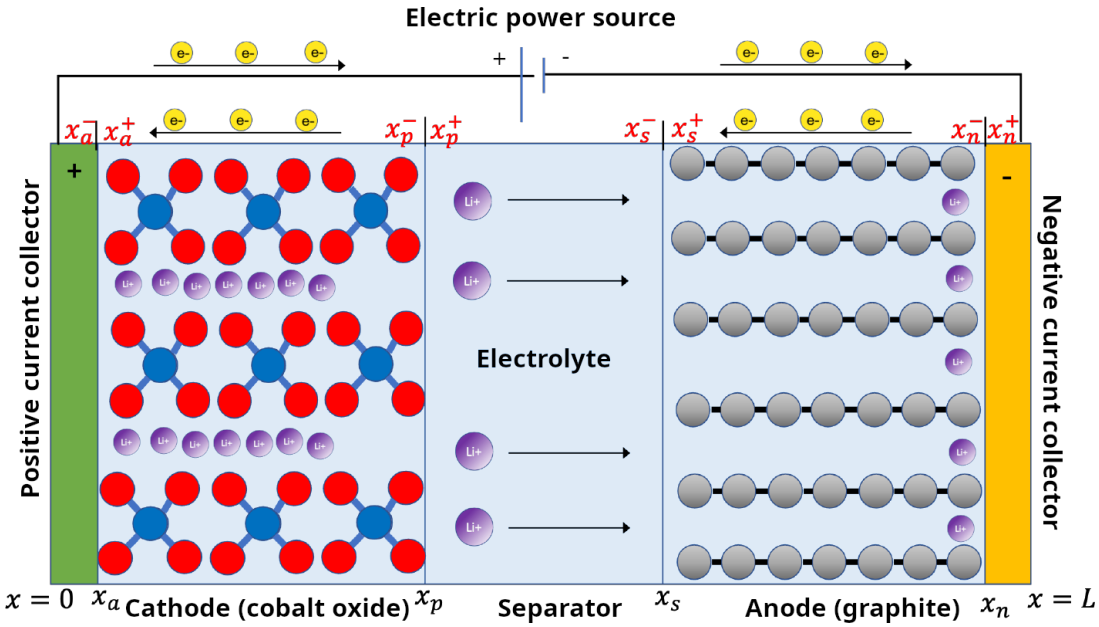
## 2.1. Working principles of a lithium ions battery

A lithium ions battery is a device that has the double ability to transform chemical energy into electrical energy and vice versa. When working in the forward direction, *i.e.* from chemical to electrical energy, the device is acting as an electrical generator and can therefore be used to power electric and electronic devices, as cell-phones, computers or electrical vehicles, until its stored chemical energy is exhausted. When working in the backward direction, *i.e.* from electrical to chemical energy, the device is said to be "charging", *i.e.* it is restoring its chemical energy and thus its capacity to act as an electrical generator in the future. This double ability provides an indirect way to bring electric power to systems that cannot be connected to conventional electrical distribution networks, as well as to smooth out supply in a conventional distribution network, which can fluctuate greatly, especially when the share of renewable energies increases, making lithium batteries a key player in the energy transition.

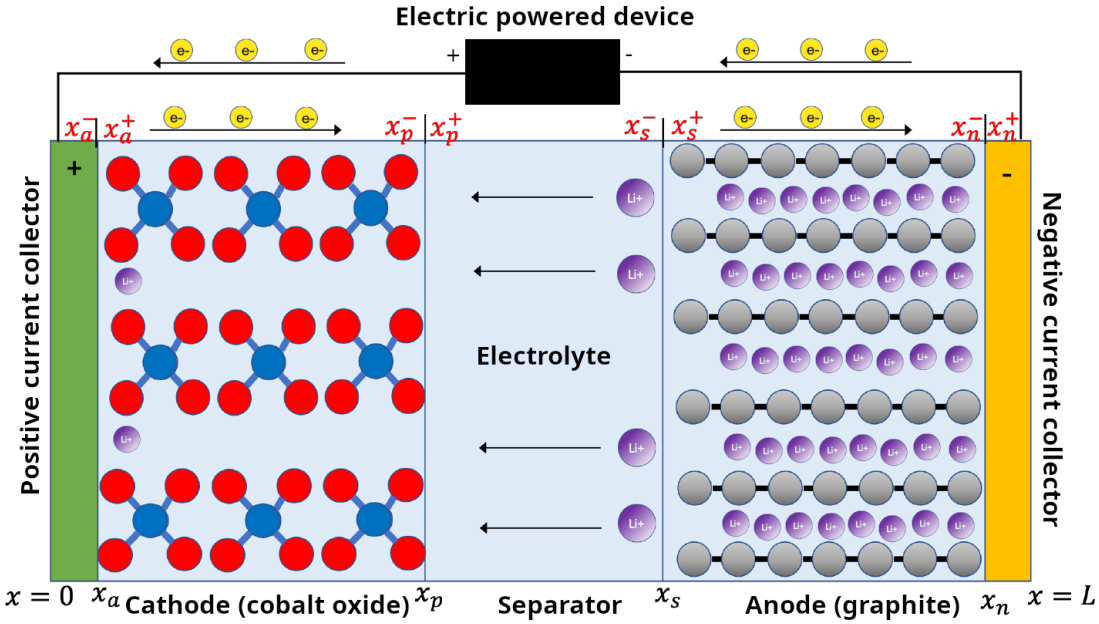
To be more specific, a lithium ions battery is a pack, consisting of several interconnected modules, themselves made up of several cells connected in series, which are the unit within which electrochemical reactions take place. Each cell consists of an anode (negative electrode), usually a graphite matrix, and a cathode (positive electrode), usually a lithiated transition metal oxide (cobalt dioxide or manganese), all immersed in a lithium salt electrolyte dissolved in an organic solvent and separated by a separator to prevent short circuits. The whole is confined by a membrane, except at the ends of the anode and cathode, which are each confined by a piece of metal known as a current collector, to allow the passage of electrons.

During discharge, lithium ions are extracted from the graphite matrix and migrate towards the lithiated metal oxide to which they have a greater affinity, generating a potential difference that can be exploited. Figure 1b summarizes this reaction. Under load, lithium ions are released by the metal oxide under the effect of the applied current, to migrate towards the graphite matrix. Figure 1a summarizes this reaction. For a more complete introduction, one could refer to Reddy (2010).

We will see in the next section, how those phenomena can be modeled, notably through electrochemical equations.




(a) **Charge reaction:** driven by the electrons (e-) flow generated by the electric power source, the lithium ions (Li+) are extracted from the cobalt oxide at the cathode by oxidation and migrate through the electrolyte to the anode where they intercalate in the graphite matrix by reduction.




(b) **Discharge reaction:** driven by the flow of electrons (e-) generated by connecting the electric powered device, the lithium ions (Li+) are extracted from the graphite matrix at the anode by oxidation and migrate through the electrolyte to the cathode where they intercalate in the cobalt oxide at the cathode by reduction.

Figure 1: Working principles of a lithium ions Cell (adapted from Han et al. 2021).

## 2.2. Electrochemical Modeling

Electrochemical models describe the electro-chemical phenomena that occur in each cell of a battery. The need of simulating the battery behavior is of paramount importance, in order to control and to monitor these systems online. Therefore reliable, fast and accurate

real-time simulation are imperative. The model that we will study in the context of this work consist of a system of partial differential equations coupled according to different physical principles, such as Fick's laws, the Buttler–Volmer and Nernst equations. The "Pseudo-Two-Dimensional" (P2D) model introduced by Doyle et al. (1993) and based on porous electrode theory (Newman and Tiedemann, 1975; Newman and Balsara, 2021) is considered the most effective and accurate, and has been validated in numerous studies (Torchio et al., 2015; Zou et al., 2017; Methekar, 2018). This model considers interactions in the direction of cell assembly only (negative current collector, anode, separator, cathode, positive current collector), neglecting radial interactions. Nevertheless the model adds a pseudo-second directional variable, which accounts for the radius of spherical particles, along which intercalation and extraction of lithium ions take place, those spherical particles being a mathematically idealized assimilation of the porous materials at cathode and anode, arranged on the whole $x$ axis. Figure 2 illustrates the P2D simplification.

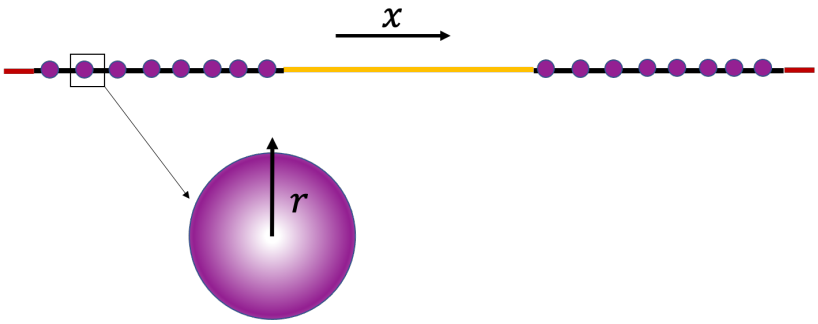



Figure 2: P2D simplification scheme of Figure 1 (taken from Han et al. 2021). Interactions in the cell are considered along $x$ axis only, positive current collector section being represented by the red line on the left, followed by the cathode section, represented as the left black line with purple dots depicting the spherical particles, then followed by separator in yellow and anode in black with purple dots according to same convention as the cathode, and finally the negative current collector in red. The highlighted zoom portrays dots as particles assimilating the porous material in which lithium ions intercalations and extractions occur along the the pseudo-second directional variable $r$ which accounts for the radius of the particle.

Although highly accurate, this model is difficult to implement in application cases, as it requires the identification of over 50 parameters (Chen et al., 2021), some of which (lithium ions concentration in the electrolyte, diffusion coefficients, transfer coefficients, *etc.*) are not directly accessible through available measurements (voltage, current, *etc.*; see Urbain 2009). What's more, each modification of initial conditions or parameters induces a complete re-run of the simulation, which is computationally expensive and may be prohibitive for real-time use. To overcome those difficulties, simplified models of the P2D model have been proposed for real-time applications, notably the Single Particle Model (SPM; see Zhang et al. 2000) and Equivalent Circuit Models (ECM; see Hu et al. 2012).

Other purely empirical approaches are also proposed, using Kalman filters (Chang and Xiaoluo, 2011), fuzzy-logic (Singh and Reisner, 2002), or neural networks (Wang et al., 2022b). We thus think that combining empirical and model-based approaches through Physics Informed Neural Networks (PINNs) could be an interesting research direction,

what drives this work.

In the next section, we enumerate the main notations used, before carefully describing the equations.

## 2.3. Notations

In this section, we introduce the main notations and variables used in the equations presented thereafter.

### 2.3.1. Main notations

We bring first notations used to identify the sections of the cell, as referred to in equations thereafter:

*Notation* 2.1 (Cell sections). We will follow the convention of Han et al. (2021) depicted in Figure 1, to designate the five sections of the Cell:

| Notation | Designation |
|---|---|
| $a$ | positive current collector |
| $p$ | cathode |
| $s$ | separator |
| $n$ | anode |
| $z$ | negative current collector |

Table 1: Cell sections notations in the P2D model.

In particular, letters with subscript $i \in \{a, p, s, n, z\}$ will refer to variables in cell section $i$ (*i.e.* negative current collector for $a$, cathode for $p$, *etc.*), except for $x_i$ (see Notation 2.2).

We then expose notations used to identify the boundaries of cell's sections:

*Notation* 2.2 (Section's boundaries coordinates). We still follow the convention of Han et al. (2021) depicted in Figure 1, to designate locations of cell's sections boundaries, adding "$\hat{x}$" notations, introduced for convenience in order to easily describe sections domains:

| Notations | Designation |
|---|---|
| $x_0$, $\hat{x}_a$ | spatial coordinate of the interface between the exterior of the cell and the positive current collector ($a$) |
| $x_a$, $\hat{x}_p$ | spatial coordinate of the interface between the positive current collector ($a$) and the cathode ($p$) |
| $x_p$, $\hat{x}_s$ | spatial coordinate of the interface between the cathode ($p$) and the separator ($s$) |
| $x_s$, $\hat{x}_n$ | spatial coordinate of the interface between the separator ($s$) and the anode ($n$) |
| $x_n$, $\hat{x}_z$ | spatial coordinate of the interface between the anode ($n$) and the current collector ($z$) |
| $x_z$ | spatial coordinate of the interface between the negative current collector ($z$) and the exterior of the cell |

Table 2: Notations of the section's boundaries coordinates in the P2D model.

We finally introduce the notations used for space and time parameters. As stated in Section 2.2, the P2D model has two dimensional parameters (space parameter in the assembly direction and time) plus one pseudo-second spatial parameter in the direction of the radius of the assimilated spherical particles.

*Notation* 2.3 (Parameters). We follow again the convention of Han et al. (2021) depicted in Figure 1, to designate space and time parameters:

| Notation | Designation | Unit | Domain | Order of magnitude |
|---|---|---|---|---|
| $t$ | time parameter | s | $[0, T_{\max}]$ | $10^3$ s |
| $x$ | space parameter in the assembly direction | m | $[x_0, x_z]$ | $10^{-4}$ m |
| $r$ | pseudo-second directional parameter | m | $[0, R_p]$ | $10^{-6}$ m |

Table 3: Notations of space and time parameters in the P2D model.
$T_{\max} \in ]0, +\infty[$ is a given constant representing the scope of time in which the model is considered, $x_0, x_z$ are those defined in Notation 2.2, and $R_p \in ]0, +\infty[$ is a given constant representing the radius of the assimilated spherical particles.

We also introduce some mathematical notations concerning evaluations of functions at boundaries, which are useful to describe boundary conditions in the equations:

*Notation* 2.4 (Evaluations at boundaries). Given a differentiable scalar function $u : [\hat{x}_i, x_i] \times [0, T_{\max}] \to \mathbb{R}$ with $i \in \{a, p, s, n, z\}$ (*cf.* Notation 2.2), evaluations of $\partial_x u$ at boundaries according to assembly direction parameter $x$ (see Notation 2.3) will be denoted by the classical notation: for all $t \in [0, T_{\max}]$

$$\partial_x u(x,t)\Big|_{x=\hat{x}_i^+} := \lim_{\substack{x \to \hat{x}_i \\ x > \hat{x}_i}} \partial_x u(x,t), \qquad \partial_x u(x,t)\Big|_{x=x_i^-} := \lim_{\substack{x \to x_i \\ x < x_i}} \partial_x u(x,t).$$

Similarly, given a scalar function $u : [\hat{x}_i, x_i] \times [0, T_{\max}] \times [0, R_{p,i}] \to \mathbb{R}$ with $i \in \{p, n\}$, evaluations of $\partial_r u$ at boundaries according to pseudo-second directional parameter $r$ (see Notation 2.3) will be denoted by the Formula: for all $t \in [0, T_{\max}]$ and for all $x \in [\hat{x}_i, x_i]$

$$\partial_r u(x,t,r)\Big|_{r=0^+} := \lim_{\substack{r \to 0 \\ r > 0}} \partial_r u(x,t,r), \qquad \partial_r u(x,t,r)\Big|_{r=R_{p,i}^-} := \lim_{\substack{r \to R_{p,i} \\ r < R_{p,i}}} \partial_r u(x,t,r).$$

We now introduce the main variables of the P2D model in the next section.

### 2.3.2. Main variables

Five main scalar variables describe the P2D model: temperature, lithium ions concentration, both in the electrolyte and in the solid phases (assimilated to spherical particles in the P2D model) and the potential, both in the electrolyte and in the solid phases (seen as homogeneous in the whole particle). We depict their main properties in Table 4 below. Those quantities are not present in every section, for instance lithium ions concentration

in the solid phases is only present in the cathode and anode. Therefore we carefully sum up the section in which each variable is present:

| Notation | Designation | Unit | Sections | Function of | Range |
|---|---|---|---|---|---|
| $T$ | Temperature | K | a,p,s,n,z | $x, t$ | $[0, +\infty)$ |
| $C_e$ | Lithium concentration in the electrolyte | $\text{mol}\cdot\text{m}^{-3}$ | p,s,n | $x, t$ | $[0, +\infty)$ |
| $C_s$ | Lithium concentration in the solid phases | $\text{mol}\cdot\text{m}^{-3}$ | p,n | $x, t, r$ | $[0, +\infty)$ |
| $\phi_e$ | Electrolyte potential | V | p,s,n | $x, t$ | $\mathbb{R}$ |
| $\phi_s$ | Solid phases potential | V | p,n | $x, t$ | $\mathbb{R}$ |

Table 4: Main variables of the P2D model.

Following Subramanian et al. (2005), we also introduce the average lithium concentration in the solid phases and the surface lithium concentration in the solid phases in order to implement the simplified P2D model (*cf.* Section 2.4.4). This is summed up in Table 5:

| Notation | Designation | Unit | Function of | Range |
|---|---|---|---|---|
| $C_s^*$ | Surface lithium concentration in the solid phases | $\text{mol}\cdot\text{m}^{-3}$ | $x, t$ | $[0, +\infty)$ |
| $\overline{C_s}$ | Average lithium concentration in the solid phases | $\text{mol}\cdot\text{m}^{-3}$ | $x, t$ | $[0, +\infty)$ |

Table 5: Supplementary variables for the simplified P2D model.

*Notation* 2.5. In the following, for the sake of notation simplicity, we will skip the section subscript on each of the variables of Table 4. This should not lead to ambiguous notations, since the sections do not share common span in the $x$ variable.

In addition to the main variables above, we also introduce several variables of interest, computed from the solutions of the considered problem (*cf.* Section 2.4.11 for more details). This is summed up in Table 6:

| Notation | Designation | Unit | Function of | Range |
|---|---|---|---|---|
| $SoC$ | State of charge | % | $t$ | $[0, 100]$ |
| $V_o$ | Cell output voltage | V | $t$ | $\mathbb{R}$ |

Table 6: Auxiliary variables for the P2D model.

Finally, the system also depends on the applied current density which accounts for the current exchange of the cell with its exterior and drives the charging/discharging regime of the cell. Table 7 sums up its characteristics:

| Notation | Unit | Function of | Range |
|---|---|---|---|
| $I_{\text{app}}$ | $\text{A}\cdot\text{m}^{-2}$ | $t$ | $\mathbb{R}$ |

Table 7: Applied current density characteristics.

In the next section, we line secondary variables used in the P2D model.

### 2.3.3. Secondary variables

Built upon the main variables described in Section 2.3.2, many secondary variables are introduced and used in P2D model. Those are summarized in Table 8:

| Notation | Designation | Unit | Sections | Dependence [1] | Eq. [2] |
|---|---|---|---|---|---|
| $\theta$ | Normalized lithium concentration in the solid phases | – | p,n | $C_s$ | (2.21) |
| $U_{\text{ref}}$ | Open circuit reference voltage | V | p,n | $\theta(C_s)$ | (2.19) (2.20) |
| $\partial_T U\big\|_{T_{\text{ref}}}$ | Open circuit potential entropic variation | $\text{V}\cdot\text{K}^{-1}$ | p,n | $\theta(C_s)$ | (2.17) (2.18) |
| $U$ | Open circuit voltage | V | p,n | $U_{\text{ref}}(C_s)$, $\partial_T U\big\|_{T_{\text{ref}}}(C_s)$, $T$ | (2.16) |
| $\eta$ | Surface overpotential | V | p,n | $\phi_s$, $\phi_e, U(C_s, T)$ | (2.15) |
| $k_{\text{eff}}$ | Effective reaction rate | $\text{m}^{2.5}\cdot\text{mol}^{-0.5}\cdot\text{s}^{-1}$ | p,n | $T$ | (2.25) |
| $j$ | Solid phases ionic flux | $\text{mol}\cdot\text{m}^{-2}\cdot\text{s}^{-1}$ | p,n | $k_{\text{eff}}(T)$, $C_e$, $C_s$, $T$, $\eta(\phi_e, \phi_s, C_s, T)$ | (2.14) |
| $\kappa_{\text{eff}}$ | Effective electrolyte conductivity | $\text{S}\cdot\text{m}^{-1}$ | p,s,n | $T$, $C_e$ | (2.24) |
| $Q_{\text{ohm}}$ | Ohmic heat source term | $\text{W}\cdot\text{m}^{-3}$ | p,s,n | $\partial_x\phi_s$, $\kappa_{\text{eff}}(T, C_e)$, $\partial_x\phi_e$, $C_e$, $\partial_x C_e$, $T$ | (2.26) (2.27) |
| $Q_{\text{rev}}$ | Reversible heat source term | $\text{W}\cdot\text{m}^{-3}$ | p,n | $j(T, C_e, C_s, \phi_e, \phi_s)$, $T$, $\partial_T U\big\|_{T_{\text{ref}}}(C_s)$ | (2.29) |
| $Q_{\text{rxn}}$ | Reaction heat source term | $\text{W}\cdot\text{m}^{-3}$ | p,n | $j(T, C_e, C_s, \phi_e, \phi_s)$, $\eta(\phi_e, \phi_s, C_s, T)$ | (2.28) |
| $D_{\text{eff}}$ | Effective electrolyte diffusion coefficient | $\text{m}^2\cdot\text{s}^{-1}$ | p,s,n | $T$, $C_e$ | (2.22) |
| $D^s_{\text{eff}}$ | Effective solid phases diffusion coefficient | $\text{m}^2\cdot\text{s}^{-1}$ | p,n | $T$ | (2.23) |

Table 8: Secondary variables of the P2D model.

[1] *i.e.* for some secondary variable $f$ that depends on some main variable $u$ and some secondary variable $g$, itself dependent of main variables $u, v$ we write the "dependence" of $f$ as: $u, g(u, v)$.
[2] Defining equation

In the next section, we bring out the constants used in the P2D model.

### 2.3.4. Constants

In addition to the main variables of Section 2.3.2 and secondary variables of Section 2.3.3, the P2D model uses several constants summarized in Table 9:

| Notation | Designation | Unit | Sections |
|---|---|---|---|
| $C_s^{\max}$ | Maximum solid phases concentration | $\mathrm{mol{\cdot}m^{-3}}$ | p,n |
| $D$ | Electrolyte diffusivity | $\mathrm{m^2{\cdot}s^{-1}}$ | p,s,n |
| $D^s$ | Solid phases diffusivity | $\mathrm{m^2{\cdot}s^{-1}}$ | p,n |
| $k$ | Reaction rate constant | $\mathrm{m^{2.5}{\cdot}mol^{-0.5}{\cdot}s^{-1}}$ | p,n |
| $R_p$ | Particle radius | m | p,n |
| $\rho$ | Density | m | a,p,s,n,z |
| $C_p$ | Specific heat | $\mathrm{J{\cdot}kg^{-1}{\cdot}K^{-1}}$ | a,p,s,n,z |
| $\lambda$ | Thermal conductivity | $\mathrm{W{\cdot}m^{-1}{\cdot}K^{-1}}$ | a,p,s,n,z |
| $\sigma$ | Solid phases conductivity | $\mathrm{S{\cdot}m^{-1}}$ | a,p,n,z |
| $\epsilon$ | Porosity | – | p,s,n |
| $a$ | Particle surface area to volume | $\mathrm{m^2{\cdot}m^{-3}}$ | p,n |
| $E_a^{D^s}$ | Solid phases diffusion activation energy | $\mathrm{J{\cdot}mol^{-1}}$ | p,n |
| $E_a^k$ | Reaction constant activation energy | $\mathrm{J{\cdot}mol^{-1}}$ | p,n |
| brugg | Bruggeman's coefficient | – | p,s,n |
| $\epsilon_f$ | Filler fraction | – | p,n |
| $t_+$ | Transference number | – | p,s,n |
| $T_{\mathrm{ref}}$ | Reference (*i.e.* ambient) temperature | K | -[3] |
| $h$ | Heat exchange coefficient | $\mathrm{W{\cdot}m^{-2}{\cdot}K^{-1}}$ | a,z |

Table 9: Constants of the P2D model

The P2D model also cal on some universal constants depicted in Table 10.

| Notation | Designation | Unit | Value |
|---|---|---|---|
| $F$ | Faraday's constant | $\mathrm{C{\cdot}mol^{-1}}$ | 96 485 |
| $R$ | Universal gas constant | $\mathrm{J{\cdot}mol^{-1}{\cdot}K^{-1}}$ | 8.314 472 |

Table 10: Universal constants used by the P2D model

Finally, the P2D model put in for convenience some constants defined from those stated in Table 9:

[3] independent

| Notation | Designation | Unit | Sections | Definition |
|---|---|---|---|---|
| $\Upsilon$ | Unspecified | $\mathrm{J{\cdot}C^{-1}{\cdot}K^{-1}}$ | -[4] | $\frac{2(1-t_+)R}{F}$ (2.1) |
| $\sigma_{\text{eff}}$ | Effective solid phases conductivity | $\mathrm{S{\cdot}m^{-1}}$ | p,n | $\sigma_i(1-\epsilon_i-\epsilon_{f,i}) \quad i \in \{p,n\}$ (2.2) |

Table 11: Derived constants of the P2D model

In the next section, we carefully report the equations of the P2D model.

## 2.4. Equations

In this section we describe the precise equations of the P2D model, which report the electrochemical phenomena taking place in a cell of lithium ions battery. A complete summary can be found in Appendix A.

### 2.4.1. Temperature

The temperature is modeled in all sections of the cell.

**Positive and negative current collectors:** In the current collector, temperature $T$ is modeled by diffusion with a source term given by Joule heating due to the passage of the applied current density $I_{\text{app}}$ through the current collectors: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned}\rho_i C_{p,i} \partial_t T(x,t) &= \partial_x \left[\lambda_i \partial_x T(x,t)\right] + \frac{I_{\text{app}}^2(t)}{\sigma_{\text{eff},i}} \\ &= \lambda_i \partial_{x^2} T(x,t) + \frac{I_{\text{app}}^2(t)}{\sigma_{\text{eff},i}} \qquad i \in \{a,z\},\end{aligned} \tag{2.3a}$$

where $\sigma_{\text{eff}}$ is the effective solid phases conductivity (*cf.* Table 11), and $\rho$, $C_p$ and $\lambda$ are respectively the density, specific heat and thermal conductivity (*cf.* Table 9). Boundary conditions on $x_0$ and $x_z$ consist in Robin boundary conditions given by Newton's law of cooling at the two ends of the battery: for all $t \in [0, T_{\max}]$

$$-\lambda_a \partial_t T(x,t)\Big|_{x=x_0^+} = h[T_{\text{ref}} - T(x_0,t)], \tag{2.3b}$$

$$-\lambda_z \partial_t T(x,t)\Big|_{x=x_z^-} = h[T(x_z,t) - T_{\text{ref}}], \tag{2.3c}$$

[4] independent

where $h$ is the heat exchange coefficient (*cf.* Table 9). Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$

$$T(x, 0) = T^{0,i}(x) \qquad i \in \{a, z\}, \tag{2.3d}$$

where for all $i \in \{a, z\}$, $T^{0,i} : [\hat{x}_i, x_i] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the temperature in the cell section.

**Cathode and anode** In the electrodes, temperature $T$ is modeled by diffusion with a source term coming from heat-source terms $Q_{\text{ohm}}$, $Q_{\text{rxn}}$ and $Q_{\text{rev}}$ (*cf.* Section 2.4.10): for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned} \rho_i C_{p,i} \partial_t T(x,t) &= \partial_x \left[\lambda_i \partial_x T(x,t)\right] + Q_{\text{ohm},i}(x,t) + Q_{\text{rxn},i}(x,t) + Q_{\text{rev},i}(x,t) \qquad (2.4a) \\ &= \lambda_i \partial_{x^2} T(x,t) + Q_{\text{ohm},i}(x,t) + Q_{\text{rxn},i}(x,t) + Q_{\text{rev},i}(x,t) \qquad i \in \{p, n\}, \end{aligned}$$

where $\rho$, $C_p$ and $\lambda$ are the density, specific heat and thermal conductivity respectively (*cf.* Table 9). Boundary conditions on $x_a$ and $x_n$ consist in interface's conditions enforcing flux continuity: for all $t \in [0, T_{\max}]$

$$-\lambda_a \partial_x T(x,t)\Big|_{x=x_a^-} = -\lambda_p \partial_x T(x,t)\Big|_{x=x_a^+}, \tag{2.4b}$$

$$-\lambda_n \partial_x T(x,t)\Big|_{x=x_n^-} = -\lambda_z \partial_x T(x,t)\Big|_{x=x_n^+}. \tag{2.4c}$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$

$$T(x, 0) = T^{0,i}(x) \qquad i \in \{p, n\}, \tag{2.4d}$$

where for all $i \in \{p, n\}$, $T^{0,i} : [\hat{x}_i, x_i] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the temperature in the cell section.

**Separator** In the separator the temperature $T$ is modeled by diffusion in the same way as for the electrodes, but with the heat-source term only equal to $Q_{\text{ohm}}$, since $Q_{\text{rxn}}$ and $Q_{\text{rev}}$ are zero due to the absence of solid phases in the separator and thus the absence of solid phases ionic flux $j_s$ (*cf.* Section 2.4.10) and thus omitted: for all $x \in [\hat{x}_s, x_s]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned} \rho_s C_{p,s} \partial_t T(x,t) &= \partial_x \left[\lambda_s \partial_x T(x,t)\right] + Q_{\text{ohm},s}(x,t) \qquad (2.5a) \\ &= \lambda_s \partial_{x^2} T(x,t) + Q_{\text{ohm},s}(x,t), \end{aligned}$$

where $\rho$, $C_p$ and $\lambda$ are the density, specific heat and thermal conductivity, respectively (*cf.* Table 9). Similarly, boundary conditions on $x_p$ and $x_s$ consist in interface's conditions enforcing flux continuity: for all $t \in [0, T_{\max}]$

$$-\lambda_p \partial_x T(x,t)\Big|_{x=x_p^-} = -\lambda_s \partial_x T(x,t)\Big|_{x=x_p^+}, \tag{2.5b}$$

$$-\lambda_s \partial_x T(x,t)\Big|_{x=x_s^-} = -\lambda_n \partial_x T(x,t)\Big|_{x=x_s^+}. \tag{2.5c}$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_s, x_s]$

$$T(x, 0) = T^{0,s}(x), \tag{2.5d}$$

where $T^{0,s} : [\hat{x}_s, x_s] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the temperature in the cell section.

### 2.4.2. Lithium ions accumulation in the electrolyte

**Cathode and anode** The accumulation of lithium ions in the electrolyte in the electrodes is modeled by diffusion of the lithium ions concentration in the electrolyte $C_e$ with a source term coming from the solid phases ionic flux $j$ (*cf.* Table 8): for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\epsilon_i \partial_t C_e(x,t) = \partial x \left[ D_{\text{eff},i}(x,t) \partial_x C_e(x,t) \right] + a_i (1 - t_+) j_i(x,t) \qquad i \in \{p, n\}, \tag{2.6a}$$

where $\epsilon$, $a$ and $t_+$ are the porosity, the particle surface area to volume and the transference number, respectively (*cf.* Table 9), and $D_{\text{eff}}$ is the effective electrolyte diffusion coefficient (*cf.* Table 8). The boundary conditions on $x_a$ and $x_n$ for $C_e$ are given by Neumann boundary conditions accounting for the absence of flux at electrodes extremities: for all $t \in [0, T_{\max}]$

$$\partial_x C_e(x,t) \Big|_{x = x_a^+, x_n^-} = 0. \tag{2.6b}$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$

$$C_e(x,0) = C_e^{0,i}(x) \qquad i \in \{p, n\}, \tag{2.6c}$$

where for all $i \in \{p, n\}$, $C_e^{0,i} : [\hat{x}_i, x_i] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the lithium ions concentration in the cell section.

**Separator** The accumulation of lithium ions in the electrolyte in the separator is modeled likewise by diffusion of $C_e$ with null source, since the solid phases ionic flux $j_s = 0$ (*cf.* Table 8) as there are no solid phases in the separator: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\epsilon_s \partial_t C_e(x,t) = \partial_x \left[ D_{\text{eff},s}(x,t) \partial_x C_e(x,t) \right], \tag{2.7a}$$

where $\epsilon$ is the porosity (*cf.* Table 9), and $D_{\text{eff}}$ is the effective electrolyte diffusion coefficient (*cf.* Table 8). Boundary conditions on $x_p$ and $x_s$ for $C_e$ are given by interface's conditions enforcing flux continuity: for all $t \in [0, T_{\max}]$

$$-D_{\text{eff},p}(x_p,t) \partial_x C_e(x,t) \Big|_{x = x_p^-} = -D_{\text{eff},s}(x_p,t) \partial_x C_e(x,t) \Big|_{x = x_p^+}, \tag{2.7b}$$

$$-D_{\text{eff},s}(x_s,t) \partial_x C_e(x,t) \Big|_{x = x_s^-} = -D_{\text{eff},n}(x_s,t) \partial_x C_e(x,t) \Big|_{x = x_s^+}. \tag{2.7c}$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_s, x_s]$

$$C_e(x,0) = C_e^{0,s}(x), \tag{2.7d}$$

where $C_e^{0,s} : [\hat{x}_s, x_s] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the lithium ions concentration in the cell section.

### 2.4.3. Lithium ions transportation

**Cathode and anode** Transportation of lithium ions in the electrodes is described by means of the Nernst–Einstein equation (Newman and Balsara, 2021, Eq. (11.45) p.239):

for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned} a_i F j_i(x,t) &= -\partial_x \left[\kappa_{\text{eff},i}(x,t)\partial_x \phi_e(x,t)\right] + \partial_x \left[\kappa_{\text{eff},i}(x,t)\Upsilon T(x,t)\partial_x \ln C_e(x,t)\right] \qquad (2.8a)\\ &= \partial_x \left(\kappa_{\text{eff},i}(x,t)\left[\Upsilon T(x,t)\frac{\partial_x C_e(x,t)}{C_e(x,t)} - \partial_x \phi_e(x,t)\right]\right) \qquad i \in \{p,n\}, \end{aligned}$$

where $a$ is the particle surface area to volume (*cf.* Table 9), $F$ is the Faraday's constant, $\kappa_{\text{eff}}$ is the effective electrolyte conductivity (*cf.* Table 8), and $\Upsilon$ is a convenient abbreviation defined in Table 11. Boundary conditions on $x_a$ and $x_n$ for $\phi_e$ are given respectively by Neumann boundary condition accounting for the absence of flux at cathode extremity: for all $t \in [0, T_{\max}]$

$$\partial_x \phi_e(x,t)\Big|_{x=x_a^+} = 0, \qquad (2.8b)$$

and Dirichlet boundary condition enforcing potential 0 reference at anode extremity: for all $t \in [0, T_{\max}]$

$$\phi_e(x_n, t) = 0. \qquad (2.8c)$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$

$$\phi_e(x,0) = \phi_e^{0,i}(x) \qquad i \in \{p,n\}, \qquad (2.8d)$$

where for all $i \in \{p,n\}$, $\phi_e^{0,i} : [\hat{x}_i, x_i] \to \mathbb{R}$ is a twice differentiable function representing the intial state of the electrolyte potential in the cell section.

**Separator** The transportation of lithium ions in the separator is modeled likewise by Ohm's and Kirchhoff's laws involving electrolyte potential $\phi_e$, temperature $T$ and lithium ions concentration in the electrolyte $C_e$ with null source, since the solid phases ionic flux $j_s = 0$, as there are no solid phases in the separator: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned} 0 &= -\partial_x \left[\kappa_{\text{eff},s}(x,t)\partial_x \phi_e(x,t)\right] + \partial_x \left[\kappa_{\text{eff},s}(x,t)\Upsilon T(x,t)\partial_x \ln C_e(x,t)\right] \qquad (2.9a)\\ &= \partial_x \left(\kappa_{\text{eff},s}(x,t)\left[\Upsilon T(x,t)\frac{\partial_x C_e(x,t)}{C_e(x,t)} - \partial_x \phi_e(x,t)\right]\right), \end{aligned}$$

where $\kappa_{\text{eff}}$ is the effective electrolyte conductivity (*cf.* Table 8), and $\Upsilon$ is a convenient abbreviation defined in Table 11. Boundary conditions on $x_p$ and $x_s$ for $\phi_e$ are given by interface's conditions enforcing flux continuity: for all $t \in [0, T_{\max}]$

$$-\kappa_{\text{eff},p}(x_p,t)\partial_x \phi_e(x,t)\Big|_{x=x_p^-} = -\kappa_{\text{eff},s}(x_p,t)\partial_x \phi_e(x,t)\Big|_{x=x_p^+}, \qquad (2.9b)$$

$$-\kappa_{\text{eff},s}(x_s,t)\partial_x \phi_e(x,t)\Big|_{x=x_s^-} = -\kappa_{\text{eff},n}(x_s,t)\partial_x \phi_e(x,t)\Big|_{x=x_s^+}. \qquad (2.9c)$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_s, x_s]$

$$\phi_e(x,0) = \phi_e^{0,s}(x), \qquad (2.9d)$$

where $\phi_e^{0,s} : [\hat{x}_s, x_s] \to \mathbb{R}$ is a twice differentiable function representing the intial state of the electrolyte potential in the cell section.

### 2.4.4. Lithium ions intercalation in the solid phases

The intercalation of Li-ions in the solid phases drives the charge and discharge of the cell. This is modeled by Fick's second law of diffusion with respect to the pseudo-second radial dimension $r$ of assimilated spherical particles on lithium ions concentration in the solid phases $C_s$ (Summerfield and Curtis, 2015): for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$, for all $r \in [0, R_{p,i}]$

$$\partial_t C_s(x,t,r) = \frac{1}{r^2}\partial_r\left[r^2 D_i^s \partial_r C_s(x,t,r)\right] \qquad i \in \{p,n\}, \tag{2.10a}$$

where $D^s$ is the solid phases diffusivity (*cf.* Table 9). Boundary conditions are modeled by Neumann boundary conditions accounting respectively for the absence of flux at $r = 0$ and equality with solid phases ionic flux $j$ (*cf.* Table 8) up to effective solid phases diffusion coefficient $D^s_{\text{eff},i}$ (*cf.* Table 8) at the surface of the spherical particles: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\partial_r C_s(x,t,r)\Big|_{r=0^+} = 0, \tag{2.10b}$$

$$\partial_r C_s(x,t,r)\Big|_{r=R_{p,i}^-} = -\frac{j_i(x,t)}{D^s_{\text{eff},i}(x,t)} \qquad i \in \{p,n\}. \tag{2.10c}$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$, for all $r \in [0, R_{p,i}]$

$$C_s(x,0,r) = C_s^{0,i}(x,r) \qquad i \in \{p,n\}, \tag{2.10d}$$

where for all $i \in \{p,n\}$, $C_s^{0,i} : [\hat{x}_i, x_i] \times [0, R_{p,i}] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the lithium ions concentration in the solid phases in the cell section.

**Model simplification** Following Subramanian et al. (2005), we will use a simplified model combining volume-averaging and polynomial approximation, which drops the pseudo-second radial dimension $r$ by replacing lithium ions concentration in the solid phases $C_s$ with the variables: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$C_s^*(x,t) := C_s(x,t,R_{p,i}), \tag{2.11a}$$

$$\overline{C_s}(x,t) := \frac{1}{R_{p,i}}\int_0^{R_{p,i}} C_s(x,t,r)\,dr \qquad i \in \{p,n\}, \tag{2.11b}$$

respectively, named surface lithium ions concentration in the solid phases and average lithium ions concentration in the solid phases governed by the equations: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\partial_t \overline{C_s}(x,t) = -3\frac{j_i(x,t)}{R_{p,i}} \qquad i \in \{p,n\}, \tag{2.12a}$$

$$C_s^*(x,t) - \overline{C_s}(x,t) = -\frac{R_{p,i}}{5D^s_{\text{eff},i}(x,t)} j_i(x,t) \qquad i \in \{p,n\}. \tag{2.12b}$$

We then remark that Equation (2.12b) can be rewritten: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\overline{C_s}(x,t) = \frac{R_{p,i}}{5D^s_{\text{eff},i}(x,t)} j_i(x,t) + C_s^*(x,t) \qquad i \in \{p,n\}.$$

Taking the partial derivative with respect to $t$ yields: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\partial_t \overline{C_s}(x,t) = \frac{R_{p,i}}{5} \frac{\partial_t j_i(x,t) D^s_{\text{eff},i}(x,t) - \partial_t D^s_{\text{eff},i}(x,t) j_i(x,t)}{\left(D^s_{\text{eff},i}(x,t)\right)^2} + \partial_t C^*_s(x,t) \qquad i \in \{p,n\}.$$

Injecting Equation (2.12a), we finally obtain: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\partial_t C^*_s(x,t) = -3\frac{j_i(x,t)}{R_{p,i}} - \frac{R_{p,i}}{5} \frac{\partial_t j_i(x,t) D^s_{\text{eff},i}(x,t) - \partial_t D^s_{\text{eff},i}(x,t) j_i(x,t)}{\left(D^s_{\text{eff},i}(x,t)\right)^2} \qquad i \in \{p,n\}, \tag{2.12c}$$

which is an equation on $C^*_s$ where $\overline{C_s}$ no longer appears. This means that we can dispense with variable $\overline{C_s}$ and replace Equations (2.12a) and (2.12b) with Equation (2.12c).

Finally, we also update the initial conditions at $t = 0$ (2.10d), to the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$

$$C^*_s(x,0) = C^{*,0,i}_s(x) \qquad i \in \{p,n\}, \tag{2.12d}$$

where for all $i \in \{p,n\}$, $C^{*,0,i}_s : [\hat{x}_i, x_i] \to [0, +\infty)$ is a twice differentiable function representing the intial state of the lithium ions concentration in the solid phases in the cell section.

### 2.4.5. Electrons motion

The motion of electrons in the electrodes is driven by the electric field which is related to the gradient of the solid phases potential $\partial_x \phi_s$. This is modeled by Ohm's law (Newman and Balsara, 2021, Eq. (1.13) p.8): for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned} a_i F j_i(x,t) &= \partial_x \left[\sigma_{\text{eff},i} \partial_x \phi_s(x,t)\right] \\ &= \sigma_{\text{eff},i} \partial_{x^2} \phi_s(x,t) \qquad i \in \{p,n\}, \end{aligned} \tag{2.13a}$$

where $a$ is the particle surface area to volume (*cf.* Table 9), $F$ is the Faraday's constant, $\sigma_{\text{eff}}$ is the effective solid phases conductivity (*cf.* Table 11). Boundary conditions are given by Neumann boundary conditions respectively accounting for the absence of flux on $x_p$ and $x_s$, and alignment of the applied current density $I_{\text{app}}(t)$ transferred by the current collectors with the electric field on $x_a$ and $x_n$: for all $t \in [0, T_{\max}]$

$$\sigma_{\text{eff},p} \partial_x \phi_s(x,t)\Big|_{x=x_p^-} = 0, \qquad \sigma_{\text{eff},n} \partial_x \phi_s(x,t)\Big|_{x=x_s^+} = 0, \tag{2.13b}$$

$$\sigma_{\text{eff},p} \partial_x \phi_s(x,t)\Big|_{x=x_a^+} = -I_{\text{app}}(t), \qquad \sigma_{\text{eff},n} \partial_x \phi_s(x,t)\Big|_{x=x_n^-} = -I_{\text{app}}(t). \tag{2.13c}$$

Finally the initial conditions at $t = 0$ are given by the Dirichlet boundary conditions: for all $x \in [\hat{x}_i, x_i]$

$$\phi_s(x,0) = \phi^{0,i}_s(x) \qquad i \in \{p,n\}, \tag{2.13d}$$

where for all $i \in \{p,n\}$, $\phi^{0,i}_s : [\hat{x}_i, x_i] \to \mathbb{R}$ is a twice differentiable function representing the intial state of the solid phases potential in the cell section.

### 2.4.6. Solid phases ionic flux

The solid phases ionic flux $j$ (*cf.* Table 8) is driven by the Butler–Volmer equation (Dickinson and Wain, 2020): for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$j_i(x,t) = 2k_{\text{eff}}(x,t)\sqrt{C_e(x,t)\left(C_{s,i}^{\max} - C_s(x,t,R_{p,i})\right)C_s(x,t,R_{p,i})} \times \sinh\left(\frac{0.5F}{RT(x,t)}\eta_i(x,t)\right) \qquad i \in \{p,n\}, \tag{2.14}$$

where $k_{\text{eff}}$ and $\eta$ are respectively the effective reaction rate and the overpotential (*cf.* Table 8), $C_e$, $C_s$, and $T$ are respectively, the lithium ions concentration in the electrolyte, the lithium ions concentration in the solid phases and the temperature (*cf.* Table 4), $F$ and $R$ are respectively the Faraday's constant and the universal gas constant, and finally $C_s^{\max}$ and $R_p$ are respectively the maximum solid phases concentration and the particle radius (*cf.* Table 9).

### 2.4.7. Surface overpotential

The overpotential refers to the magnitude of the potential drop caused by resistance to the passage of the current (Newman and Balsara, 2021, p.3). $\eta$ (*cf.* Table 8) is the one occurring at the surface of assimilated spherical particles in the electrode and is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\eta_i(x,t) = \phi_s(x,t) - \phi_e(x,t) - U_i(x,t) \qquad i \in \{p,n\}, \tag{2.15}$$

where $\phi_e$ and $\phi_s$ are respectively, the electrolyte potential and the solid phases potential (*cf.* Table 4), and $U$ is the open-circuit potential (*cf.* Table 8).

### 2.4.8. Open circuit potential

The open-circuit potential $U$ is modeled by an approximation of the Nernst equation accounting for temperature dependence (see Newman and Balsara 2021, Chapter 1 Section 1.2 and Chapter 2 Section 2.12 or Kumaresan et al. 2007): for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$U_i(x,t) = U_{\text{ref},i} + (T(x,t) - T_{\text{ref}})\partial_T U_i\big|_{T_{\text{ref}}}(x,t) \qquad i \in \{p,n\}, \tag{2.16}$$

where $T$ is the temperature (*cf.* Table 4), $T_{\text{ref}}$ is the reference (or ambient) temperature (*cf.* Table 9), and $U_{\text{ref}}$ and $\partial_T U\big|_{T_{\text{ref}}}$ are respectively the open circuit reference voltage and open circuit potential entropic variation (*cf.* Table 8). Those two last variables are fitted experimentally by polynomial based approximations: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\partial_T U_p|_{T_{\text{ref}}}(x,t) = -0.001\left(\frac{N_p(\theta_p(x,t))}{D_p(\theta_p(x,t))}\right), \tag{2.17}$$

with: for all $\theta \in [0,1]$

$$N_p(\theta) = 0.199521039 - 0.928373822\,\theta + 1.364550689000003\,\theta^2 - 0.6115448939999998\,\theta^3,$$
$$D_p(\theta) = 1 - 5.661479886999997\,\theta + 11.47636191\,\theta^2 - 9.82431213599998\,\theta^3 + 3.046755063\,\theta^4,$$

for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\partial_T U_n|_{T_{\text{ref}}}(x,t) = 0.001\left(\frac{N_n(\theta_n(x,t))}{D_n(\theta_n(x,t))}\right), \tag{2.18}$$

with: for all $\theta \in [0,1]$

$$\begin{aligned} N_n(\theta) =& 0.005269056 + 3.299265709\,\theta_n - 91.79325798\,\theta_n^2 + 1004.911008\,\theta^3 - 5812.278127\,\theta^4 \\ &+ 19329.7549\,\theta^5 - 37147.8947\,\theta^6 + 38379.18127\,\theta^7 - 16515.05308\,\theta^8, \\ D_n(\theta) =& 1 - 48.09287227\,\theta + 1017.234804\,\theta^2 - 10481.80419\,\theta^3 + 59431.3\,\theta^4 \\ &- 195881.6488\,\theta^5 + 374577.3152\,\theta^6 - 385821.1607\,\theta^7 + 165705.8597\,\theta^8, \end{aligned}$$

for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$U_{\text{ref},p}(x,t) = B_p(\theta_p(x,t)), \tag{2.19}$$
$$U_{\text{ref},n}(x,t) = B_n(\theta_n(x,t)), \tag{2.20}$$

with for all $\theta \in [0,1]$

$$\begin{aligned} B_p(\theta) =& \frac{-4.656 + 88.669\,\theta^2 - 401.119\,\theta^4 + 342.909\,\theta^6 - 462.471\,\theta^8 + 433.434\,\theta^{10}}{-1 + 18.933\,\theta^2 - 79.532\,\theta^4 + 37.311\,\theta^6 - 73.083\,\theta^8 + 95.96\,\theta^{10}}, \\ B_n(\theta) =& 0.7222 + 0.1387\,\theta + 0.029\,\theta^{0.5} - \frac{0.0172}{\theta} + \frac{0.0019}{\theta^{1.5}} + 0.2808e^{0.9-15\,\theta} \\ &- 0.7984e^{0.4465\,\theta - 0.4108}, \end{aligned}$$

where the normalized lithium concentration in the solid phases $\theta$ is defined as: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\theta_i(x,t) = \frac{C_s(x,t,R_{p,i})}{C_{s,i}^{\max}} \qquad i \in \{p,n\}, \tag{2.21}$$

where $C_s$ is the lithium ions concentration in the solid phases (*cf.* Table 4), and $C_s^{\max}$ is the maximum solid phases concentration (*cf.* Table 9).

### 2.4.9. Effective coefficients

Following effective medium theory (EMT; see Choy 2015) and in particular its approximation developed by Bruggeman (1935), the P2D model uses effective coefficients approximating mesoscopic (Imry, 2002) phenomena.

**Effective electrolyte diffusion coefficient** $D_{\text{eff},i}$ is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$D_{\text{eff},i}(x,t) = \epsilon_i^{\text{brugg}_i} \times 10^{-4} \times 10^{-4.43 - \frac{54}{T(x,t) - 229 - 5\times 10^{-3} C_e(x,t)} - 0.22\times 10^{-3} C_e(x,t)} \qquad i \in \{p,n\}, \tag{2.22}$$

where $\epsilon$ and brugg are respectively the porosity and Bruggeman's coefficient (*cf.* Table 9), $T$ and $C_e$ are respectively the temperature and the lithium ions concentration in the electrolyte (*cf.* Table 4).

**Effective solid phases diffusion coefficient** $D^s_{\text{eff},i}$ is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$D^s_{\text{eff},i}(x,t) = D^s_i e^{\frac{E_{a,i}^{D^s}}{R}\left(\frac{1}{T(x,t)} - \frac{1}{T_{\text{ref}}}\right)} \qquad i \in \{p,n\}, \tag{2.23}$$

where $D^s$, $E_a^{D^s}$ and $T_{\text{ref}}$ are respectively the solid phases diffusivity, the solid phases diffusion activation energy and the reference (or ambiant) temperature (*cf.* Table 9), $R$ is the universal gas constant and $T$ is the temperature (*cf.* Table 4).

**Effective solid phases diffusion coefficient** $\kappa_{\text{eff}}$ is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\kappa_{\text{eff},i}(x,t) = \epsilon_i^{\text{brugg}_i} \times 10^{-4} \times C_e(x,t) \times A(x,t)^2 \qquad i \in \{p,n\}, \tag{2.24}$$

with

$$\begin{aligned} A(x,t) = & -10.5 + 0.668 \times 10^{-3} C_e(x,t) + 0.494 \times 10^{-6} C_e(x,t)^2 \\ & + T(x,t)\left[0.074 - 1.78 \times 10^{-5} C_e(x,t) - 8.86 \times 10^{-10} C_e(x,t)^2\right] \\ & + T(x,t)^2\left[-6.96 \times 10^{-5} + 2.8 \times 10^{-8} C_e(x,t)\right], \end{aligned}$$

where $\epsilon$ and brugg are respectively the porosity and Bruggeman's coefficient (*cf.* Table 9), $T$ and $C_e$ are respectively the temperature and the lithium ions concentration in the electrolyte (*cf.* Table 4).

**Effective reaction rate** is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$k_{\text{eff}}(x,t) = k_i e^{\frac{E_{a,i}^k}{R}\left(\frac{1}{T(x,t)} - \frac{1}{T_{\text{ref}}}\right)} \qquad i \in \{p,n\}, \tag{2.25}$$

where $k$, $E_a^k$ and $T_{\text{ref}}$ are respectively the reaction rate constant, the reaction constant activation energy and the reference (or ambient) temperature (*cf.* Table 9), $R$ is the universal gas constant and $T$ is the temperature (*cf.* Table 4).

### 2.4.10. Heat source terms

The heat source terms derivation are carefully described in (Gu and Wang, 2000, see in particular Eq. (29) p.2913). Three heat source terms are considered here. $Q_{\text{ohm}}$ accounts for the ohmic Joule heating in both the electrolyte and the solid phases. $Q_{\text{rev}}$ and $Q_{\text{rxn}}$ accounts for their part for the electrochemical reactions occurring at the interface between the electrode material and the electrolyte (occurring then only in $p$ and $n$). $Q_{\text{rev}}$ is the reversible part of the reaction heat, mainly due to the entropy change of the electrode reaction, and $Q_{\text{rxn}}$ the irreversible part, due to the electrochemical reaction resistance at the interface (Gu and Wang, 2000, in particular Eq. (15)). Phase transformation is neglected here.

**Ohmic Joule heating** $Q_{\text{ohm}}$ is given in the electrodes by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$\begin{aligned} Q_{\text{ohm},i}(x,t) = & \sigma_{\text{eff},i}\left(\partial_x \phi_s(x,t)\right)^2 + \kappa_{\text{eff},i}(x,t)\left(\partial_x \phi_e(x,t)\right)^2 \\ & + \frac{2\kappa_{\text{eff},i}(x,t) RT(x,t)}{F}(1-t_+)\partial_x \ln C_e(x,t) \partial_x \phi_e(x,t) \qquad i \in \{p,n\}, \end{aligned} \tag{2.26}$$

where $\sigma_{\text{eff}}$ is the effective solid phases conductivity (*cf.* Table 11), $\kappa_{\text{eff}}$ is the effective electrolyte conductivity (*cf.* Table 8), $F$ and $R$ are respectively the Faraday's constant and the universal gas constant, $t_+$ is the transference number and $\phi_e$, $\phi_s$, $C_e$ and $T$ are respectively, the electrolyte potential, the solid phases potential, the lithium ions concentration in the electrolyte and the temperature (*cf.* Table 4). In the separator, the only effects remaining are those taking place in the electrolyte, giving: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$Q_{\text{ohm},s}(x,t) = \kappa_{\text{eff},s}(x,t)\left(\partial_x \phi_e(x,t)\right)^2 + \frac{2\kappa_{\text{eff},s}(x,t) RT(x,t)}{F}(1-t_+)\partial_x\left(\ln C_e(x,t)\right)\partial_x \phi_e(x,t). \tag{2.27}$$

**Irreversible electrochemical reaction heat** is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$Q_{\text{rxn}}(x,t) = Fa_i j_i(x,t)\eta_i(x,t) \qquad i \in \{p,n\}, \tag{2.28}$$

where $F$ is the Faraday's constant, $a$ is the particle surface area to volume (*cf.* Table 9), and $j$ and $\eta$ are respectively the solid phases ionic flux and the surface overpotential (*cf.* Table 8).

**Reversible electrochemical reaction heat** is modeled by: for all $x \in [\hat{x}_i, x_i]$, for all $t \in [0, T_{\max}]$

$$Q_{\text{rev}}(x,t) = Fa_i j_i(x,t)T(x,t)\partial_T U_i|_{T_{\text{ref}}}(x,t) \qquad i \in \{p,n\}, \tag{2.29}$$

where $F$ is the Faraday's constant, $a$ is the particle surface area to volume (*cf.* Table 9), $j$ and $\partial_T U|_{T_{\text{ref}}}$ are respectively the solid phases ionic flux and the open circuit potential entropic variation (*cf.* Table 8), and $T$ is the temperature (*cf.* Table 4).

### 2.4.11. Auxiliary variables

**State of Charge (SoC)** accounts for the level of charge of the cell relative to its maximal capacity. In real applications, this maximal capacity is a function of time, as the cell is subjected to an ageing mechanism (see Vetter et al. 2005) and the ratio of the maximal capacity at time $t$ (which can be challenging to estimate, see *e.g.* Jiang and Song 2022) over the nominal maximal capacity is called the State of Health (SoH). Nevertheless, we will neglect ageing effects in this work, and the maximal capacity will be considered as constant. As the power of the cell is given by the migration of Lithium ions from the anode to the cathode, the maximal charge is reached at $t_0 \in \mathbb{R}^+$ when for all $(x,r) \in [x_s, x_n] \times [0, R_{p,n}]$, $C_s(x, t_0, r) = C^s_{\max,n}$ where $C_s$ is the lithium ions concentration in the solid phases (*cf.* Table 4) and $C^s_{\max,n}$ is the maximum solid phases concentration in the anode (*cf.* Table 9). In our simplified model (see Section 2.4.4, Eq. (2.11b)) this is replaced by the equivalent statement for all $x \in [x_s, x_n]$, $\overline{C}_s(x, t_0) = C^s_{\max,n}$ with $\overline{C}_s$ the average lithium ions concentration in the solid phases, which in turn can be rephrased as $\frac{1}{x_n - x_s}\int_{x_s}^{x_n} \overline{C}(x, t_0)dx = C^s_{\max,n}$. Following Torchio et al. (2015), SoC is therefore defined as: for all $t \in [0, T_{\max}]$

$$SoC(t) = \frac{1}{x_n - x_s}\int_{x_s}^{x_n} \frac{\overline{C}(x,t)}{C^s_{\max,n}}dx. \tag{2.30}$$

**Output voltage** $U$ is the difference of potential between the positive pole and the negative pole of the cell, *i.e.* between the exterior extremity of the cathode ($x_a$) and the exterior extremity of the anode ($x_n$), since the effect of current collectors on the potential is neglected in the model. Following Torchio et al. (2015), we then define the output voltage as: for all $t \in [0, T_{\max}]$

$$V_o(t) = \phi_s(x_a, t) - \phi_s(x_n, t), \tag{2.31}$$

where $\phi_s$ is the solid phases potential (*cf.* Table 4).

# 3. Neural Networks and Physics Informed Neural Networks (PINNs)

In this section, we will introduce the main tools used in this work to handle the problem presented in Section 2. Those tools are composed of two main ingredients: Neural Networks (NNs), also refereed as Artificial Neural Networks (ANNs), and Physics Informed Neural Networks (PINNs).

**Neural Networks** were historically introduced under the name "Perceptron" by Rosenblatt (1958) as mathematical models emulating the behavior of a biological neurons (hence their name). In a biological neural network, several dendrites receive information from other neurons which is combined by the Soma before being transmitted by the axon and delivered to other neurons by synapses. Drawing inspiration from this, the idea for artificial neural networks is to combine "input signals" in an affine manner and to "filter" them by an "activation function" (for instance the sign function) before transferring this signal to the next neuron, performing thus an elementary operation. The important aspect is that the weights applied to each input, as well as the affine coefficient called "bias", in the affine combination are seen as "learnable parameters", *i.e.* parameters that will be adjusted in order to design the neuron to perform an elementary useful operation, which combined through the neurons stack called Multiple Layer Perceptron (MLP), will eventually perform very complex operations. Another fundamental idea is a training method known as "Stochastic Gradient Descent" (SGD), which was introduced by Amari (1967). Finally the third fundamental idea that gave birth to today known neural networks is back-propagation and automatic differentiation, which were first implemented and used in this context by Linnainmaa (1976) and then presented in their modern version by Werbos (1982), before being popularized by LeCun et al. (1998). Since then, this method has been developed in many directions and successfully applied to a very wide range of problems, including vision (Voulodimos et al., 2018), or natural language processing (Goldberg, 2022). In this work, we will stick to the classical Multiple Layer Perceptron (MLP) which is sufficient for our work.

**Physics Informed Neural Networks (PINNs)** were first introduced by Lagaris et al. (1998). They designate a method to approximate solutions of Partial Differential Equations (PDEs) built upon Neural Networks. The idea is to compute derivatives of the outputs of a neural network with respect to its inputs to finally combine it into a PDE residual and then train the neural network to minimize this residual. They have seen a resurgence of interest after being rediscovered by Raissi et al. (2019) who have wisely taken advantage of automatic differentiation (Baydin et al., 2018) of neural networks to compute the derivatives and have applied this method successfully to several forward and inverse problems. Since then, numerous extensions and applications have been developed (for a review, see for instance Karniadakis et al. 2021; Cuomo et al. 2022).

In the following, we will introduce Neural Networks (NNs) and Physics Informed Neural Networks (PINNs) in a mathematically formal manner, first introducing a pure formal framework in a functional analysis perspective, before depicting how this theoretical framework can be effectively approximated in practice.

## 3.1. Theoretical framework

In this section, we will introduce Neural Networks and PINNs in the language of functional analysis. Beyond "mathematical fanciness", this formalism effort is made in order to contextualize this work in theoretical frameworks that are developed nowadays upon Neural Networks, and notably the work of Jacot et al. (2018). This presentation will then follow while clarifying the formalism developed in Jacot et al. (2018), while extending the given point of view, partially following Mishra and Molinaro (2020), which is up to our knowledge, an original contribution.

### 3.1.1. Notations

In the remaining of Section 3.1, the following notations will be used.

- $\mathbb{N}_k$ will denote the set of natural integers starting at integer $k$.

  In particular, $\mathbb{N}_0$ will be the set of natural integers with 0 included and $\mathbb{N}_1$ the set of natural integers with 0 excluded.

- $\mathcal{F}(A \to B)$ will denote the set of functions from set A to set B. When $A = B$ we will simply denote $\mathcal{F}(A)$.

- $\mathcal{C}^0(A \to B)$ will denote the set of continuous functions from $A$ to $B$, when $A$ and $B$ are Banach spaces. When $A = B$ we will simply denote $\mathcal{C}^0(A)$.

- $\mathcal{C}^k(A \to B)$ will denote the set of $k$ times continuously differentiable functions from $A$ to $B$, when $A$ and $B$ are Banach spaces, with $k \in [1, +\infty) \cap \mathbb{N}_1$. When $A = B$ we will simply denote $\mathcal{C}^k(A)$.

- $\mathcal{C}^\infty(A \to B)$ will denote the set of indefinitely continuously differentiable functions from $A$ to $B$, when $A$ and $B$ are Banach spaces. When $A = B$ we will simply denote $\mathcal{C}^\infty(A)$.

- $\mathcal{C}^k_c(A \to B)$ will denote the set of functions in $\mathcal{C}^k(A \to B)$ for $k \in ([0, +\infty) \cap \mathbb{N}_0) \cup \{\infty\}$, with compact support. When $A = B$ we will simply denote $\mathcal{C}^k_c(A)$.

  When $k = 0$, we will simply note $\mathcal{C}_c(A \to B)$ or $\mathcal{C}_c(A)$ if $A = B$.

- $\mathcal{D}^k(A \to B)$ will denote the set of functions $k$ times almost everywhere differentiable functions from $A$ to $B$, when $A$ and $B$ are Banach spaces equipped with their Lebesgue measure, with $k \in [0, +\infty]$. When $A = B$ we will simply denote $\mathcal{D}(A)$.

- $\mathrm{d}f|_x$ will denote the differential of a function $f \in \mathcal{D}^k(A \to B)$ evaluated at point $x \in A$ whenever it exists.

- $\|\cdot\|_{p,m}$ will denote the $p$-norm $\|\cdot\|_{p,m} : x \in \mathbb{R}^m \mapsto \sqrt[p]{\sum_{i=1}^m |x_i|^p}$ on $\mathbb{R}^m$ with $1 \leqslant p < +\infty$. When the context is unambiguous, we will simply denote it $\|\cdot\|_p$.

  Of paramount importance will be the Euclidean norm, which is the 2-norm.

- $(e_i)_{i=1}^d$ wille denote the canonical basis of $\mathbb{R}^d$, *i.e.* the basis such that the coordinates of $e_i$ are zero except at position $i$ for all $1 \leqslant i \leqslant d$.

- $\mathrm{L}^p(A \to \mathbb{R}^m, \lambda)$ will denote the $L^p$ quotient space of functions:

$$\mathcal{L}^p(A \to \mathbb{R}^m) := \left\{ \int_A \|f(x)\|_{p,m}^p \, \lambda(\mathrm{d}x) < +\infty : f \in \mathcal{F}(A \to B),\ f\ \lambda\text{-mesurable} \right\},$$

quotiented by the equivalence relation: for all $f, g \in \mathcal{L}^p(A \to \mathbb{R}^m)$

$$f \sim_\lambda g : \iff \lambda\left(\{x \in A\ :\ f(x) \neq g(x)\}\right) = 0,$$

where $A$ is a measurable space with measure $\lambda$. This space is equipped with the norm $\|\cdot\|_{\mathrm{L}^p_m(A,\lambda)} : f \in \mathrm{L}^p(A \to \mathbb{R}^m, \lambda) \mapsto \left(\int_A \|f(x)\|_{p,m}^p \, \lambda(\mathrm{d}x)\right)^{\frac{1}{p}}$.

- $\mathcal{M}_{m,n}(\mathbb{R})$ will denote the set of matrices with $m$ lines and $n$ columns with real coefficients.

- Given a set sequence $(A_i)_{1 \leqslant i \leqslant d}$, $\prod_{i=1}^d A_i$ will design the Cartesian cross-product of the $k$ terms $(A_i)_{1 \leqslant i \leqslant d}$. More formally, it is the operation defined by induction by:
  - if $d = 1$: $\prod_{i=1}^d A_i = A_1$
  - otherwise: $\prod_{i=1}^d A_i = \left(\prod_{i=1}^{d-1} A_i\right) \times A_d$

- Given a set sequence $(A_i)_{0 \leqslant i \leqslant d}$ and a function sequence $(\alpha_i)_{1 \leqslant i \leqslant d} \in \prod_{i=1}^d \mathcal{F}(A_{i-1} \to A_i)$, $\circ_{i=1}^d \alpha_i \in \mathcal{F}(A_0 \to A_d)$ will design the $d-1$ consecutive functions composition. More formally, it is the operation defined by induction by:
  - if $d = 1$: $\circ_{i=1}^d \alpha_i = \alpha_1 \in \mathcal{F}(A_0 \to A_1)$
  - otherwise: $\circ_{i=1}^d \alpha_i = \underbrace{\alpha_d}_{\in \mathcal{F}(A_{d-1} \to A_d)} \circ \underbrace{\left(\circ_{i=1}^{d-1} \alpha_i\right)}_{\in \mathcal{F}(A_0 \to A_{d-1})}$

### 3.1.2. Neural networks

We begin by giving the formal definition of a multiple layers perceptron.

**Definition 3.1** (Multiple layers perceptron (MLP))**.** Let us fix:

- $d \in \mathbb{N}_1$, called the depth. When $d \geqslant 2$, we say that the MLP is deep.
- $(l_i)_{0 \leqslant i \leqslant d} \in \mathbb{N}_1^{d+1}$, called the widths.
- $(W_i)_{1 \leqslant i \leqslant d} \in \prod_{i=1}^d \mathcal{M}_{l_i, l_{i-1}}(\mathbb{R})$, called the weights (sometimes also kernels).
- $(b_i)_{1 \leqslant i \leqslant d} \in \prod_{i=1}^d \mathbb{R}^{l_i}$, called the biases.
- $(\alpha_i)_{1 \leqslant i \leqslant d} \in \prod_{i=1}^d \mathcal{F}(\mathbb{R}^{l_i})$, called the activations.

A Multiple Layers Perceptron is then the function defined by:

$$\mathrm{MLP} : \left\{ \begin{array}{ccc} \mathbb{R}^{l_0} & \longrightarrow & \mathbb{R}^{l_d} \\ x & \longmapsto & \left(\circ_{i=1}^d \alpha_i (W_i \cdot + b_i)\right)(x) \end{array} \right.$$

*Remark* 3.1. In the context of neural networks, the activation functions are usually "a scalar function repeated $k$ times". More precisely, for a given $k \in \mathbb{N}_1$, and given an activation $\alpha \in \mathcal{F}(\mathbb{R}^k)$, it exists $g : \mathbb{R} \to \mathbb{R}$ such that $\alpha = x \in \mathbb{R}^k \mapsto (g(x_i))_{1 \leqslant i \leqslant k} \in \mathbb{R}^k$.

In this case, it is also common in the context of neural networks, to abusively denote $\alpha$ by $g$, *i.e.* to say that we take a scalar function as an activation of the neural network.

Popular choices for activations, in the sense of choices for the function $g$ of Remark 3.1, can be found in Table 16 in Appendix B, the orignal source being Berner et al. (2021).

Identifying the most effective activation functions is still an active area of research (Dubey et al., 2022).

*Remark* 3.2. Since differentiability is a key ingredient in neural networks, notably for their training (see Section 3.1.3), we will mostly restrict to activations that are almost everywhere differentiable, *i.e.* functions in the set $\mathcal{D}^1(\mathbb{R}^k)$ for some $k \in \mathbb{N}_1$ (this is for instance the case for Lipschitz-continuous functions, as stated by the Rademacher's theorem, see *e.g.* Evans 2018, Section 3.1).

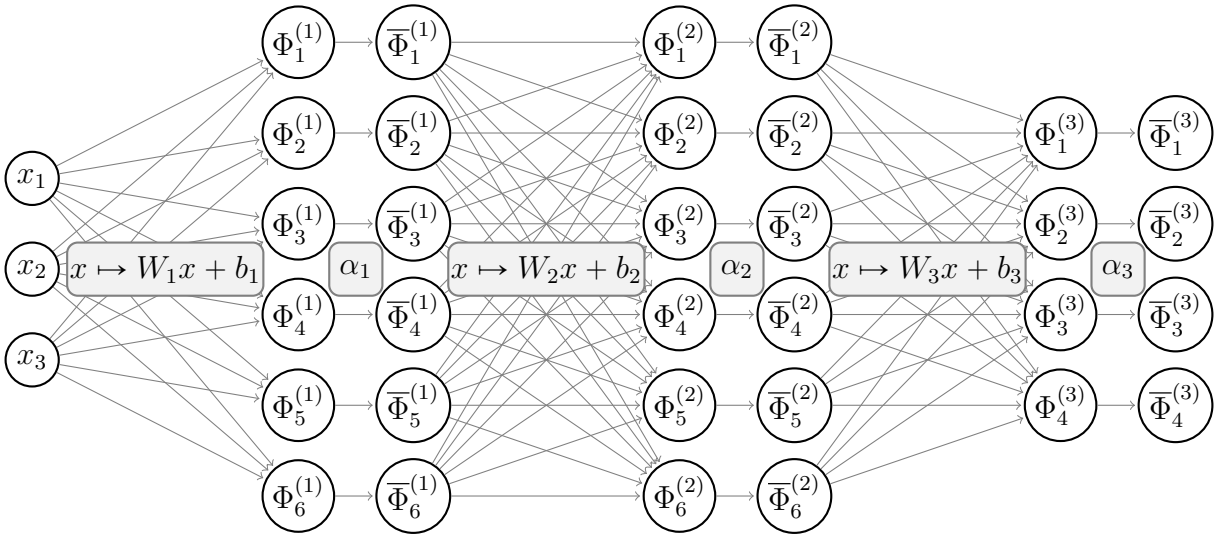



Figure 3: Illustration of a MLP : $\mathbb{R}^3 \to \mathbb{R}^4$ of depth 3 with widths $(3, 6, 6, 4)$ and activations $(\alpha_i)_{i=1}^3$ (adapted from Berner et al. 2021).
*Reading:* the input $x \in \mathbb{R}^3$ is first multiplied by matrix $W_1 \in \mathcal{M}_{6,3}(\mathbb{R})$ before adding $b_1 \in \mathbb{R}^6$. Then $\alpha_1$ is applied to the result. We carry on in the same way, applying the operations from left to right along the graph, thus obtaining at the end a result in $\mathbb{R}^4$ outputted from $\alpha_3$.
More formally, we have: for all $1 \leqslant i \leqslant 3$ and for all $1 \leqslant j \leqslant l_i$ $\Phi_j^{(i)} : x \in \mathbb{R}^3 \mapsto \left(W_j \left(\circ_{k=1}^{j-1} \alpha_k \left(W_k \cdot + b_k\right)\right)(x) + b_j\right)_i$ and $\overline{\Phi}_j^{(i)} : x \in \mathbb{R}^3 \mapsto \left(\left(\circ_{k=1}^{j} \alpha_k \left(W_k \cdot + b_k\right)\right)(x)\right)_i$.

We will now move to a functional analysis perspective, which will allow us to really develop our approach. We begin with a structure that lies at the heart of the practical use of neural networks, that is architectures. An architecture fixes the size of the neural network (*i.e.* its depth and layers size), the symmetries among weights (for instance, we may want some coefficients of the weights to be equal as for convolutional nets ; see *e.g.* Albawi et al. 2017), as well as the activations used, leaving only the choice of some weights and biases free, which will be "learned" (giving a rigorous definition to this will the subject of the next section), the others being fixed (for instance to ensure sparsity). We give thereafter a formal definition, in the language of functional analysis:

**Definition 3.2** (architecture)**.** Let fix:

- $d \in \mathbb{N}_1$, called the depth.

- $(l_i)_{0\leqslant i\leqslant d} \in \mathbb{N}_1^d$, called the widths.
- $(\alpha_i)_{1\leqslant i\leqslant d} \in \prod_{i=1}^d \mathcal{D}^1(\mathbb{R}^{l_i})$, called the activations.
- $P \in \mathbb{N}_1$ such that $1 \leqslant P \leqslant \sum_{i=1}^d (l_i \times (l_{i-1}+1))$, called the number of parameters.
- $\psi \in \mathcal{C}^1\left(\mathbb{R}^P \to \prod_{i=1}^n \mathcal{M}_{l_i,l_{i-1}}(\mathbb{R}) \times \prod_{i=1}^n \mathbb{R}^{l_i}\right)$, called the parameters map.

Then an architecture for an MLP is a function:

$$\Psi : \left\{ \begin{array}{ccc} \mathbb{R}^P & \longrightarrow & \mathcal{F}(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d}) \\ \theta & \longmapsto & \circ_{k=1}^d \alpha_i \left(\psi_k(\theta) \cdot + \psi_{n+k}(\theta)\right) \end{array} \right. .$$

*Remark* 3.3. The parameters map $\psi$ is the map that encodes symmetries among weights and biases, as well as the selection of "learnable" and "fixed" weights. In this work, we will essentially see $\psi$ as a suitable re-parametrization of a real vector of dimension $P = \sum_{i=1}^d (l_i \times (l_{i-1}+1))$ into coefficients of matrices in $\prod_{i=1}^n \mathcal{M}_{l_i,l_{i-1}}(\mathbb{R})$ and coordinates of smaller vectors in $\prod_{i=1}^n \mathbb{R}^{l_i}$.

A natural question that arises is to understand the structure of the image of an architecture. Interestingly, it can be shown that it has a Riemmanian pseudo-manifold structure with metric related to the so-called Neural-Tangent Kernel (NTK) developed in Jacot et al. (2018). Details can be found in Appendix D.3.

To be able to formulate approximation results, as well as results concerning certain asymptotic behaviors, it is convenient to consider functional spaces obtained by taking the union of a collection of architectures. To this aim, we will define some spaces of interest. We begin by the space of MLPs, which is the functional space covering all MLP configurations, given input and output dimensions:

**Definition 3.3** (Space of MLPs)**.** The functional space of MLPs is the subset $\mathcal{NN}_{n,m}$ of $\mathcal{F}(\mathbb{R}^n \to \mathbb{R}^m)$ defined by:

$$\begin{aligned} \mathcal{NN}_{n,m} := \Bigg\{ \circ_{i=1}^d \alpha_i(W_i \cdot + b_i) : d \in \mathbb{N}_1,\ (l_i)_{0\leqslant i\leqslant d} \in \mathbb{N}_1^{d+1},\ l_0 = n,\ l_d = m, \\ (W_i)_{1\leqslant i\leqslant d} \in \prod_{i=1}^d \mathcal{M}_{l_i,l_{i-1}}(\mathbb{R}),\ (b_i)_{1\leqslant i\leqslant d} \in \prod_{i=1}^d \mathbb{R}^{l_i}, \\ (\alpha_i)_{1\leqslant i\leqslant d} \in \prod_{i=1}^d \mathcal{D}^1(\mathbb{R}^{l_i}) \Bigg\}. \end{aligned} \tag{3.1}$$

Now, we want a generic way of designating a subspace obtained as a union of architectures images. We will then define a notation for this in pseudo-formal terms:

*Notation* 3.1 (Set of architecture collections spaces). The set of architecture collections spaces is the subset $\mathcal{P}_{n,m}$ of the set of subsets of $\mathcal{NN}_{n,m}$ denoted by:

$$\mathcal{P}_{n,m} := \left\{ \bigcup_{i\in I} \operatorname{Im} \Psi_i \ : (\Psi_i)_{i\in I} \text{ architecture collections with codomain } \mathcal{F}(\mathbb{R}^n \to \mathbb{R}^m) \right\}.$$

In the context of our work, we will be particularly interested in collections of architectures that only use smooth activations. We will thus set special notations for that. First we denote the smooth space of MLPs:

*Notation* 3.2 (Smooth space of MLPs). The smooth space of MLP, denoted $\mathcal{NN}^{\infty}_{n,m}$, is the subspace of $\mathcal{NN}_{n,m}$ where the activations $(\alpha_i)_{1\leqslant i\leqslant d}$ are infinitely differentiable.

In the same spirit, we set a special notation for the set of architecture collections spaces, with smooth activations:

*Notation* 3.3 (Set of architecture collections with smooth activation spaces). The set of architecture collections with smooth activation spaces, denoted $\mathcal{P}^{\infty}_{n,m}$, is the subset of $\mathcal{P}_{n,m}$ where all architectures collections use infinitely differentiable activations.

Finally, most of the time, we do not want the domain of a MLP to be the whole $\mathbb{R}^n$, but only a subdomain $\Omega$ of $\mathbb{R}^n$. In this case we introduce the natural restriction on $\mathcal{NN}_{n,m}$:

**Definition 3.4** (Restricted space of MLPs)**.** Given $\Omega$ a subdomain of $\mathbb{R}^n$, the MLP space restricted to $\Omega$ is:

$$\mathcal{NN}_{n,m}(\Omega) := \{f_{|\Omega} : \ f \in \mathcal{NN}_{n,m}\}. \tag{3.2}$$

We introduce the same restriction as well for $\mathcal{P}_{n,m}$:

*Notation* 3.4 (Restricted set of architecture collections spaces). Given $\Omega$ a subdomain of $\mathbb{R}^n$, the set of architecture collections restricted to $\Omega$ is:

$$\mathcal{P}_{n,m}(\Omega) := \left\{\{f_{|\Omega} : f \in A\} \ : A \in \mathcal{P}_{n,m}\right\}.$$

*Remark* 3.4. We will also use the notations $\mathcal{NN}^{\infty}_{n,m}(\Omega)$ and $\mathcal{P}^{\infty}_{n,m}(\Omega)$, which are defined likewise, combining Notation 3.2 and Definition 3.4, and Notation 3.3 and Notation 3.4, respectively.

We now turn to a density result that justifies the interest of the tools we have just introduced. Indeed, $\mathcal{NN}_{n,m}$ has the interesting property of being dense in $\mathcal{C}^0_c(\mathbb{R}^n \to \mathbb{R}^m)$, the space of continuous functions from $\mathbb{R}^n$ to $\mathbb{R}^m$ with compact support. This is of paramount importance, since $\mathcal{C}^0_c(\mathbb{R}^n \to \mathbb{R}^m)$ is dense in most "common" functional spaces ($\mathrm{L}_p$ spaces for $1 \leqslant p < \infty$, Sobolev spaces, *etc.*) and thus implies that MLPs can be used to approximate most of the "common" functions. Moreover, this result remains true for many spaces $\mathcal{A} \in \mathcal{P}_{n,m}$ built upon infinite architecture collections. We state this result more precisely:

**Theorem 3.1** (Universal approximation theorem)**.** *Let be $f \in \mathcal{C}^0_c(\mathbb{R}^n \to \mathbb{R}^m)$, $d \geqslant 2$, $\|\cdot\|$ a given norm on $\mathbb{R}^m$, and $(g_i)_{1\leqslant i\leqslant d-1} \in \mathcal{F}(\mathbb{R} \to \mathbb{R})^{d-1}$, each being locally bounded, piecewise continuous, and non-polynomial.*

*Then for all $\varepsilon > 0$, there exist $(l_i)_{0\leqslant i\leqslant d} \in \mathbb{N}_1^d$ with $l_0 = n$ and $l_d = m$, $(W_i)_{1\leqslant i\leqslant d} \in \prod_{i=1}^d \mathcal{M}_{l_i,l_{i-1}}(\mathbb{R})$, and $(b_i)_{1\leqslant i\leqslant d} \in \prod_{i=1}^d \mathbb{R}^{l_i}$ such that*

$$f_\theta : x \in \mathbb{R}^n \mapsto \left(\circ_{i=1}^d \alpha_i(W_i \cdot + b_i)\right)(x) \in \mathbb{R}^m$$

*with $\forall 1 \leqslant i \leqslant d-1$, $\alpha_i : x \in \mathbb{R}^{l_i} \mapsto (g_i(x_j))_{1\leqslant j\leqslant l_i} \in \mathbb{R}^{l_i}$ and $\alpha_d = x \in \mathbb{R}^m \mapsto x \in \mathbb{R}^m$, satisfies:*

$$\sup_{x\in\mathbb{R}^n} \|f_\theta(x) - f(x)\| < \varepsilon$$

*Proof.* *cf.* Leshno et al. (1993). □

Having introduced the fundamental formal notions for neural networks spaces, we will introduce in the next section, the notion of training and the fundamental notions associated to it.

### 3.1.3. Neural network training

The "training" of a neural network is a general term referring to any method which, given an architecture in the sense of Definition 3.2, seeks to find the optimal weights and biases to fit a given "objective", for instance approximating a given function. The state-of-the-art approach to "objective" is by designation of a so-called "Loss"-function encompassing the problem structure and which allows to state the objective in terms of a minimization problem. Finding the appropriate losses for a given objective is a significant part of the construction of deep learning algorithms (Wang et al., 2022a), and physics informed neural networks can be seen as an example of this.

Following our guideline, we now give a general and formal definition of Loss functions in the language of functional analysis:

**Definition 3.5** (Loss)**.** Let $\mathcal{B}$ be a Banach space equipped with a norm $\|\cdot\|_{\mathcal{B}}$, containing a space $\mathcal{A} \in \mathcal{P}_{n,m}(\Omega)$ as a dense subspace with respect to $\|\cdot\|_{\mathcal{B}}$, for some domain $\Omega \subset \mathbb{R}^n$.

A **loss** is then a differentiable function $\mathcal{L} : \mathcal{B} \to \mathbb{R}$ which is bounded from below. Without loss of generality, we may assume $\mathcal{L} : \mathcal{B} \to [0, +\infty)$.

*Remark* 3.5. Other properties can be asked for losses, such as convexity in order to ensure existence of an unique minimum. Similarly, the density assumption is not strictly necessary, but no useful result can be expected without it.

Thereafter, we give a fundamental example of a theoretical loss function:

*Example* 3.1 (Theoretical mean square error). Let $\mathcal{B} = \mathrm{L}^2(\mathbb{R}^n \to \mathbb{R}^m)$ be the $\mathrm{L}^2$ quotient space of functions defined on $\mathbb{R}^n$ with values in $\mathbb{R}^m$. Then $\mathcal{B}$ is complete, thus a Banach space, and admits $\mathcal{C}_c^{+\infty}(\mathbb{R}^n \to \mathbb{R}^m)$ (and therefore $\mathcal{NN}_{n,m}$ or even $\mathcal{NN}_{n,m}^{\infty}$) as a dense subspace. We can then define the Loss:

$$\mathcal{L}_2 : f \in \mathrm{L}^2(\mathbb{R}^n \to \mathbb{R}^m) \mapsto \left( \int_{\mathbb{R}^n} \sum_{i=1}^{m} f_i(x)^2 \, \mathrm{d}x \right) \in [0, +\infty).$$

$\mathcal{L}_2$ is then differentiable on the whole $\mathrm{L}^2(\mathbb{R}^n \to \mathbb{R}^m)$ with differential $\mathrm{d}\mathcal{L}_2\Big|_f$ at $f \in \mathrm{L}^2(\mathbb{R}^n \to \mathbb{R}^m)$ given by:

$$\mathrm{d}\mathcal{L}_2\Big|_f : h \in \mathrm{L}^2(\mathbb{R}^n \to \mathbb{R}^m) \mapsto \left( \int_{\mathbb{R}^n} \sum_{i=1}^{m} 2 f_i(x) h_i(x) \, \mathrm{d}x \right) \in [0, +\infty).$$

This loss is called the theoretical mean square error and is used extensively in the context of neural networks.

We now have the key ingredients to define the theoretical gradient descent of an MLP, also known as the theoretical gradient flow:

**Definition 3.6** (Theoretical gradient flow of an MLP)**.** Let us fix $\theta_0 \in \mathbb{R}^P$, an architecture $\Psi : \mathbb{R}^P \to \mathcal{F}(\mathbb{R}^n \to \mathbb{R}^m)$ as in Definition 3.2, and a Banach space $\mathcal{B}$ and a Loss $\mathcal{L} : \mathcal{B} \to \mathbb{R}$ as in Definition 3.5. The theoretical gradient flow for an MLP with architecture $\Psi$ is the following Ordinary Differential Equation (ODE) on the function $\theta : [0, +\infty) \to \mathbb{R}^P$:

$$\begin{cases} \theta(0) & = \theta_0 \\ \frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) & = -\mathrm{d}\mathcal{L}|_{\Psi(\theta(t))}(\partial_p \Psi(\theta(t))), \quad \forall 1 \leqslant p \leqslant P, \, \forall t \in (0, +\infty) \end{cases}. \tag{3.3}$$

*Remark* 3.6. We can remark that if for some $t_0 \geqslant 0$, $-\mathrm{d}\mathcal{L}|_{\Psi(\theta(t_0))} = 0$, then $\forall t \geqslant t_0$, $\theta(t) = \theta_0$. Stated differently, if $\theta$ "reaches" a critical point of $\mathcal{L}|_{\mathrm{Im}\,\Psi}$, then it will "remain stuck" in this critical point. This justifies the "training" denomination, since the goal of the gradient descent is to reach a local minimum of $\mathcal{L}|_{\mathrm{Im}\,\Psi}$ with $t$ as small as possible (since we want to reach an approximation in as little time as possible).

It can be noted that in the definition above, neither the convergence of the ODE to a critical point, nor the "quality" of this critical point, *i.e.* if it is a global optimum or even an optimum at all, is guaranteed. In fact, while general enough convergence results and bounds estimates exists for linear neural networks trained with such a gradient flow (Arora et al., 2018), such results under suitably general assumptions are still missing for deep neural networks. Nevertheless, some advances have been made recently by Boursier et al. (2022) with strong assumptions on the loss. Interestingly enough Yang (2019); Chizat et al. (2019) have identified different asymptotic training regimes with respect to the layer's size, depending on the choice of $\theta_0$, while studying the dynamics in the so-called lazy and mean-field regime. The study of the dynamics in the so-called active regime has been conducted by Jacot et al. (2022). Other promising works, conducted by Chizat and Bach (2018) obtained asymptotic global convergence results with a gradient flow adapted from Wasserstein's gradient flow of Optimal Transport (Santambrogio, 2017). Finally Bach and Chizat (2021) carefully studied the asymptotic dynamics of the gradient flow with respect to the layer's size.

Interestingly, Jacot et al. (2018) have mentioned a correspondence between the theoretical gradient flow of Definition 3.6 and the so-called Neural Tangent Kernel (NTK) they developed. In Appendix D.2, we show that both are equivalent under some assumptions.

Having developed the main theoretical tools of MLP, we will introduce in the next section, tools which are more specific to Physics Informed Neural Networks (PINNs).

### 3.1.4. PINNs problem position

Let $\Omega$ be an open domain of $\mathbb{R}^n$, $n \in \mathbb{N}_1$, with compact closure $\overline{\Omega}$ and boundary $\partial\Omega$, such that $\overline{\Omega} = \Omega \cup \partial\Omega$. Let us then fix $m \in \mathbb{N}_1$ and $\mathcal{F}_{\mathrm{NN}}$ a Banach space, equipped with a norm $\|\cdot\|_{\mathrm{NN}}$ containing some $\mathcal{A} \in \mathcal{P}^{\infty}_{n,m}(\overline{\Omega})$ as a dense subspace with respect to $\|\cdot\|_{\mathrm{NN}}$. Even if we keep it abstract for the sake of generalization, we can essentially see $\mathcal{F}_{\mathrm{NN}}$ as a $\mathcal{C}^k$ space or a Sobolev space.

We can then define the theoretical PINN problem:

**Definition 3.7.** Let $m_D, m_B \in \mathbb{N}_1$ and two operators $D : \mathcal{F}_{\mathrm{NN}} \to \mathrm{L}^2(\Omega \to \mathbb{R}^{m_D}, \lambda_n)$ and $B : \mathcal{F}_{\mathrm{NN}} \to \mathrm{L}^2(\partial\Omega \to \mathbb{R}^{m_B}, \sigma_n)$, differentiable for $\|\cdot\|_{\mathrm{L}^2_{m_D}(\Omega,\lambda_n)}$ and $\|\cdot\|_{\mathrm{L}^2_{m_B}(\partial\Omega,\sigma_n)}$ respectively, with respect to $\|\cdot\|_{\mathrm{NN}}$, where $\lambda_n$ and $\sigma_n$ are Lebesgue measures in $\Omega$ and surface measure on $\partial\Omega$ respectively. The PINN equation in $u \in \mathcal{F}_{\mathrm{NN}}$ is then given by:

$$\begin{cases} D(u) = 0 & \text{in } \Omega \\ B(u) = 0 & \text{on } \partial\Omega \end{cases}. \tag{3.4}$$

Of course Definition 3.7 encompasses most PDEs, $D$ being in this case a differential operator, and $B$ a trace operator (Evans, 2010, Section 5.5 page 271). We illustrate this in the following example, for the Laplace equation:

*Example* 3.2. Let us take $\mathcal{F}_{\mathrm{NN}} = \mathcal{C}^2(\Omega \to \mathbb{R}) \cap \mathcal{C}(\overline{\Omega} \to \mathbb{R})$, $D$ a second order PDE, *e.g.* $D(f) := \Delta f$, and $B$ a Dirichlet boundary condition, *e.g.* $B(f) = f_{|\partial\Omega} - u_0$ where $u_0 \in \mathcal{C}^0(\partial\Omega \to \mathbb{R})$.

Now, to solve the problem of Definition 3.7 with Neural Networks, we have to connect it with the theory developed in Sections 3.1.2 and 3.1.3. This link will be achieved through the theoretical loss of PINNs we will now define:

**Definition 3.8** (Theoretical loss of PINNs)**.** Let be $f \in \mathcal{F}_{\text{NN}}$ with $\mathcal{F}_{\text{NN}}$ as in Definition 3.7. The theoretical loss of PINNs is then defined as:

$$\mathcal{L}_{\text{PINN}}(f) := \gamma_D \left\| D(f) \right\|^2_{\text{L}^2_{m_D}(\Omega,\lambda_n)} + \gamma_B \left\| B(f) \right\|^2_{\text{L}^2_{m_B}(\partial\Omega,\sigma_n)}, \tag{3.5}$$

where $\gamma_D, \gamma_B > 0$ are weights that are of utmost importance for performing the minimization in practice, as outlined in Section 4.3.1.

*Remark* 3.7. It is straightforward to see that: for all $f \in \mathcal{F}_{\text{NN}}$

$$\mathcal{L}_{\text{PINN}}(f) = 0 \iff f \text{ is solution to (3.4)}.$$

This motivates the fact that finding a "good minimizer" $f$ for $\mathcal{L}_{\text{PINN}}$, gives reasonable hope to find a good approximation of a solution to (3.4). Of course this assertion lacks precision and effective approximation results, but such estimates strongly rely on $D$ and $B$ properties, and are thus problem specific. For second-order elliptic PDE, we can for instance refer to (Gilbarg et al., 1977, Chapter 6). More precise generalization error estimates results, using PINNs methods applied to specific problems, will be mentioned in Section 3.2.4.

*Remark* 3.8. To ensure consistency with Sections 3.1.2 and 3.1.3, $\mathcal{L}_{\text{PINN}}(f)$ of definition Definition 3.8 should be a loss in the sense of Definition 3.5. But this is essentially true by assumption, since $\mathcal{F}_{\text{NN}}$ is a Banach space containing an $\mathcal{A} \in \mathcal{P}^\infty_{n,m}(\overline{\Omega})$ as a dense subspace, and $\mathcal{L}_{\text{PINN}}$ is differentiable with respect to $\|\cdot\|_{\text{NN}}$ by differentiability of $D$ and $B$ with respect to $\|\cdot\|_{\text{NN}}$, and differentiability of $\|\cdot\|^2_{\text{L}^2_{m_D}(\Omega,\lambda_n)}$ and $\|\cdot\|^2_{\text{L}^2_{m_B}(\partial\Omega,\sigma_n)}$.

While this framework is useful to conduct further theoretical analysis, it cannot be implemented immediately in an algorithmic way. To overcome this, we will develop in the following section, a framework approximating the one presented above.

## 3.2. Approximations to the theoretical framework

While it is very useful to have a mathematically coherent picture, the theoretical framework presented in Section 3.1 cannot be actually implemented on a computer, since it involves non-"computable" operations. Without going into the theory of computation (Sipser, 2012), which would be beyond the scope of this work, we will simply say that a function is computable if it requires only a finite number of discrete operations. In this regard, the framework introduced above has three main shortcomings:

- the impossibility to actually compute the norms $\|\cdot\|^2_{\text{L}^2_{m_D}(\Omega,\lambda_n)}$ and $\|\cdot\|^2_{\text{L}^2_{m_B}(\partial\Omega,\sigma_n)}$ defining the theoretical PINN Loss of Definition 3.8, since they rely on the computation of integrals. We must therefore rewrite it using some quadrature rule. This will be the purpose of Section 3.2.3.

- the impossibility to actually solve the gradient flow of Definition 3.6. We must therefore rewrite it using some iterative method. This will the purpose of Section 3.2.1.

- Depending on the operators $D$ and $B$ of Definition 3.7, an approximation may be also required, as well as for their differentials. Nevertheless, we will see that for differential operators, such a problem does not exist, thanks to automatic differentiation. A cautious presentation of this method is conducted in Section 3.2.2.

In addition to these problems, we will discuss existing theoretical concepts in deep learning for quantifying approximation error, and review the main existing results in Section 3.2.4.

We start in the next section by presenting the gradient flow approximation.

### 3.2.1. Gradient flow approximation

In this section, we will explain how the theoretical gradient flow defined in Equation (3.3) can be approximated in order to be numerically implemented.

For the sake of clarity, let us recall Equation (3.3):

$$\begin{cases} \theta(0) & = \theta_0 \\ \frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) & = -\mathrm{d}\mathcal{L}|_{\Psi(\theta(t))}(\partial_p\Psi(\theta(t))), \quad \forall 1 \leqslant p \leqslant P, \, \forall t \in (0,+\infty) \end{cases}.$$

One of the most glaring problem in the perspective of implementing this algorithm is the computation of the differential $\mathrm{d}\mathcal{L}|_{\Psi(\theta(t))}$, considering that $\mathrm{d}\mathcal{L}$ is the differential of a functional defined on a functional space $\mathcal{B}$. However, this problem can easily be circumvented by remarking that: for all $t \in (0,+\infty)$

$$\frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) = -\mathrm{d}\mathcal{L}|_{\Psi(\theta(t))}(\partial_p\Psi(\theta(t))), \quad \forall 1 \leqslant p \leqslant P,$$

can be rewritten, by defintion of $\partial_p\Psi(\theta(t))$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) = -\mathrm{d}\mathcal{L}|_{\Psi(\theta(t))}\left(\mathrm{d}\Psi|_{\theta(t)}(e_p)\right), \quad \forall 1 \leqslant p \leqslant P,$$

with $(e_i)_{i=1}^P$ being the canonical basis of $\mathbb{R}^P$. We can then introduce the new loss:

$$\ell : \begin{cases} \mathbb{R}^P & \longrightarrow & [0,+\infty) \\ \omega & \longmapsto & \mathcal{L}\left(\Psi(\omega)\right) \end{cases}, \tag{3.6}$$

which is a differentiable multivariate function from the finite-dimensional vector space $\mathbb{R}^P$ to the scalar space $[0,+\infty)$. We can then rewrite the Equation (3.3) as: for all $t \in (0,+\infty)$

$$\frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) = -\mathrm{d}\ell|_{\theta(t)}(e_p), \quad \forall 1 \leqslant p \leqslant P, \tag{3.7}$$

thus eliminating the need of computing $\mathrm{d}\mathcal{L}$. It remains to be seen how to calculate $\mathrm{d}\ell_\omega$ for a given $\omega \in \mathbb{R}^P$ on a computer, but we will show that this can be easily achieved through the automatic differentiation mechanism presented in Section 3.2.2 below. For the sake of brevity, let us introduce the so-called gradient notation:

$$\nabla\ell : \begin{cases} \mathbb{R}^P & \longrightarrow & \mathbb{R}^P \\ \omega & \longmapsto & \left(\mathrm{d}\ell|_\omega(e_p)\right)_{p=1}^P \end{cases},$$

to rewrite the Equation (3.7) as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\theta(t) = -\nabla\ell(\theta(t)). \tag{3.8}$$

The only problem remaining is now solving the Ordinary Differential Equation (ODE):

$$\begin{cases} \theta(0) & = \theta_0 \\ \frac{\mathrm{d}}{\mathrm{d}t}\theta(t) & = -\nabla\ell(\theta(t)), \quad \forall t \in (0, +\infty) \end{cases}. \tag{3.9}$$

Unless $\ell$ has a particularly appropriate form, there is often no closed-form solution to this ODE. This is because $\ell$ strongly depends on the architecture $\Psi$, which can be very complex and highly nonlinear. So we need to find a way of approximating this equation, bearing in mind the difficulties caused by the non-linearity of $\Psi$.

*Remark* 3.9. In a certain perspective, this can be seen as a consequence of substituting the functional space $\mathcal{B}$ by $\operatorname{Im}\Psi$ (or even any $\mathcal{A} \in \mathcal{NN}_{n,m}(\Omega)$). While the loss function $\mathrm{d}\mathcal{L}$ can have "good properties" (*e.g.* convexity) in $\mathcal{B}$, the elements of $\mathcal{B}$ cannot be computed on a computer. We have therefore come to rely on the functions of $\operatorname{Im}\Psi$, which are by design computable on a computer and are known to have good approximations properties in $\mathcal{B}$ (*cf.* Theorem 3.1), but this comes at the cost of losing the "good properties" of $\mathcal{L}$ due to the highly nonlinear nature of $\Psi$. Note also that this non-linear nature is not a contingency that could easily be dispensed with, but the very essence of what allows neural networks to have good density properties in $\mathcal{B}$. We also refer you to Section 3.2.4, where we develop the notion of "optimization error", which is intimately linked to the above considerations.

The most simple idea is to write the first order Taylor's expansion of the function $\theta : \mathbb{R} \to \mathbb{R}^P$ at the point $t \in [0, +\infty)$: for all $h > 0$

$$\theta(t + h) = \theta(t) + \frac{\mathrm{d}}{\mathrm{d}t}\theta(t)h + \varepsilon_t(h)h,$$

with $\varepsilon_t : [0, +\infty) \to \mathbb{R}^P$ such that $\lim_{h\to 0} \|\varepsilon_t(h)\|_{2,P} = 0$. Injecting Equation (3.8), we get: for all $h > 0$

$$\theta(t + h) = \theta(t) - \nabla\ell(\theta(t))h + \varepsilon_t(h)h. \tag{3.10}$$

For $h > 0$ small enough, we can then make the Ansatz:

$$\theta(t + h) \simeq \theta(t) - \nabla\ell(\theta(t))h,$$

neglecting the higher-order terms $\varepsilon_t(h)h$. We can then iteratively approximate any $\theta(t_1)$ for $t_1 > 0$ and $N_1 \in \mathbb{N}_1$ large enough by defining the sequence: for all $0 \leqslant n \leqslant N_1 - 1$,

$$\tilde{\theta}_n^{t_1,N_1} := \begin{cases} \theta_0 & \text{if } n = 0 \\ \tilde{\theta}_{n-1}^{t_1,N_1} - \nabla\ell(\tilde{\theta}_{n-1})\frac{t_1}{N_1} & \text{if } 1 \leqslant n < N_1 \end{cases}.$$

This very simple method dates back to Euler and today bears his name. It is now part of a set of classical methods for numerically solving ODEs, known as the Runge–Kutta methods (Butcher, 2016, Chapter 3).

Surprisingly[5], it appears that this approximation works well enough in practice to train neural networks, since all the breakthroughs in deep learning, from image recognition (LeCun et al., 1998) to more recent transformers architectures (Vaswani et al., 2017) among which Generative pre-trained transformers (GPT)[6] models, have been obtained with adaptations of this method, that we will now present. Let us start by rephrasing the general "vanilla" gradient descent:

[5] In fact, a proper understanding of this phenomenon is still missing. See Remark 3.7 for a small survey of the existing results.

[6] https://en.wikipedia.org/wiki/Generative_pre-trained_transformer

**Definition 3.9** (Gradient descent (GD))**.** Let us fix $\theta_0 \in \mathbb{R}^P$, and a loss $\ell : \mathbb{R}^P \to \mathbb{R}$ built after, like in Equation (3.6), an architecture $\Psi : \mathbb{R}^P \to \mathcal{F}(\mathbb{R}^n \to \mathbb{R}^m)$ from Definition 3.2, and a Loss $\mathcal{L} : \mathcal{B} \to \mathbb{R}$ from Definition 3.5. Then, given a so-called **"learning rate"** $\delta > 0$, the **gradient descent** (LeCun et al., 1998, Equation 2) applied to $\ell$ is the inductively defined sequence $(\tilde{\theta}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$: for all $n \in \mathbb{N}_0$

$$\tilde{\theta}_n := \begin{cases} \theta_0 & \text{if } n = 0 \\ \tilde{\theta}_{n-1} - \delta \nabla \ell(\tilde{\theta}_{n-1}) & \text{if } n \geqslant 1 \end{cases}. \tag{3.11}$$

*Remark* 3.10. As already mentioned in seminal papers such like (LeCun et al., 1998, Equation 3), the method used in deep learning is called "Stochastic Gradient Descent" (SGD) and was introduced as a slightly modification of the gradient descent algorithm. In fact the modification concerns the loss $\ell$ and not directly the algorithm, which remains the same once $\ell$ has been modified. We mention this in more detail in Remark 3.12 in Section 3.2.3, where we discuss the theoretical loss approximation.

The difficulty in making the algorithm of Definition 3.9 work in practice, *i.e.* ensuring that it converges to at least a local minimum of $\ell$, lies in finding a suitable learning rate $\delta$. This is usually an important part of the experimental work to be carried out when working with neural networks. Nevertheless, two adaptations of this algorithm have been proposed, offering significant performance gains in practice:

**(S)GD with momentum** first introduced by Rumelhart et al. (1985) (similar ideas seem[7] to have been developed by Nesterov 1983). The idea is to take into account the former updates of the $(\tilde{\theta}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$, *i.e.* to somehow keep a record of $\left(\nabla \ell(\tilde{\theta}_i)\right)_{i=0}^{n-1}$ to get a better idea of the direction in which to update $\tilde{\theta}_{n-1}$ in order to get $\tilde{\theta}_n$. This is achieved by fixing $0 < \beta < 1$ and $\delta > 0$, and adding a sequence $(\overline{m}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$ inductively defined by: for all $n \in \mathbb{N}_0$

$$\tilde{\theta}_n := \begin{cases} \theta_0 & \text{if } n = 0 \\ \tilde{\theta}_{n-1} - \delta \, \overline{m}_{n-1} & \text{if } n \geqslant 1 \end{cases}, \tag{3.12a}$$

$$\overline{m}_n := \begin{cases} \nabla \ell(\theta_0) & \text{if } n = 0 \\ \beta \overline{m}_{n-1} + (1-\beta) \nabla \ell(\tilde{\theta}_{n-1}) & \text{if } n \geqslant 1 \end{cases}, \tag{3.12b}$$

where we used the same notation $(\tilde{\theta}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$ for the sake of simplicity. Qian (1999) showed that in the limit $\delta \to 0$, $(\overline{m}_n)_{n \in \mathbb{N}_0}$ is analogous to the mass of Newtonian particles moving through a viscous medium, *i.e.* a momentum, hence the name. This adaptation can also be related to evolution strategies ideas (Beyer, 2001; Emmerich et al., 2018), *e.g. cf.* Hansen (2016, Section 2).

**RMS Prop** introduced by Geoffrey Hinton[8]. The motivation is to take into account the fact that the coordinates of the gradients $\nabla \ell(\tilde{\theta}_n)$ can be of very different orders of magnitude, while also significantly varying over the course of the training, thus strongly disrupting the convergence. RMSProp tackles this problem by tracking the

[7] the paper is written in russian.

[8] Surprisingly, this result was not published, but presented for the first time in a course given at the University of Toronto. The slides are available at www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

average (coordinates by coordinates) of the squared gradient and scaling the update of $\tilde{\theta}_n$ according to this magnitude. More precisely, let us fix $0 < \gamma < 1$, $\epsilon > 0$ and $\delta > 0$, and define an additionnal sequence $(\varsigma_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$ inductively defined by: for all $n \in \mathbb{N}_0$

$$\tilde{\theta}_n := \begin{cases} \theta_0 & \text{if } n = 0 \\ \tilde{\theta}_{n-1} - \delta \operatorname{diag}\left(\frac{1}{\sqrt{\varsigma_{n-1}}+\epsilon}\right) \nabla \ell(\tilde{\theta}_{n-1}) & \text{if } n \geqslant 1 \end{cases}, \tag{3.13a}$$

$$\varsigma_n := \begin{cases} \left(\nabla \ell(\theta_0)\right)^2 & \text{if } n = 0 \\ \gamma \varsigma_{n-1} + (1-\gamma) \left(\nabla \ell(\tilde{\theta}_{n-1})\right)^2 & \text{if } n \geqslant 1 \end{cases}, \tag{3.13b}$$

where square, square root, an inverse functions are to be understood as applied componentwise, and diag is the function that maps a vector $\omega \in \mathbb{R}^P$ to the diagonal matrix $\operatorname{diag}(\omega)$ which diagonal entry $\operatorname{diag}(\omega)_{ii}$ is $\omega_i$ for all $1 \leqslant i \leqslant P$. We also used the same notation $(\tilde{\theta}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$ for the sake of simplicity. Once again, this can be related to evolution strategies ideas, *e.g. cf.* Hansen (2016, Section 3).

These two adaptations have been grouped together with minor adaptation in the **Adam**[9] algorithm introduced by Kingma and Ba (2014), which is today considered as the state of the art in deep learning. For the sake of completeness, let us introduce it below:

**Definition 3.10** (Adam)**.** Let us fix $\theta_0 \in \mathbb{R}^P$, and a loss $\ell : \mathbb{R}^P \to \mathbb{R}$ as in Definition 3.9. Then given $\epsilon, \delta > 0$, and $0 < \gamma, \beta < 1$, the Adam algorithm applied to $\ell$ can be described by the three inductively defined sequences $(\tilde{\theta}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$, $(\overline{m}_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$, $(\varsigma_n) \in \left(\mathbb{R}^P\right)^{\mathbb{N}_0}$: for all $n \in \mathbb{N}_0$

$$\tilde{\theta}_n := \begin{cases} \theta_0 & \text{if } n = 0 \\ \tilde{\theta}_{n-1} - \frac{\delta}{(1-\gamma^n)(1-\beta^n)} \operatorname{diag}\left(\frac{1}{\sqrt{\varsigma_{n-1}}+\epsilon}\right) \overline{m}_{n-1} & \text{if } n \geqslant 1 \end{cases}, \tag{3.14a}$$

$$\overline{m}_n := \begin{cases} \nabla \ell(\theta_0) & \text{if } n = 0 \\ \beta \overline{m}_{n-1} + (1-\beta) \nabla \ell(\tilde{\theta}_{n-1}) & \text{if } n \geqslant 1 \end{cases}, \tag{3.14b}$$

$$\varsigma_n := \begin{cases} \left(\nabla \ell(\theta_0)\right)^2 & \text{if } n = 0 \\ \gamma \varsigma_{n-1} + (1-\gamma) \left(\nabla \ell(\tilde{\theta}_{n-1})\right)^2 & \text{if } n \geqslant 1 \end{cases}, \tag{3.14c}$$

where square, square root, an inverse functions are to be understood as applied componentwise, and diag is the the function that maps a vector $\omega \in \mathbb{R}^P$ to the diagonal matrix $\operatorname{diag}(\omega)$ which diagonal entry $\operatorname{diag}(\omega)_{ii}$ is $\omega_i$ for all $1 \leqslant i \leqslant P$.

It must also be mentioned that hyperparameters $\delta$, $\beta$ and $\gamma$ in the above definition can also be adapted, *i.e.* not considered as constants anymore, but rather as sequences. Such a process is known as "scheduling". Numerous heuristics are available for this purpose, including some making $(\delta_n), (\beta_n), (\gamma_n) \in \mathbb{R}^{\mathbb{N}}$ dependent from $\tilde{\theta}_n$, $(\overline{m}_n)$ and $(\varsigma_n)$, see for instance Li and Arora (2019); Kim et al. (2021); Lewkowycz (2021).

One may also wonder whether refining to a higher order the Taylor expansion of Equation (3.10), could gives us a better approximation of the theoretical gradient flow. The

[9] which stands for "adaptive moment estimation", although it's not strictly speaking an acronym.

answer is obviously yes, but the problem comes from the fact that the number of parameters required to calculate higher-order differentials grows exponentially with the order of the approximation. Indeed, each $k$-order differential of $\ell$, $k \in \mathbb{N}_1$, evaluated at a point $\omega \in \mathbb{R}^P$ can be assimilated to a $k$-linear application from $\mathbb{R}^P$ to $\mathbb{R}$, *i.e.* belonging to a space of dimension $P^k$, which is rhedibitory since $P$ is itself already very large[10]. However in the case where the order of $P$ stays under $10^4$, the order 2 derivative, *i.e.* the Hessian, can still be approximated. This is the purpose of the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm (Fletcher, 2000, Section 3.4 page 62), which is inspired by the Newton's method (Fletcher, 2000, Section 3.1 page 44). Liu and Nocedal (1989) introduced a modification of this algorithm that makes it able to scale to higher dimensions, which makes it suitable to train neural networks.

We conclude with a few words on the choice of $\theta_0 \in \mathbb{R}^P$. An extensive literature exists on this topic and its consequences, see for instance Kumar (2017); Narkhede et al. (2022). In particular the link between the statistical distribution choosed to initialize the coordinate of $\theta_0$ and the asymptotic convergence of the neural network function $\Psi(\theta_0)$ to a Gaussian process when $P \to +\infty$, has been known since the seminal work of Neal (1996); Williams (1996). More recently, Jacot et al. (2018) showed that under commonly satisfied assumptions on the statistical distribution of the coordinate of $\theta_0$, the neural network's training by (stochastic) gradient descent (SGD) asymptotically converges when $P \to +\infty$ to a kernel regression according to the so-called Neural Tangent Kernel (NTK), briefly presented in Appendix D.1 of Appendix D. Nevertheless, in the context of this work, we will stick to the so-called "Glorot-normal" and "Glorot uniform"[11] statistical initialization introduced by Glorot and Bengio (2010), and considered to be state-of-the-art. Namely, using the terms of Definition 3.2, let us fix an architecture $\Psi$ built unpon a map $\psi$ assumed to be just a reparametrization, such that any $\theta_p$ for $1 \leqslant p \leqslant P$ is unambiguously mapped to a weight in $\mathcal{M}_{l_{i-1},l_i}$ or a bias in $\mathbb{R}^{l_i}$ for $1 \leqslant i \leqslant d$. Then: for all $1 \leqslant p \leqslant P$

**Glorot normal** initializes $\theta_{0p}$ by sampling from the normal distribution:

$$\theta_{0p} \sim \mathcal{N}\left(0, \frac{2}{l_{i-1} + l_i}\right),$$

if $\theta_{0p}$ is mapped by $\psi$ to a weight in $\mathcal{M}_{l_{i-1},l_i}$ and sets:

$$\theta_{0p} = 0,$$

if $\theta_{0p}$ is mapped by $\psi$ to a bias.

**Glorot uniform** initializes $\theta_{0p}$ by sampling from the uniform distribution:

$$\theta_{0p} \sim \mathcal{U}\left(-\sqrt{\frac{6}{l_{i-1} + l_i}}, \sqrt{\frac{6}{l_{i-1} + l_i}}\right),$$

if $\theta_{0p}$ is mapped by $\psi$ to a weight in $\mathcal{M}_{l_{i-1},l_i}$ and sets:

$$\theta_{0p} = 0,$$

if $\theta_{0p}$ is mapped by $\psi$ to a bias.

In the next section, we present the automatic differentiation mechanism that allows to effectively implement the gradient descent algorithm, presented in this section.

[10] of the order of $10^3$ for the simplest architectures, and up to $10^{10}$ or even $10^{11}$ for architectures such as transformers.

[11] sometimes also refered as "Xavier normal" and "Xavier uniform", both relating to Xavier Glorot.

### 3.2.2. Automatic differentiation of neural networks

In this section, we will present the automatic differentiation method also called algorithmic differentiation or simply "autodiff", strongly following Baydin et al. (2018). Automatic differentiation is a set of techniques designed for near exact[12] numerical differentiation. Methods for such numerical differentiation can be classified into four categories: "by hand" differentiation, finite difference approximations, symbolic differentiation, and automatic differentiation.

Working out derivatives by hand is very time-consuming, prone to many hard to detect errors, and should therefore be avoided.

Symbolic differentiation can be seen as an automation of the former, built upon a small set of simple computing rules[13] guaranteeing no calculation errors. Nevertheless they often result in exponentially large symbolic expressions which are both cryptic and costly to evaluate, such a problem being known as "expression swell" (Corliss, 1988). What's more, they need close form expressions for the functions to differentiate, which can challenging for functions that are only available as an algorithmic procedure.

On the other hand, finite difference approximations estimate a partial derivative of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ in a direction $r \in \mathbb{R}^n$ evaluated at $x \in \mathbb{R}^n$ by coming back to the difference quotient in its limit definition, *i.e.* by setting $\partial_r f(x) \simeq \frac{f(x+hr)-f(x)}{h}$ or $\partial_r f(x) \simeq \frac{f(x+hr)-f(x-hr)}{2h}$ with $h \in (0, 1]$ small enough. In comparison with symbolic differentiation, this method scales linearly in the size of the Jacobian matrix (*i.e.* $nm$ here) and is therefore relatively cheap to compute. Moreover, it only needs to evaluate $f$, which gives more flexibility to the expression of $f$. It is withal subject to inherently ill-conditioning and instabilities, due to truncation[14] and round-off errors, the later increasing when the former decreases (Jerrell, 1997).

Automatic differentiation tries to take the best of two words: the exactness of symbolic differentiation and the flexibility and simplicity of finite difference approximations. Built over the observation that all numerical computations performed by an algorithm can be ultimately decomposed as a succession of elementary operations whose derivatives are known (Griewank and Walther, 2008), the core idea of automatic differentiation is then to double the computations conducted by the algorithm with derivatives computations, combining known derivatives of elementary operations through simple differentiation rules[13], echoing in that symbolic computation although being quite different, since the derivatives computations are not performed on abstract variables, but on specific instances of it as for finite difference approximations. This allows to evaluate derivatives not only on closed form formulas, but on general algorithmic operations, even if they include branching (*i.e.* conditional statements) or loops, while preserving near exactness[12] of the computation. An illustration of the differences between those four methods, borrowed from Baydin et al. (2018), can be found in Figure 28 in Appendix C.

In more concrete terms, let us give two differentiable functions $g : \mathbb{R} \to \mathbb{R}^m$ and $\rho : \mathbb{R} \to \mathbb{R}$ and set $f : x \in \mathbb{R}^n \mapsto \left(\sum_{i=1}^n \rho(x_i)\right) g(x_n) \in \mathbb{R}^m$. Then, given $x \in \mathbb{R}^n$ an evaluation point, and $r \in \mathbb{R}^n$ a partial differentiation direction or $b \in \mathbb{R}^m$ a "partial differentiation codirection", automatic differentiation of $f$ can be carried out in two different ways: in forward mode to compute $\partial_r f$ or in reverse (or backward) mode to compute $Jf(x)^T b$ with $Jf(x)$ the Jacobian matrix of $f$ evaluated at $x$ and $^T$ the matrix transposition.

[12] up to round-off errors that one gets when performing a computation using bits representations, which have accuracy limitations and do not deal well in particular with multi scale information.

[13] Leibniz product rule, chain rule, differentiation of polynomials and classical functions, *etc.*.

[14] *i.e.* the inaccuracy, one gets from $h$ not actually being zero.

**Forward mode** performs the derivative computation by initializing a differentiation variable $\dot{x} = r$ and then computing the derivatives of the operations performed as they come with the help of the chain rule as shown in Table 12 below.

| Sequence of operations to compute $f(x)$ | | |
|---|---|---|
| $v_{-n+i}$ | $= x_i$ | $1 \leqslant i \leqslant n$ |
| $v_i$ | $= \rho(v_{-n+i})$ | $1 \leqslant i \leqslant n$ |
| $v_{n+1}$ | $= \sum_{i=1}^{n} v_i$ | |
| $v_{n+1+j}$ | $= g(v_0)_j$ | $1 \leqslant j \leqslant m$ |
| $v_{n+1+m+j}$ | $= v_{n+1+j} v_{n+1}$ | $1 \leqslant j \leqslant m$ |
| $\boldsymbol{f(x)_j}$ | $= \boldsymbol{v_{n+1+m+j}}$ | $1 \leqslant j \leqslant m$ |

| Sequence of operations to compute $\partial_r f(x)$ | | |
|---|---|---|
| $\dot{v}_{-n+i}$ | $= r_i$ | $1 \leqslant i \leqslant n$ |
| $\dot{v}_i$ | $= \rho'(v_{-n+i})\dot{v}_{-n+i}$ | $1 \leqslant i \leqslant n$ |
| $\dot{v}_{n+1}$ | $= \sum_{i=1}^{n} \dot{v}_i$ | |
| $\dot{v}_{n+1+j}$ | $= g'(v_0)_j \dot{v}_0$ | $1 \leqslant j \leqslant m$ |
| $\dot{v}_{n+1+m+j}$ | $= \dot{v}_{n+1+j} v_{n+1} + v_{n+1+j} \dot{v}_{n+1}$ | $1 \leqslant j \leqslant m$ |
| $\partial_{\boldsymbol{r}} \boldsymbol{f(x)_j}$ | $= \boldsymbol{\dot{v}_{n+1+m+j}}$ | $1 \leqslant j \leqslant m$ |

Table 12: Forward mode of automatic differentiation applied to $f : x \in \mathbb{R}^n \mapsto \left(\sum_{i=1}^{n} \rho(x_i)\right) g(x_n) \in \mathbb{R}^m$ with given $g : \mathbb{R} \to \mathbb{R}^m$ and $\rho : \mathbb{R} \to \mathbb{R}$, for which respective derivatives functions $g'$ and $\rho'$ are known (adapted from Baydin et al. 2018).
The left table keeps track of the intermediate operations performed to actually compute $f(x)$ in the variables $(v_k)_{k=1-n}^{n+1+2m}$. The right table shows how the computation of $\partial_r f(x)$ for a given $r \in \mathbb{R}^n$ can be easily processed by instantiating the variables $(\dot{v}_k)_{k=1-n}^{n+1+2m}$ and using them to track differentiation operations, built iteratively starting from a partial differentiation direction $r \in \mathbb{R}^n$, and then worked out following differentiation rules in conjunction with the values recorded in variables $(v_k)_{k=1-n}^{n+1+2m}$ and derivatives functions $g'$ and $\rho'$.

As a consequence, this mode is very convenient when $n \ll m$, since in the extreme case where $n = 1$, the derivative of $f$ can be obtained after applying the above procedure[15] only once. In contrast, when $n \gg m$, obtaining the whole Jacobian of $f$ implies applying the above procedure $n$ times (iterating $r$ over the canonical basis vectors $(e_i)_{i=1}^{n}$ of $\mathbb{R}^n$), which makes it inefficient, in particular in the extreme case where $m = 1$. Unfortunately, the former is precisely the case encountered in the gradient descent algorithm presented in Section 3.2.1, since applying the gradient descent of Definition 3.9 involves computing the differential of $\ell : \mathbb{R}^P \to \mathbb{R}$. Fortunately, the reverse mode presented below bridges this gap.

**Reverse (or backward) mode** is less intuitive than the forward mode, since it does not perform differentiation operations concomitantly with the evaluations of $g$, $\rho$ and ultimately $f$. Instead it keeps track of the elementary operations performed ($g$ and $\rho$ evaluation at $x_n$ and each $(x_i)_{i=1}^{n}$, respectively, summation of the $(\rho(x_i))_{i=1}^{n}$, and finally multiplication of $g(x_1)$ by $\sum_{i=1}^{n} \rho(x_i)$), while computing partial derivatives of each elementary operation with respect to the following when they are related, thanks to the backward chain rule and the derivatives of elementary operations. Once the evaluation of $f$ at $x$ has been performed, it finally computes the derivatives in reverse (also called backward) direction[16] following the tracked elementary operations dependencies, starting from the "partial differentiation codirection" $b$. This procedure is illustrated in Table 13 below.

[15] called forward pass, hence mode's name.
[16] hence mode's name

Sequence of operations to compute $f(x)$

| | | |
|---|---|---|
| $v_{-n+i}$ | $= x_i$ | $1 \leqslant i \leqslant n$ |
| $v_i$ | $= \rho(v_{-n+i})$ | $1 \leqslant i \leqslant n$ |
| $v_{n+1}$ | $= \sum_{i=1}^{n} v_i$ | |
| $v_{n+1+j}$ | $= g(v_0)_j$ | $1 \leqslant j \leqslant m$ |
| $v_{n+1+m+j}$ | $= v_{n+1+j} v_{n+1}$ | $1 \leqslant j \leqslant m$ |
| $\boldsymbol{f(x)_j}$ | $= \boldsymbol{v_{n+1+m+j}}$ | $1 \leqslant j \leqslant m$ |

Sequence of operations to compute $Jf(x)^T b$

| | | |
|---|---|---|
| $\boldsymbol{Jf(x)^T b_i}$ | $= \boldsymbol{\overline{v}_{-n+i}}$ | $1 \leqslant i \leqslant n$ |
| $\overline{v}_{-n+i}$ | $= \overline{v}_i \frac{\partial v_i}{\partial v_{-n+i}}$ | $1 \leqslant i \leqslant n$ |
| | $= \overline{v}_i \rho'(v_{-n+i})$ | |
| $\overline{v}_0$ | $= \sum_{j=1}^{m} \overline{v}_{n+1+j} \frac{\partial v_{n+1+j}}{\partial v_0} + \overline{v}_n \frac{\partial v_n}{\partial v_0}$ | |
| | $= \sum_{j=1}^{m} \overline{v}_{n+1+j} g'(v_0)_n + \overline{v}_n \rho'(v_0)$ | |
| $\overline{v}_i$ | $= \overline{v}_{n+1} \frac{\partial v_{n+1}}{\partial v_i}$ | $1 \leqslant i \leqslant n$ |
| | $= \overline{v}_{n+1}$ | |
| $\overline{v}_{n+1}$ | $= \sum_{j=1}^{m} \overline{v}_{n+1+m+j} \frac{\partial v_{n+1+j}}{\partial v_{n+1}}$ | |
| | $= \sum_{j=1}^{m} \overline{v}_{n+1+m+j} v_{n+1+j}$ | |
| $\overline{v}_{n+1+j}$ | $= \overline{v}_{n+1+m+j} \frac{\partial v_{n+1+m+j}}{\partial v_{n+1+j}}$ | $1 \leqslant j \leqslant m$ |
| | $= \overline{v}_{n+1+m+j} v_{n+1}$ | |
| $\boldsymbol{\overline{v}_{n+1+m+j}}$ | $= \boldsymbol{b_j}$ | $1 \leqslant j \leqslant m$ |

Table 13: Reverse mode of automatic differentiation applied to $f : x \in \mathbb{R}^n \mapsto (\sum_{i=1}^{n} \rho(x_i))\, g(x_n) \in \mathbb{R}^m$ with given $g : \mathbb{R} \to \mathbb{R}^m$ and $\rho : \mathbb{R} \to \mathbb{R}$, for which respective derivatives functions $g'$ and $\rho'$ are known (adapted from Baydin et al. 2018).
The left table keeps track of the intermediate operations performed to actually compute $f(x)$ in the variables $(v_k)_{k=1-n}^{n+1+2m}$. The right table shows how the computation of $Jf(x)^T b$ for a given $b \in \mathbb{R}^m$ can be easily processed by storing the depencies between the variables $(v_k)_{k=1-n}^{n+1+2m}$ tracked in the left table while computing their relative derivatives $\frac{\partial v_{i+k}}{\partial v_i}$ for $1-n \leqslant i < i+k \leqslant n+1+2m$ when $v_{i+k}$ is related to $v_i$ in the left table computations, following differentiation rules in conjunction with the values recorded in variables $(v_k)_{k=1-n}^{n+1+2m}$ and derivatives functions $g'$ and $\rho'$. Once the left table operations are done, one can the iteratively compute variables $(\overline{v}_k)_{k=1-n}^{n+1+2m}$ starting from a partial differentiation codirection $b \in \mathbb{R}^m$, and then working out following differentiation rules in conjunction with the values recorded in variables $\frac{\partial v_{i+k}}{\partial v_i}$ for $1-n \leqslant i < i+k \leqslant n+1+2m$ when $v_{i+k}$ is related to $v_i$.

As stated above, this mode is thus very convenient when $n \gg m$, since in the extreme case where $m = 1$, the gradient of $f$ can be obtained after applying the above procedure only once by setting $b = 1$. Conversely, when $n \ll m$, obtaining the whole Jacobian of $f$ implies applying the procedure described above $m$ times (iterating $b$ over the canonical basis vectors $(\ell_i)_{i=1}^{m}$ of $\mathbb{R}^m$), which makes it inefficient, in particular in the extreme case where $n = 1$.

Beyond gradient descent, presented in Section 3.2.1, for which it was first introduced in deep learning[17], automatic differentiation can also be used to compute the derivatives of the outputs of a neural network with respect to its inputs. Furthermore, as this operation is also a numerical computation, it can in turn be differentiated with the same procedure. This is the applicative cornerstone of Physics Informed Neural Networks (PINNs). It's also worth noting that beyond numerical results, that is the main idea added by Raissi et al. (2019) in comparison to Lagaris et al. (1998), which no doubt explains why interest in PINNs only really took off from then on. More precisely, if $D$ or $B$ of Definition 3.8

[17] the combination of gradient descent and automatic differentiation has become so popular that it now bears its own name : back-propagation.

are differential operators, they can be computed on the image evaluated at some $\theta \in \mathbb{R}^P$ of an architecture $\psi : \mathbb{R}^P \to \mathcal{F}(\mathbb{R}^n \to \mathbb{R}^m)$ of Definition 3.2, evaluated itself at some point $x \in \Omega$. Then the result of this operation can be in turn differentiated with respect to $\theta$. In particular we could then apply back-propagation to $\mathcal{L} \circ \psi$ with $\mathcal{L}$ being the theoretical PINNs loss of Definition 3.8, but this is not possible as it stands, since $\mathcal{L}$ involves calculating integrals. Approximating such integrals is precisely the aim of the next section.

### 3.2.3. Loss approximation

In this section, we will discuss the approximation of the Loss defined in Equation (3.5), so that it can actually be implemented on a computer. For the sake of clarity, let us recall Equation (3.5):

$$\mathcal{L}_{\text{PINN}}(f) := \gamma_D \left\| D(f) \right\|^2_{\mathrm{L}^2_{m_D}(\Omega,\lambda_n)} + \gamma_B \left\| B(f) \right\|^2_{\mathrm{L}^2_{m_B}(\partial\Omega,\sigma_n)}.$$

By definition, this can be rewritten as:

$$\mathcal{L}_{\text{PINN}}(f) := \gamma_D \int_\Omega \left\| D(f)(x) \right\|^2_{2,m_D} \lambda_n(\mathrm{d}x) + \gamma_B \int_{\partial\Omega} \left\| B(f)(x) \right\|^2_{2,m_B} \sigma_n(\mathrm{d}x). \tag{3.15}$$

As mentioned, we assume that operators $D$ and $B$ are indeed computable, which will be the case for differentiable operators, thanks to automatic differentiation presented above in Section 3.2.2. The norms $\|\cdot\|_{2,m_D}$ and $\|\cdot\|_{2,m_B}$ are Euclidean norms on $\mathbb{R}^{m_D}$ and $\mathbb{R}^{m_B}$ respectively and are therefore computable. The only non-computable operations are then the two integrals.

While numerical integration is an extensively studied subject (Davis and Rabinowitz, 2007), most if not all approximations used for losses, in the context of deep learning, are based on Monte Carlo or quasi-Monte Carlo methods (Lemieux, 2009). In a nutshell, given a measurable space $A$ equipped with a finite-measure $\mu$ and an integrable function $f : A \to \mathbb{R}$, (quasi-)Monte Carlo methods approximates the integral

$$\int_A f(x)\mu(\mathrm{d}x),$$

by sampling $k \in \mathbb{N}_1$ points $(X_i)_{i=1}^k$ from the probability distribution $\mathbb{P}_\mu := \frac{\mu}{\mu(A)}$ and then stating

$$\int_A f(x)\mu(\mathrm{d}x) \simeq \frac{\mu(A)}{k} \sum_{i=1}^k f(X_i).$$

This is justified by the law of large numbers. Indeed, given $X$ a random variable following the law given by $\mathbb{P}_\mu$, and $(X_i)_{i\in\mathbb{N}_1}$ independent and identically distributed random variables also following the law given by $\mathbb{P}_\mu$, we have:

$$\mathbb{P}\left( \lim_{k\to+\infty} \frac{1}{k} \sum_{i=1}^k f(X_i) = \mathbb{E}\left[f(X)\right] = \int_A f(x)\, \mathbb{P}_\mu(\mathrm{d}x) = \frac{1}{\mu(A)} \int_A f(x)\, \mu(\mathrm{d}x) \right) = 1$$

There are two strong arguments advocating for the use of (quasi-)Monte Carlo methods in deep learning:

1. for most deep learning applications, only a certain amount of data is available for a given probability distribution, which are usually data coming from experiments. Therefore Monte Carlo methods allows to approximate the loss integral with the available data, while giving reasonable hope that this will indeed be a good approximation of the theoretical loss.

2. Monte Carlo's methods are "computationally cheap", *i.e.* they only involve operations that are easy and fast to perform on a computer. Indeed, aside the computation of $f$ which is generally just a norm calculated on empirical data, there is only a sum to perform, which is computationally inexpensive and easy to parallelize, followed by scalar multiplications to apply to the sum obtained, which is also very fast.

3. (quasi-)Monte Carlo methods have a slower convergence speed than other classical quadrature methods, as for instance Gaussian quadrature. Nevertheless they are independent of the space dimension, which makes them particularly suitable for most deep learning applications, where the dimension of the "data space" can be very large (for instance of the order of one milllion for images).

Since $\overline{\Omega}$ is compact, $\lambda_n(\Omega) < +\infty$ and $\sigma_n(\partial\Omega) < +\infty$. We can thus apply a Monte Carlo approximation to the loss (3.15). Given samples $(X_{D,i})_{i=1}^{m_D}$, $(X_{B,i})_{i=1}^{m_B}$ sampled from $\mathbb{P}_{\lambda_n} := \frac{\lambda_n}{\lambda_n(\Omega)}$ and $\mathbb{P}_{\sigma_n} := \frac{\sigma_n}{\sigma_n(\partial\Omega)}$ respectively, the loss (3.15), rewrites as:

$$\mathcal{L}_{m_D,m_B}(f) := \frac{\gamma_D\lambda_n(\Omega)}{m_D}\sum_{i=1}^{m_D}\|D(f)(X_{D,i})\|_2^2 + \frac{\gamma_B\sigma_n(\partial\Omega)}{m_B}\sum_{i=1}^{m_B}\|B(f)(X_{B,i})\|_2^2$$

We will state that in a definition, setting for convenience $\tilde{\gamma}_D := \gamma_D\lambda_n(\Omega)$ and $\tilde{\gamma}_B := \gamma_B\sigma_n(\partial\Omega)$.

**Definition 3.11** (Empirical loss of PINNs)**.** Let be $f \in \mathcal{F}_{\text{NN}}$ with $\mathcal{F}_{\text{NN}}$ as in Definition 3.7 and $m_D, m_B \in \mathbb{N}_1$. The $(m_D, m_B)$-empirical loss of PINNs is then defined as:

$$\mathcal{L}_{m_D,m_B}(f) := \frac{\tilde{\gamma}_D}{m_D}\sum_{i=1}^{m_D}\|D(f)(X_{D,i})\|_2^2 + \frac{\tilde{\gamma}_B}{m_B}\sum_{i=1}^{m_B}\|B(f)(X_{B,i})\|_2^2, \tag{3.16}$$

where $\tilde{\gamma}_D$, $\tilde{\gamma}_B > 0$ and $(X_{D,i})_{i=1}^{m_D}$, $(X_{B,i})_{i=1}^{m_B}$ are sampled from $\mathbb{P}_{\lambda_n} := \frac{\lambda_n}{\lambda_n(\Omega)}$ and $\mathbb{P}_{\sigma_n} := \frac{\sigma_n}{\sigma_n(\partial\Omega)}$ respectively.

*Remark* 3.11. Strictly speaking, the loss $\mathcal{L}_{m_D,m_B}$ of Definition 3.11 is a random variable, built upon $(X_{D,i})_{i=1}^{m_D}$ and $(X_{B,i})_{i=1}^{m_B}$. Nevertheless, for empirical applications, one works with an actual outcome of this random variable, or equivalently with actual outcomes of $(X_{D,i})_{i=1}^{m_D}$ and $(X_{B,i})_{i=1}^{m_B}$, which makes effective evaluation possible.

*Remark* 3.12. We mentionned in Remark 3.10 in Section 3.2.1, that the gradient descent used in deep learning was a slightly modified version of the one presented here, with the modification coming from the loss. To explain this, let us fix outcomes $(y_{i,D})_{i=1}^{m_D}$ and $(y_{i,B})_{i=1}^{m_B}$ from $(X_{D,i})_{i=1}^{m_D}$ and $(X_{B,i})_{i=1}^{m_B}$, respectively, as explained in Remark 3.11. The Stochastic Gradient Descent (SGD) algorithm consists then in applying the Gradient Descent algorithm (or any of the variants described in Section 3.2.1) to a modified version of the loss $\mathcal{L}_{m_D,m_B}$ using only a subset of the outcomes $(y_{i,D})_{i=1}^{m_D}$ and $(y_{i,B})_{i=1}^{m_B}$, namely given $S_D$ a subset of $\{1,\dots,m_D\}$ and $S_B$ a subset of $\{1,\dots,m_B\}$: for all $f \in \mathcal{F}_{\text{NN}}$

$$\mathcal{L}_{S_D,S_B}(f) := \frac{\tilde{\gamma}_D}{|S_D|}\sum_{x\in S_D}\|D(f)(x)\|_2^2 + \frac{\tilde{\gamma}_B}{|S_B|}\sum_{x\in S_B}\|B(f)(x)\|_2^2,$$

with for all $S \in \{S_D, S_B\}$, $|S|$ the cardinal of $S$. This change is made because $m_D$, $m_B$ are often of the order of tens of thousands, or even millions, making it impossible to compute $\mathcal{L}_{m_D,m_B}$ in practice. The term stochastic comes from the fact that the sets $S_D$ and $S_B$ are generally chosen at random, but in such a way that all outcomes $(y_{i,D})_{i=1}^{m_D}$ and $(y_{i,B})_{i=1}^{m_B}$ are chosen the same number of times during the learning process.

Now that we have approximated the loss, two potential pitfalls remain: how to the effectively sample points from a given measure in order to effectively implement a Monte Carlo method for integral's quadrature and evaluate the convergence rate of Monte Carlo's quadrature according to number of sampling points.

First point is in fact a field of research in its own right, and goes far beyond the scope of our work. Furthermore, for our specific case, this question does not present any difficulties as it will be shown in Section 4.1.1. For an exhaustive introduction, one may refer to Lemieux (2009).

The second point is precisely the so-called generalization error that is introduced in the next section, along with other theoretical tools designed to study the approximation's errors of neural networks. Results on generalization errors analysis for PINNs will be carefully reviewed.

### 3.2.4. Theoretical tools for error's studies of neural networks

Following Definition 3.7, let us fix some PINN's problem in a Banach space $\mathcal{F}_{\mathcal{NN}}$, equipped with norm $\|\cdot\|_{\mathcal{NN}}$, with some architecture collections dense subspace $\mathcal{A} \in \mathcal{P}^{\infty}_{n,m}(\overline{\Omega})$ and some PINNs loss $\mathcal{L}$ as defined in (3.5). We assume that such a PINN problem admits a solution $u^\star \in \mathcal{F}_{\mathcal{NN}}$. We also fix a subspace $\mathcal{H} \subset \mathcal{A}$ which is assumed to be non dense in $\mathcal{F}_{\mathcal{NN}}$. We want now to provide a coherent framework for discussing approximation errors informally introduced in Remark 3.7. To this end, we introduce the subsequent key's concepts, that are the three components of the total approximation error (see Berner et al. 2021 for a more formal introduction):

**The approximation error** is a quantification of the error made when considering $\hat{h} \in \mathcal{H}$ such that $\mathcal{L}(\hat{h}) = \inf_{f \in \mathcal{H}} \mathcal{L}(f)$ (provided it exists) instead of $u^\star \in \mathcal{F}_{\mathcal{NN}}$, *e.g.* $\left\|\hat{h} - u^\star\right\|_{\mathcal{NN}}$.

This error makes sense, since $\mathcal{H}$ is not assumed to be dense in $\mathcal{F}_{\mathcal{NN}}$ anymore and thus the universal approximation Theorem 3.1 does not hold. Proving general estimates for such approximation errors with neural networks function classes is an active field of research. One could for instance refer to Elbrächter et al. (2021) for a general introduction, to De Ryck et al. (2021) for explicit bounds estimates in high-order Sobolev norms for MLPs with tanh activations and to Schwab and Zech (2021) for a careful review of bounds estimates in $\mathrm{L}^2$ spaces with Gaussian's measures.

**The generalization error** is the error that arises from the quadrature described in Section 3.2.3. More precisely, let us fix $m_D, m_B \in \mathbb{N}_1$, set $m := (m_D, m_B) \in \mathbb{N}_1^2$, and consider the empirical loss $\mathcal{L}_{m_D,m_B}$ of Definition 3.11 rewritten as $\mathcal{L}_m$ for convenience. Then the generalization error is a quantification of the error made when considering $h_m \in \mathcal{H}$ such that $\mathcal{L}_m(h_m) = \inf_{f \in \mathcal{H}} \mathcal{L}_m(f)$ (provided it exists) instead of $\hat{h} \in \mathcal{H}$ such that $\mathcal{L}(\hat{h}) = \inf_{f \in \mathcal{H}} \mathcal{L}(f)$ (provided again it exists), *e.g.* $\left\|h_m - \hat{h}\right\|_{\mathcal{NN}}$.
Generalization errors for deep learning have been intensively studied and is still an active field of study. One could refer to Goodfellow et al. 2016, Chapter 5, Section 2, page 110 for a introduction, and to Jakubovitz et al. (2019) for a survey.

**The optimization error** quantifies the error made when considering some given $\tilde{h}_m \in \mathcal{H}$ instead of $h_m \in \mathcal{A}$ such that $\mathcal{L}_m(h_m) = \inf_{g \in \mathcal{A}} \mathcal{L}_m(g)$ (assuming that such a $h_m$ exists), *e.g.* $\left\| \tilde{h}_m - h_m \right\|_{\mathcal{NN}}$. $\tilde{h}_m$ is meant to be the result returned by some algorithmic optimization procedure, in particular in the context of neural networks, a (stochastic) gradient descent as described in Section 3.2.1. Those optimizations often fail to find a global minimum, due the non-convexity of $\mathcal{L}_{m|\mathcal{H}}$ or even $\mathcal{L}_{|\mathcal{A}}$. Indeed, even if $\mathcal{L}$ is strictly convex, $\mathcal{L}_{|\mathcal{A}}$ is often highly non-convex, due to the highly non-convex structure of $\mathcal{A}$. The same remarks apply to $\mathcal{L}_m$ and $\mathcal{H}$. The understanding of the properties of those losses is known as loss landscape study (Sun et al., 2020), and is an active research domain (Simsek et al., 2021; Cooper, 2018). This is also strongly related to convergence results and dynamics analysis mentioned in Remark 3.6 of Section 3.1.3. We also believe that the preliminary work developed in Appendix D.3, following Jacot et al. (2018), may be an interesting direction to contribute to this problem's study.

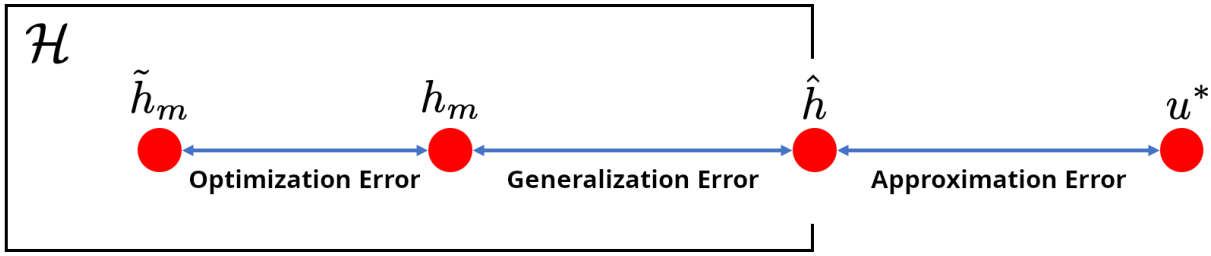



Figure 4: Illustrations of the theoretical total errors of a neural network (adapted from Shin et al. 2020)

Here, $\mathcal{H}$ is meant to be some functional space covered by a neural network's architecture (see Definition 3.2) when varying the weights, which is in general non dense in the considered ambient space $\mathcal{F}_{\mathcal{NN}}$ in which a solution $u^\star \in \mathcal{F}_{\mathcal{NN}}$ to a given PDE exists (see Section 3.1.4). $\hat{h} \in \mathcal{H}$ is a neural network minimizing the theoretical PINN's loss $\mathcal{L}$ (see Definition 3.8), while $h_m \in \mathcal{H}$ is a neural network minimizing the empirical PINN's loss $\mathcal{L}_{m_D,m_B}$ (see Definition 3.11). Finally $\tilde{h}_m \in \mathcal{H}$ is a neural network obtained from an optimization process that did not converge to a global minimum of $\mathcal{L}_{m_D,m_B}$ in $\mathcal{H}$. Thus, the total approximation error made by taking $\tilde{h}_m$ instead of $u^\star$ can be decomposed in three components:

- The approximation error, made when taking $\hat{h}$ instead of $u^\star$, which comes from the "lack of expressivity" of the neural network (or equivalently "to what extent is $\mathcal{H}$ not dense in $\mathcal{F}_{\mathcal{NN}}$").
- The generalization error, made when taking $h_m$ instead of $\hat{h}$, which comes from the quadrature error when tacking the empirical loss instead of the theoretical loss.
- The optimization error, made when taking $\tilde{h}_m$ instead of $\tilde{h}_m$, which comes from the fact that the optimization procedure failed to reach a global minimum of the empirical loss in $\mathcal{H}$.

We illustrate those three different errors in Figure 4. We can now review the few existing results (indicating that more research is needed) concerning the generalization errors for PINNs:

**Shin et al. (2020)** show that the generalization error can be made arbitrarily small when the PINN problem is a linear second-order elliptic or parabolic type PDE having a solution in a Hölder space, under strong precise assumptions on $\Omega$ and the PDE. The approximation error is briefly treated, but relies mostly on the universal approximation Theorem 3.1. As a complement, we detail a proof in Appendix E.

**Mishra and Molinaro (2020)** show precise error estimates in Sobolev's norm for the generalization error in an abstract setup. They also apply these results to three classes of partial differential equations, namely semi-linear parabolic PDEs, viscous scalar conservation laws, and the incompressible Euler equations of fluid dynamics.

# 4. Approximation of the P2D model with PINNs

In this section we present our empirical results. As announced in the introduction, the aim of our work is to show that the Physics Informed Neural Networks (PINNs) methods presented in Section 3 are a viable option to tackle the approximation of the Pseudo-Two Dimensional (P2D) model, presented in Section 2, that emulates the electrochemical behaviour of a lithium ions battery cell through Partial Differential Equations (PDEs). Although we did not obtain complete results, *i.e.* a full resolution of the equations in long timescale (of the order of a thousand seconds), we did obtain two important partial results.

The first concerns the success of the method for intermediate timescales (of the order of a hundred seconds) supported by the input of additional data coming from the LIONSIMBA solver (*cf.* Torchio et al. 2016), located on the borders of the cell sections (*cf.* Notation 2.2 in Section 2.3.1). Those results are carefully presented in Section 4.3. The second partial result concerns the extension of the intermediate timescales results to long timescales (of the order of a thousand seconds), thanks to the extended-PINN method (Jagtap and Karniadakis, 2021).

We finally review the method in a short timescale (in the order of one second), without additional data. Those results are carefully presented in Section 4.4.

In the following, we briefly introduce our experimental framework, reviewing the tools and metric used to asses our results. We then detail some results concerning the approximation of the P2D model by learning from data alone, as it may be done in industrial applications (*e.g. cf.* Sarvi and Masoum 2008; Zhang et al. 2009), and show its limitations, while illustrating the metrics of our experimental framework.

## 4.1. Presentation of the experimental framework

In this section, we present the different tools and metrics that we used to produce and evaluate our experimental results. We start by showing how the equations of the (simplified) P2D model in Section 2 can be reformulated into the PINNs problem framework of Section 3.1.4, before presenting the LIONSIMBA library to which we will compare ourselves, and finally introducing the different metrics used to evaluate our results. The hardware, softwares and libraries used and their functionalities are reviewed in Appendix F.1.

### 4.1.1. Reformulation of the (simplified) P2D model into a PINNs framework

Following the theoretical loss:

$$\mathcal{L}_{\text{PINN}}(f) := \gamma_D \left\| D(f) \right\|^2_{\text{L}^2_{m_D}(\Omega,\lambda_n)} + \gamma_B \left\| B(f) \right\|^2_{\text{L}^2_{m_B}(\partial\Omega,\sigma_n)},$$

of Definition 3.8 in Section 3.1.4 and its approximated version:

$$\mathcal{L}_{m_D,m_B}(f) := \frac{\tilde{\gamma}_D}{m_D} \sum_{i=1}^{m_D} \left\| D(f)(X_{D,i}) \right\|_2^2 + \frac{\tilde{\gamma}_B}{m_B} \sum_{i=1}^{m_B} \left\| B(f)(X_{B,i}) \right\|_2^2,$$

of Definition 3.11 in Section 3.2.3, we have to define the domain $\Omega$ and the operators $D$ and $B$ according to the simplified P2D model. Let us start by defining $\Omega$.

$\Omega$ is the union of the interiors of each of the cell sections (*cf.* Table 1) over time. It is then defined as:

$$\Omega := \bigcup_{i\in\{a,p,s,n,z\}} (\hat{x}_i, x_i) \times [0, T_{\max}], \tag{4.1}$$

where for all $i \in \{a, p, s, n, z\}$ $\hat{x}_i$ and $x_i$ are the cell sections ends (*cf.* Table 2) and $T_{\max}$ is the scope of time in which the model is considered (*cf.* Table 3). Note that $\Omega$ has closure $[x_0, x_z] \times [0, T_{\max}]$, which is compact. Thus, we have that $\Omega$ is relatively compact and therefore of finite Lebesgue measure, and that its boundary is:

$$\partial\Omega = [x_0, x_z] \times \{0\} \bigcup \left( \bigcup_{i \in \{0,a,p,s,n,z\}} \{x_i\} \times [0, T_{\max}] \right) \bigcup [x_0, x_z] \times \{T_{\max}\}, \tag{4.2}$$

which is also of finite measure for the surface measure. Before going any further, let's introduce the following notations for the sake of simplicity:

*Notation* 4.1 (Subdomains of $\Omega$).

- for all $i \in \{a, p, s, n, z\}$, $\Omega_i := (\hat{x}_i, x_i) \times [0, T_{\max}]$ will design the inner domain of section $i$.
- $\partial\Omega^{bc} := \bigcup_{i \in \{0,a,p,s,n,z\}} \{x_i\} \times [0, T_{\max}]$ and for all $i \in \{0, a, p, s, n, z\}$, $\partial\Omega_i^{bc} := \{x_i\} \times [0, T_{\max}]$ will design the domain of the interface between two consecutive sections at coordinate $x_i$. See Table 2 for more details.
- $\partial\Omega^{ic} := [x_0, x_z] \times \{0\}$ and for all $i \in \{a, p, s, n, z\}$, $\partial\Omega_i^{ic} := [\hat{x}_i, x_i] \times \{0\}$ will design the domain of the intial conditions for section $i$.

*Notation* 4.2 (Main variables of section). For all $i \in \{a, p, s, n, z\}$, the notations MV($i$) (for Main Variables) will designate the main variables that are defined on cell section $i$, namely:

- $\text{MV}(a) = \text{MV}(z) = \{T\}$,
- $\text{MV}(p) = \text{MV}(n) = \{T, C_e, C_s^\star, \phi_e, \phi_s\}$,
- $\text{MV}(s) = \{T, C_e, \phi_e\}$,

*Notation* 4.3 (Cell sections domain of variable). We also introduce the "dual" notation for all $u \in \{T, C_e, C_s^\star, \phi_e, \phi_s\}$, SD($u$) (for Sections Domain) that designate the cell sections on which $u$ is defined, namely:

- $\text{SD}(T) = \{a, p, s, n, z\}$,
- $\text{SD}(C_e) = \text{SD}(\phi_e) = \{p, s, n\}$,
- $\text{SD}(C_s^\star) = \text{SD}(\phi_s) = \{p, n\}$,

Let us now define the operators $B$ and $D$. To this end, we need to rewrite each of the equations of Section 2 by subtracting from both sides of the equality all the elements on the right of the equal sign, so as to obtain an equation with one side equal to zero. For instance, Equation (2.3a) on $a$: for all $(x, t) \in \Omega_a$

$$\rho_a C_{p,a} \partial_t T(x,t) = \lambda_a \partial_{x^2} T(x,t) + \frac{I_{app}^2(t)}{\sigma_{\text{eff},a}}$$

rewrites: for all $(x, t) \in \Omega_a$

$$\rho_a C_{p,a} \partial_t T(x,t) - \lambda_a \partial_{x^2} T(x,t) - \frac{I_{app}^2(t)}{\sigma_{\text{eff},a}} = 0.$$

Now, to each section $i \in \{a, p, s, n, z\}$ of the cell, we attach an operator:

$$D_i : \mathcal{C}^2\left(\Omega_i \to \mathbb{R}\right)^{m_i} \to \mathcal{C}^0\left(\Omega_i \to \mathbb{R}\right)^{m_i}, \tag{4.3}$$

where $m_i \in \mathbb{N}_1$ is the number of equations defined on section $i$, each coordinate of the image of $D_i$ corresponding to the left hand side of the rewritten equations. For instance, the differential operator $D_a$ attached to section $a$, is the operator:

$$D_a : \left\{ \begin{array}{rcl} \mathcal{C}^2\left(\Omega_a \to [0, +\infty)\right)^1 & \longrightarrow & \mathcal{C}^0\left(\Omega_a \to [0, +\infty)\right)^1 \\ T & \longmapsto & \left(\rho_a C_{p,a} \partial_t T - \lambda_a \partial_{x^2} T - \frac{I_{app}^2}{\sigma_{\text{eff},a}}\right) \end{array} \right. . \tag{4.4}$$

*Remark* 4.1. As one may have noticed, we have defined the $D_i$ operators as operating on $\mathcal{C}^2\left(\Omega_i \to \mathbb{R}\right)^{m_i}$, but we have written that the $D_a$ operates on $\mathcal{C}^2\left(\Omega_a \to [0, +\infty)\right)^1$. We have taken this shortcut on the range of the functions so as not to make the notations even more cumbersome. See "Range" column of Table 4 for details.

*Remark* 4.2. We cannot define an independent operator for each equation, since the equations on each section are coupled to each other.

Note that we have:

- $m_a = 1$, since the only equation defined on $a$ is (2.3a).
- $m_p = 5$, accounting for Equations (2.4a), (2.6a), (2.8a), (2.12c) and (2.13a).
- $m_s = 3$, accounting for Equations (2.5a), (2.7a) and (2.9a).
- $m_n = 5$, accounting for Equations (2.4a), (2.6a), (2.8a), (2.12c) and (2.13a).
- $m_z = 1$, since the only equation defined on $z$ is (2.3a).

Having done so, we can finally define the operator $D$ by:

$$D : \left\{ \begin{array}{rcl} \mathcal{C}^2\left(\Omega \to \mathbb{R}\right)^5 & \longrightarrow & \displaystyle\prod_{i \in \{a,p,s,n,z\}} \mathcal{C}^0\left(\Omega_i \to \mathbb{R}\right)^{m_i} \\ (T, C_e, C_s^\star, \phi_s, \phi_e) & \longmapsto & \left( \begin{array}{l} D_a(T|_{\Omega_a}) \\ D_p(T|_{\Omega_p}, C_e|_{\Omega_p}, C_s^\star|_{\Omega_p}, \phi_e|_{\Omega_p}, \phi_s|_{\Omega_p}) \\ D_p(T|_{\Omega_s}, C_e|_{\Omega_s}, \phi_e|_{\Omega_s}) \\ D_p(T|_{\Omega_n}, C_e|_{\Omega_n}, C_s^\star|_{\Omega_n}, \phi_e|_{\Omega_n}, \phi_s|_{\Omega_n}) \\ D_z(T|_{\Omega_z}) \end{array} \right) \end{array} \right. , \tag{4.5}$$

In a more informal way, $D$ is the operator that maps the five main variables $T, C_e, C_s^\star, \phi_s, \phi_e$ to their restrictions on each section so that $D(T, C_e, C_s^\star, \phi_s, \phi_e) = 0$ is equivalent to the fact $(T, C_e, C_s^\star, \phi_s, \phi_e)$ are solutions to Equations (2.3a), (2.4a), (2.5a), (2.6a), (2.7a), (2.8a), (2.9a), (2.12c) and (2.13a). $B$ is defined in the same way. More precisely, we will define two operators $B^{ic}$ and $B^{bc}$, representing the boundary conditions at the sections interfaces and the initial conditions respectively. We then have:

- for all $i \in \{a, p, s, n, z\}$, $B_i^{ic} : \mathcal{C}^2\left(\partial\Omega_i^{ic} \to \mathbb{R}\right)^{m_i} \to \mathcal{C}^2\left(\partial\Omega_i^{ic} \to \mathbb{R}\right)^{m_i}$ with $m_i$ as defined above. Note that:

  $B_a^{ic}$ is built upon Equation (2.3d).

  $B_p^{ic}$ is built upon Equations (2.4d), (2.6c), (2.8d), (2.12d) and (2.13d).

  $B_s^{ic}$ is built upon Equations (2.5d), (2.7d) and (2.9d).

$B_n^{ic}$ is built upon Equations (2.4d), (2.6c), (2.8d), (2.12d) and (2.13d).
$B_z^{ic}$ is built upon Equation (2.3d).

- for all $i \in \{0, a, p, s, n, z\}$, $B_i^{bc} : \mathcal{C}^2\left(\partial\Omega_i^{bc} \to \mathbb{R}\right)^{m_i^{bc}} \to \mathcal{C}^0\left(\partial\Omega_i^{bc} \to \mathbb{R}\right)^{m_i^{bc}}$ with:
  $m_0^{bc} = 1$ accounting for Equation (2.3b).
  $m_a^{bc} = 4$ accounting for Equations (2.4b), (2.6b), (2.8b) and (2.13c).
  $m_p^{bc} = 4$ accounting for Equations (2.5b), (2.7b), (2.9b) and (2.13b).
  $m_s^{bc} = 4$ accounting for Equations (2.5c), (2.7c), (2.9c) and (2.13b).
  $m_n^{bc} = 4$ accounting for Equations (2.4c), (2.6b), (2.8c) and (2.13c).
  $m_z^{bc} = 1$ accounting for Equation (2.3c).

As for $D$, we then define $B^{ic}$ as:

$$B^{ic} : \begin{cases} \mathcal{C}^2\left(\partial\Omega^{ic} \to \mathbb{R}\right)^5 & \longrightarrow \displaystyle\prod_{i\in\{a,p,s,n,z\}} \mathcal{C}^0\left(\partial\Omega_i^{ic} \to \mathbb{R}\right)^{m_i} \\ (T, C_e, C_s^\star, \phi_s, \phi_e) & \longmapsto \begin{pmatrix} B_a^{ic}(T|_{\partial\Omega_a^{ic}}) \\ B_p^{ic}(T|_{\partial\Omega_p^{ic}}, C_e|_{\partial\Omega_p^{ic}}, C_s^\star|_{\partial\Omega_p^{ic}}, \phi_e|_{\partial\Omega_p^{ic}}, \phi_s|_{\partial\Omega_p^{ic}}) \\ B_s^{ic}(T|_{\partial\Omega_s^{ic}}, C_e|_{\partial\Omega_s^{ic}}, \phi_e|_{\partial\Omega_s^{ic}}) \\ B_n^{ic}(T|_{\partial\Omega_n^{ic}}, C_e|_{\partial\Omega_n^{ic}}, C_s^\star|_{\partial\Omega_n^{ic}}, \phi_e|_{\partial\Omega_n^{ic}}, \phi_s|_{\partial\Omega_n^{ic}}) \\ B_z^{ic}(T|_{\partial\Omega_z^{ic}}) \end{pmatrix} \end{cases}, \tag{4.6}$$

and $B^{bc}$ as:

$$B^{bc} : \begin{cases} \mathcal{C}^2\left(\partial\Omega^{bc} \to \mathbb{R}\right)^4 & \longrightarrow \displaystyle\prod_{i\in\{a,p,s,n,z\}} \mathcal{C}^0\left(\partial\Omega_i^{bc} \to \mathbb{R}\right)^{m_i^{bc}} \\ (T, C_e, \phi_s, \phi_e) & \longmapsto \begin{pmatrix} B_0^{bc}(T|_{\partial\Omega_0^{bc}}) \\ B_a^{bc}(T|_{\partial\Omega_a^{bc}}, C_e|_{\partial\Omega_a^{bc}}, \phi_e|_{\partial\Omega_a^{bc}}, \phi_s|_{\partial\Omega_a^{bc}}) \\ B_p^{bc}(T|_{\partial\Omega_p^{bc}}, C_e|_{\partial\Omega_p^{bc}}, \phi_e|_{\partial\Omega_p^{bc}}, \phi_s|_{\partial\Omega_p^{bc}}) \\ B_s^{bc}(T|_{\partial\Omega_s^{bc}}, C_e|_{\partial\Omega_s^{bc}}, \phi_e|_{\partial\Omega_s^{bc}}, \phi_s|_{\partial\Omega_s^{bc}}) \\ B_n^{bc}(T|_{\partial\Omega_n^{bc}}, C_e|_{\partial\Omega_n^{bc}}, \phi_e|_{\partial\Omega_n^{bc}}, \phi_s|_{\partial\Omega_n^{bc}}) \\ B_z^{bc}(T|_{\partial\Omega_z^{bc}}) \end{pmatrix} \end{cases}. \tag{4.7}$$

Finally, we define $B$ as:

$$B : \begin{cases} \mathcal{C}^2\left(\partial\Omega \to \mathbb{R}\right)^5 & \longrightarrow \mathcal{C}^0\left(\partial\Omega^{ic} \to \mathbb{R}\right)^{15} \times \mathcal{C}^0\left(\partial\Omega^{bc} \to \mathbb{R}\right)^{18} \\ (T, C_e, C_s^\star, \phi_s, \phi_e) & \longmapsto \begin{pmatrix} B^{ic}(T|_{\partial\Omega^{ic}}, C_e|_{\partial\Omega^{ic}}, C_s^\star|_{\partial\Omega^{ic}}, \phi_e|_{\partial\Omega^{ic}}, \phi_s|_{\partial\Omega^{ic}}) \\ B^{bc}(T|_{\partial\Omega^{bc}}, C_e|_{\partial\Omega^{bc}}, \phi_e|_{\partial\Omega^{bc}}, \phi_s|_{\partial\Omega^{bc}}) \end{pmatrix} \end{cases}, \tag{4.8}$$

where $15 = \sum_{i\in\{a,p,s,n,z\}} m_i$, and $18 = \sum_{i\in\{0,a,p,s,n,z\}} m_i^{bc}$.

In order to define loss

$$\mathcal{L}_{m_D,m_B}(f) := \frac{\tilde{\gamma}_D}{m_D}\sum_{i=1}^{m_D} \|D(f)(X_{D,i})\|_2^2 + \frac{\tilde{\gamma}_B}{m_B}\sum_{i=1}^{m_B} \|B(f)(X_{B,i})\|_2^2,$$

of Definition 3.11, all that remains is to define a sampling method for $\Omega$ and $\partial\Omega$.

Given the extremely simple geometry of $\Omega$, we could simply draw $m_D \in \mathbb{N}_1$ numbers $(x_i)_{1\leqslant i\leqslant m_d}$ uniformly in $[x_0, x_z]$ and $m_D$ numbers $(t_i)_{1\leqslant i\leqslant m_d}$ in $[0, T_{\max}]$ and then take the pairs $(x_i, t_i)_{1\leqslant i\leqslant m_d}$. Nevertheless, we're going to modify this procedure slightly, firstly

to have exactly the same number of pairs in each of the sections, *i.e.* in each of the $\Omega_i$ $\forall i \in \{a, p, s, n, z\}$, and secondly by forcing the pairs to be more evenly distributed in each of the $\Omega_i$ by replacing the uniform sampling with the Latin Hypercube Sampling (LHS; *cf.* McKay et al. 2000). The latter is justified by the fact that LHS has been shown to have more suitable statistical properties than uniform sampling for numerical integration with Monte Carlo methods, see for instance Lemieux 2009, page 234. We proceed in exactly the same way for sampling for $B$ on $\partial\Omega$, with a dedicated draw on each of the $\partial\Omega_i^{ic}$ $\forall i \in \{a, p, s, n, z\}$ and $\partial\Omega_i^{bc}$ $\forall i \in \{0, a, p, s, n, z\}$ domains.

We have already pointed out that the loss weights $\gamma_D$ and $\gamma_B$ are of particular importance to train a neural network with loss $\mathcal{L}_{m_D,m_B}$. Nevertheless, we will not discuss this aspect here, but rather illustrate their choice in the following sections devoted to results reviewing.

Now that we have specified how the equations of the sinplified P2D model can be used to build a PINNs loss, we will briefly introduce in the next section, the LIONSIMBA solver for the P2D model, which we will use as a reference to evaluate our results.

### 4.1.2. LIONSIMBA

Introduced by Torchio et al. (2016), LIONSIMBA (Lithium-ION SIMulation BAttery Toolbox) is a solver for the P2D model written in Matlab. It approximates the solutions of the simplified P2D model equations by first reformulating it into a differential-algebraic system of equations (DAE; see for instance Kunkel 2006 for an introduction) and then discretizing the spatial domain (the cell assembly direction, *i.e.* the first variables domain in our model presented in Section 2) using Finite Volume Methods (FVM; see for instance Liu 2018 for a brief introduction). The equations are then solved using a time adaptive DAE solver.

To produce the results presented in this chapter, we set up LIONSIMBA to return linearly interpolated results, with an accuracy of half a second in time, and a discretization of two hundred points per section in space. These parameters have been chosen in order to maximize the accuracy that can be obtained with LIONSIMBA (indeed, we have experimentally observed that LIONSIMBA struggles to converge for an accuracy greater than 200 spatial discretization points). To facilitate the remainder of this presentation, we will introduce the following notations:

*Notation* 4.4 (LIONSIMBA domains and variables).

- for all $i \in \{a, p, s, n, z\}$, $M_{\Omega_i}$ will denote the mesh used by LIONSIMBA to discretize $\Omega_i$, *i.e.* with a precision of half a second in time, and two hundred points evenly distributed along the space direction of section $i$. More formally, we have: for all $i \in \{a, p, s, n, z\}$

$$M_{\Omega_i} := \left\{ \left( \frac{\hat{x}_i + (j - \frac{1}{2})(x_i - \hat{x}_i)}{200}, \frac{k}{2} \right) \ : \ 1 \leqslant j \leqslant 200,\ 0 \leqslant k \leqslant \lfloor 2T_{\max} \rfloor \right\}. \tag{4.9}$$

- $M_\Omega$ will denote the mesh used by LIONSIMBA to discretize $\Omega$ as a whole, *i.e.* :

$$M_\Omega = \bigcup_{i \in \{a,p,s,n,z\}} M_{\Omega_i} \tag{4.10}$$

- for all $i \in \{a,p,s,n,z\}$, $x_i^{beg}$ and $x_i^{end}$ will denote the discretization points in space used by LIONSIMBA at the left end of the section cell $i$ and at the right end of the section cell $i$, respectively. More formally, we have: for all $i \in \{a,p,s,n,z\}$:

$$x_i^{beg} := \frac{\hat{x}_i + (x_i - \hat{x}_i)}{400} = \frac{x_i}{400}, \tag{4.11}$$

and:

$$x_i^{end} := \frac{\hat{x}_i + 399(x_i - \hat{x}_i)}{400}. \tag{4.12}$$

- for all $i \in \{a,p,s,n,z\}$, $\hat{M}_{\Omega_i}^{bc}$ and $M_{\Omega_i}^{bc}$ will denote the part of the LIONSIMBA's mesh for section $i$ through time that is at the left end of the cell and at the right end of the section cell, respectively. More formally, we have: for all $i \in \{a,p,s,n,z\}$

$$\hat{M}_{\Omega_i}^{bc} := \left\{ \left( x_i^{beg}, \frac{k}{2} \right) \, : \, 1 \leqslant k \leqslant \lfloor 2T_{\max} \rfloor \right\}, \tag{4.13}$$

and:

$$M_{\Omega_i}^{bc} := \left\{ \left( x_i^{end}, \frac{k}{2} \right) \, : \, 1 \leqslant k \leqslant \lfloor 2T_{\max} \rfloor \right\}. \tag{4.14}$$

- for all $i \in \{a,p,s,n,z\}$, $M_{\Omega_i}^{ic}$ will denote the space discretization points of section $i$ in LIONSIMBA taken at initial time. More formally, we have: for all $i \in \{a,p,s,n,z\}$

$$M_{\Omega_i}^{ic} := \left\{ \left( \frac{(j - \frac{1}{2})(x_i - \hat{x}_i)}{200}, 0 \right) \, : \, 1 \leqslant j \leqslant 200 \right\}. \tag{4.15}$$

- for all $u \in \{T, C_e, C_s^\star, \phi_e, \phi_s\}$, $u_{\text{LS}}$ will denote the approximation of the variable $u$ by LIONSIMBA on the meshes of the sections corresponding to it. More precisely, we have that:
  - $T_{\text{LS}}$ is defined on $\bigcup_{i \in \{a,p,s,n,z\}} M_{\Omega_i}$
  - $C_{e\text{LS}}$ and $\phi_{e\text{LS}}$ are defined on $\bigcup_{i \in \{p,s,n\}} M_{\Omega_i}$
  - $C_{s\,\text{LS}}^\star$ and $\phi_{s\text{LS}}$ are defined on $\bigcup_{i \in \{p,n\}} M_{\Omega_i}$

- Following the last point, we introduce for all $i \in \{a,p,s,n,z\}$ the sequences $(v^{\text{LS},i}{}_j)_{1 \leqslant j \leqslant m_i}$ defined as:
  - for all $i \in \{a,z\}$, $(v_j^{\text{LS},i})_{1 \leqslant j \leqslant 1} = \left( T_{\text{LS}}\big|_{M_{\Omega_i}} \right)$.
  - for all $i \in \{p,n\}$, $(v_j^{\text{LS},i})_{1 \leqslant j \leqslant 5} = \left( T_{\text{LS}}\big|_{M_{\Omega_i}}, C_{e\text{LS}}\big|_{M_{\Omega_i}}, C_{s\,\text{LS}}^\star\big|_{M_{\Omega_i}}, \phi_{e\text{LS}}\big|_{M_{\Omega_i}}, \phi_{s\text{LS}}\big|_{M_{\Omega_i}} \right)$.
  - $(v_j^{\text{LS},s})_{1 \leqslant j \leqslant 3} = \left( T_{\text{LS}}\big|_{M_{\Omega_s}}, C_{e\text{LS}}\big|_{M_{\Omega_s}}, \phi_{e\text{LS}}\big|_{M_{\Omega_s}} \right)$.

Finally, without going into further detail, we will simply point out that LIONSIMBA slightly modifies the boundary conditions equations, by applying an harmonic mean to the parameters on either side of the interface between two sections of the cell, in order to make the transition smoother. For instance Equation (2.4b) is thus rewritten as: for all $t \in [0, T_{\max}]$

$$-\tilde{\lambda}_{a,p} \partial_x T(x,t)\Big|_{x = x_a^{end}} = -\tilde{\lambda}_{a,p} \partial_x T(x,t)\Big|_{x = x_p^{beg}},$$

where $\tilde{\lambda}_{a,p} := \frac{\lambda_a \lambda_p}{\beta\lambda_a + (1-\beta)\lambda_p}$ with $\beta$ accounting for the difference between the discretization widths in section $a$ and section $p$, *i.e.* $\beta = \frac{x_a - x_0}{x_p - x_0}$. See paragraph "Implementation of boundary and interface conditions" page A1197 of Torchio et al. (2016) for more details.

We now present in the next section, the metrics used to evaluate our results.

### 4.1.3. Evaluation metrics

Two types of metrics will be used to evaluate our results. The first are common metrics in the deep learning context, derived directly from the learning process and the second are more classical metrics in the context of approximating PDEs solutions:

**Loss based metrics:** Following the formalism of Section 3.2.1, they consist of the sequence of weights $\left(\ell(\tilde{\theta}_n)\right)_{1\leqslant n\leqslant\infty}$, bevor $\left(\tilde{\theta}_n\right)_{1\leqslant n\leqslant\infty}$ is the sequence of weights obtained by one of the gradient descent algorithms. More precisely, we will consider one such sequence $\ell^i$ for each output line $i$ of operators $D$ and $B$ defined in Section 4.1.1 (see Equations (4.5) and (4.8); line $i$ corresponds thus to an equation). The loss $\ell^i$ is computed by averaging the square of the line output over the sampled points. For instance, back to our example of Equation (2.3a), the associated loss $\ell^{(2.3a)}$ is given by:

$$\ell^{(2.3a)}(\theta) := \frac{1}{|S_{\Omega_a}|} \sum_{(x,t)\in S_{\Omega_a}} \left| \rho_a C_{p,a} \partial_t \Psi(\theta)(x,t) - \lambda_a \partial_{x^2} \Psi(\theta)(x,t) - \frac{I_{app}^2(t)}{\sigma_{\text{eff},a}} \right|^2 .$$

where $S_{\Omega_a}$ is a set of points sampled on $\Omega_a$, of cardinal $|S_{\Omega_a}|$.

Those metrics will usually be illustrated by the graph of $\left(\ell(\tilde{\theta}_n)\right)_{1\leqslant n\leqslant\infty}$ with respect to $n$.

**Absolute and relative errors:** Those metrics will be computed with the help of the data obtained by LIONSIMBA (see Section 4.1.2). More precisely, given a variable $u \in \{T, C_e, C_s^\star, \phi_e, \phi_s\}$ defined on a domain $\Omega_i$, $i \in \{a, p, s, n, z\}$, a subset $A \subset M_{\Omega_i}$ of $M_{\Omega_i}$, we will assess the PINNs results by computing the Mean Absolute Error (MAE):

$$\text{MAE}\left(u\big|_A\right) := \frac{1}{|A|} \sum_{(x,t)\in A} |u_{\text{PINN}}(x,t) - u_{\text{LS}}(x,t)| , \tag{4.16}$$

and the Mean Relative Error (MRE):

$$\text{MRE}\left(u\big|_A\right) := \frac{1}{|A|} \sum_{(x,t)\in A} \frac{|u_{\text{PINN}}(x,t) - u_{\text{LS}}(x,t)|}{|u_{\text{LS}}(x,t)|}, \tag{4.17}$$

where $u_{\text{PINN}}$ is an approximation of $u$ obtained by a neural network trained with a PINNs loss and $u_{\text{LS}}$ is an approximation of $u$ given by LIONSIMBA. Note that $|A|$ refers again to the cardinal of the set $A$. Along the same lines, we could consider the Mean Squared Error (MSE):

$$\text{MSE}\left(u\big|_A\right) := \frac{1}{|A|} \sum_{(x,t)\in A} |u_{\text{PINN}}(x,t) - u_{\text{LS}}(x,t)|^2 , \tag{4.18}$$

and the Mean Squared Relative Error (MSRE):

$$\mathrm{MSRE}\left(u\big|_A\right) := \frac{1}{|A|}\sum_{(x,t)\in A}\frac{|u_{\mathrm{PINN}}(x,t)-u_{\mathrm{LS}}(x,t)|^2}{|u_{\mathrm{LS}}(x,t)|^2}, \tag{4.19}$$

with the same notations as for MAE and MRE.

Rather than considering these numbers, we will also often plot as a heat map, the difference function:

$$\Delta_u := (x,t)\in M_{\Omega_i} \mapsto u_{\mathrm{PINN}}(x,t)-u_{\mathrm{LS}}(x,t), \tag{4.20}$$

and the relative difference function:

$$\Delta_u^r := (x,t)\in M_{\Omega_i} \mapsto \frac{u_{\mathrm{PINN}}(x,t)-u_{\mathrm{LS}}(x,t)}{u_{\mathrm{LS}}(x,t)}, \tag{4.21}$$

with the same notations as above.

In the following section, we illustrate the framework presented in this section by applying a purely data-driven approach, taking the opportunity to show its limitations, which justifies our work.

## 4.2. Data driven approach

The first experiments we conducted involved an approach based purely on data from LIONSIMBA (*cf.* Section 4.1.2). Their aim is to show that neural networks can correctly approximate the solutions of the simplified P2D model (*cf.* Section 2), and to infer certain hyper-parameters and modifications of the neural network and loss. They also serve to illustrate the metrics we presented in Section 4.1.3.

### 4.2.1. Loss

To define the loss, we will consider for all $i\in\{a,p,s,n,z\}$ and $m\in[0,100]$, sub-samples $M_{\Omega_i}^{m\%}$ of $M_{\Omega_i}$ containing $m$ percent of the points contained in $M_{\Omega_i}$, with $M_{\Omega_i}$ the notation of Equation (4.9) introduced in Section 4.1.2. Given these sets, we will train a neural network $u_{\mathrm{PINN}}$ to learn the values of $u_{\mathrm{LS}}\big|_{M_{\Omega_i}^{m\%}}$ for all $u\in\{T,C_e,C_s^\star,\phi_s,\phi_e\}$ with the notations introduced in Section 4.1.3.

More formally, for all $i\in\{a,p,s,n,z\}$ we replace the operator $D_i$ of Section 4.1.1 by:

$$D_i^{\mathrm{dd}} : \begin{cases} \mathcal{C}^2\left(\Omega_i\to\mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\Omega_i\to\mathbb{R}\right)^{m_i} \\ (u_j)_{1\leqslant j\leqslant m_i} & \longmapsto & \left(x\in\Omega_i\mapsto\sum_{y\in M_{\Omega_i}}\delta_y(x)\left(u_j(x)-\upsilon_j^{\mathrm{LS},i}(y)\right)\right)_{1\leqslant j\leqslant m_i} \end{cases}, \tag{4.22}$$

with the notations of Section 4.1.2 and for all $y\in M_{\Omega_i}$ $\delta_y$ being the Dirac delta distribution on $\Omega_i$ at point $y$:

$$\delta_y : x\in\Omega_i \mapsto \begin{cases} 1 & \text{if } x=y \\ 0 & \text{if } x\neq y \end{cases},$$

resulting in a modifed operator $D^{\mathrm{dd}}$ built from $D_i^{\mathrm{dd}}$ as $D$ is from $D_i$. We then set the $B$ operator to 0 and obtain the data-driven loss on $m$ percent samples:

$$\mathcal{L}_{m\%}(f) := \frac{1}{|M_\Omega^{m\%}|} \sum_{(x,t)\in M_\Omega^{m\%}} \left\| D^{\mathrm{dd}}(f)(x,t) \right\|_2^2, \tag{4.23}$$

with $M_\Omega^{m\%} := \bigcup_{i\in\{a,p,s,n,z\}} M_{\Omega_i}^{m\%}$, or equivalently:

$$\mathcal{L}_{m\%}(f) := \sum_{i\in\{a,p,s,n,z\}} \frac{1}{|M_{\Omega_i}^{m\%}|} \sum_{(x,t)\in M_{\Omega_i}^{m\%}} \left\| D_i^{\mathrm{dd}}(f)(x,t) \right\|_2^2.$$

### 4.2.2. Neural network setting

To define the approximating neural network, we take for all $i \in \{a,p,s,n,z\}$ one architecture $\Psi_i$ approximating simultaneously the $m_i$ variables defined in section $i$, with widths $(2, 128, 128, 128, m_i)$ and scalar activations tanh (*cf.* Table 16), except on the last layer, where we take a linear activation. Thus, we have: for all $i \in \{a,p,s,n,z\}$

$$\mathrm{Im}\,\Psi_i \subset \mathcal{C}^\infty(\Omega_i \to \mathbb{R}^{m_i}) \subset \mathcal{C}^\infty(\Omega_i \to \mathbb{R})^{m_i},$$

and we set for instance for $i = s$, for all $\theta \in \mathbb{R}^P$, $\Psi_i(\theta) = (T_{\mathrm{PINN}}, C_{e\mathrm{PINN}}, \phi_{e\mathrm{PINN}})$, with $P = \sum_{j=1}^4 (l_j \times (l_{j-1}+1))$, where $(l_j)_{0\leqslant j\leqslant 4} = (2, 128, 128, 128, m_i)$.

More precisely, for all $i \in \{a,p,s,n,z\}$, we right-compose $\Psi_i$ by an input scaling function $s_i : \Omega_i \to \left(-\frac{\sqrt{12}}{2}, \frac{\sqrt{12}}{2}\right) \times \left[-\frac{\sqrt{12}}{2}, \frac{\sqrt{12}}{2}\right]$ that transforms the coordinates of $\Omega_i$ so that the image of an uniform sampling on $\Omega_i$ by $s_i$ has mean zero and variance 1. This is motivated by the fact that neural networks are known to perform poorly if their inputs are not properly scaled, see for instance Sola and Sevilla (1997). More precisely, we define: for all $i \in \{a,p,s,n,z\}$

$$s_i : \left\{ \begin{array}{ccc} \Omega_i & \longrightarrow & \left(-\frac{\sqrt{12}}{2}, \frac{\sqrt{12}}{2}\right) \times \left[-\frac{\sqrt{12}}{2}, \frac{\sqrt{12}}{2}\right] \\ (x,t) & \longmapsto & \left( \frac{x - \frac{x_i + \hat{x}_i}{2}}{\frac{x_i - \hat{x}_i}{\sqrt{12}}}, \frac{t - \frac{T_{\max}}{2}}{\frac{T_{\max}}{\sqrt{12}}} \right) \end{array} \right., \tag{4.24}$$

and modify $\Psi_i$ into $\Psi_i'$, so that for all $\theta \in \mathbb{R}^P$, $\Psi_i'(\theta)$ is defined in $\left(-\frac{\sqrt{12}}{2}, \frac{\sqrt{12}}{2}\right) \times \left[-\frac{\sqrt{12}}{2}, \frac{\sqrt{12}}{2}\right]$ and no longer in $\Omega_i$, and consider $\Psi_i'(\theta) \circ s_i$ instead of $\Psi_i$. Similarly, as the orders of magnitude of the different variables approximated by $\Psi_i$ is very different, we also left-compose, for all $i \in \{a,p,s,n,z\}$, $\Psi_i'$ by an output scaling function $o_i$, so that the outputs of the neural network are of zero mean and variance equal to 1 when the network correctly learns the solutions of the P2D model. To do so, we compute the empirical mean and variance of each variable on each section thanks to the LIONSIMBA approximation. More formally, we define: for all $i \in \{a,p,s,n,z\}$

$$o_i : \left\{ \begin{array}{ccc} \mathbb{R}^{m_i} & \longrightarrow & \mathbb{R}^{m_i} \\ (y_j)_{1\leqslant j\leqslant m_i} & \longmapsto & \left( y_j \sqrt{w_j^i} + \frac{1}{|M_{\Omega_i}|} \sum_{z\in M_{\Omega_i}} v_j^{\mathrm{LS},i}(z) \right)_{1\leqslant j\leqslant m_i} \end{array} \right., \tag{4.25}$$

where: for all $i \in \{a,p,s,n,z\}$ and for all $1 \leqslant j \leqslant m_i$

$$w_j^i = \frac{1}{|M_{\Omega_i}| - 1} \sum_{z\in M_{\Omega_i}} \left( v_j^{\mathrm{LS},i}(z) - \frac{1}{|M_{\Omega_i}|} \sum_{\omega\in M_{\Omega_i}} v_j^{\mathrm{LS},i}(\omega) \right), \tag{4.26}$$

and consider for all $\theta \in \mathbb{R}^P$, $o_i \circ \Psi_i'(\theta) \circ s_i$ as the approximating function. To set the mind at rest, we define on each cell section $i \in \{a, p, s, n, z\}$, the neural network to be trained on section $i$ as the function:

$$\tilde{\Psi}_i : \left\{ \begin{array}{ccc} \mathbb{R}^P & \longrightarrow & \mathcal{C}^\infty(\Omega_i \to \mathbb{R}^{m_i}) \\ \theta & \longmapsto & o_i \circ \Psi_i'(\theta) \circ s_i \end{array} \right. , \tag{4.27}$$

and the global neural as the function:

$$\tilde{\Psi} : \left\{ \begin{array}{ccc} \left(\mathbb{R}^P\right)^5 & \longrightarrow & \prod_{i\in\{a,p,s,n,z\}} \mathcal{C}^\infty(\Omega_i \to \mathbb{R}^{m_i}) \\ (\theta_i)_{i\in\{a,p,s,n,z\}} & \longmapsto & \left(\tilde{\Psi}_i(\theta_i)\right)_{i\in\{a,p,s,n,z\}} \end{array} \right. , \tag{4.28}$$

### 4.2.3. Naive data driven approach

Given the setup defined above, we will train the neural network defined in Equation (4.28) according to the loss defined in Equation (4.23) with $m = 75$, such that each set $M_{\Omega_i}^{75\%}$ with $i \in \{a, p, s, n, z\}$, contains 75% of the points in $M_{\Omega_i}$, and a constant applied current density $I_{\text{app}} = 40\,\text{A}{\cdot}\text{m}^{-2}$ (*cf.* Table 7). The training will be performed with 15000 steps of the Adam algorithm, initialized with $(\delta, \beta, \gamma) = (10^{-3}, 0.9, 0.999)$ (*cf.* Definition 3.10), followed by 1000 steps of L-BFGS algorithm. We also apply a scheduling heuristic that decays $\delta$ to $10^{-4}$ from step 500 and to $10^{-5}$ from step 4000 during the Adam training. We will also monitor the "test" loss:

$$\mathcal{L}_{25\%}(f) := \sum_{i\in\{a,p,s,n,z\}} \frac{1}{|M_{\Omega_i} \backslash M_{\Omega_i}^{m\%}|} \sum_{(x,t)\in M_{\Omega_i}\backslash M_{\Omega_i}^{m\%}} \left\| D^{\text{dd}}(f)(x,t) \right\|_2^2 , \tag{4.29}$$

which evaluates the generalization error (*cf.* Section 3.2.4) of the neural network. Results are plotted in Figure 5.

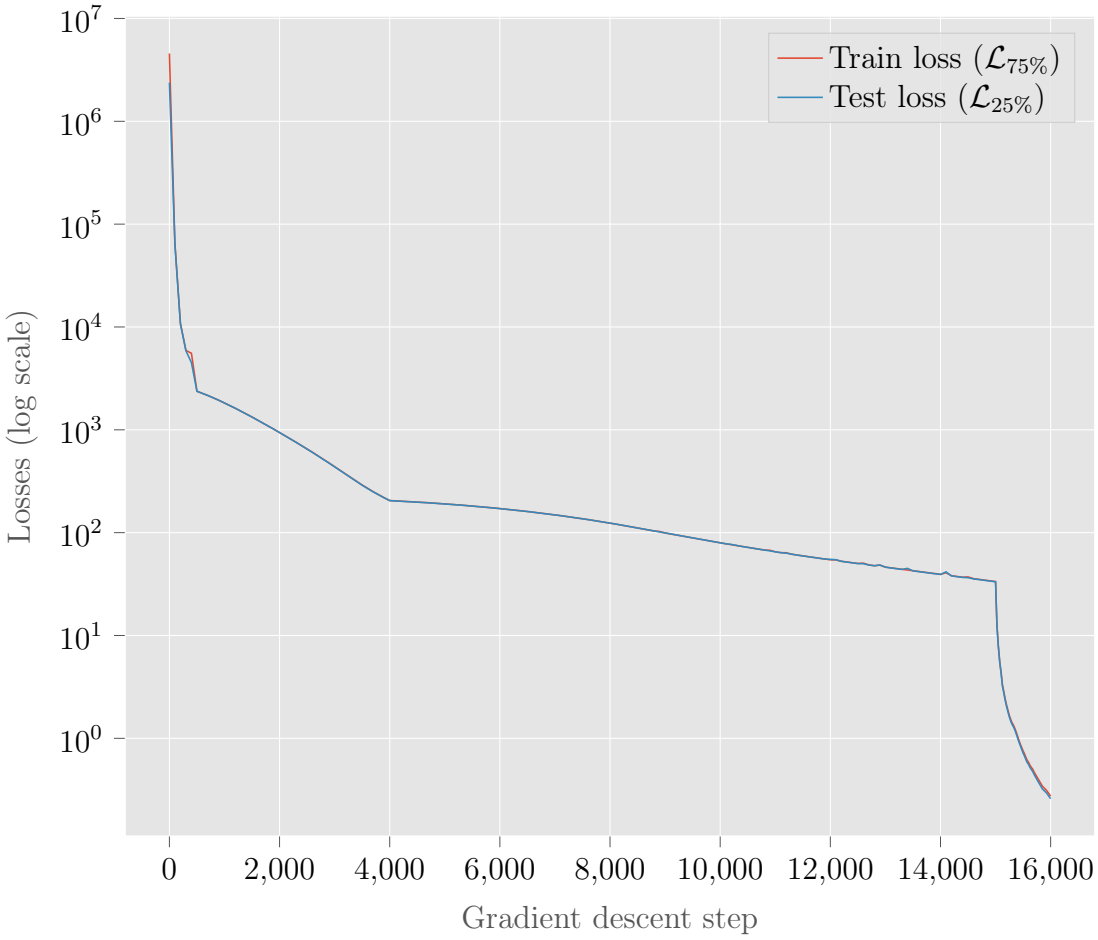



Figure 5: Train loss ($\mathcal{L}_{75\%}$) and Test loss ($\mathcal{L}_{25\%}$) for the data-driven approach with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the two losses match perfectly and lose 8 orders of magnitude during training, indicating that the neural network learns and generalizes very well.
On the other hand, we may wonder about the values of these errors. This is because the variables $C_e$ and $C_s^\star$ have an orders of magnitude of $10^4$, which is consistent with an initial MSE of $10^8$. On the other hand, one may wonder if the other variables have been properly learned.
Finally, we note the effect of scheduling, with clear inflections of the loss slope at the 500th and at the 4000th step, and of the change from Adam to L-BFGS at the 15000th step, with an even clearer change in the loss slope.

As observed in Figure 5, we may wonder whether all the variables have been learned equally well. To check this, we will monitor the "test" MSEs and "train" MSEs for each of the variables, *i.e.* for a given variable $u \in \{T, C_e, C_s^\star, \phi_s, \phi_e\}$, the average over $\text{SD}(u)$ of the Mean Squared Errors of the variable $u$, computed with $M_{\Omega_i}^{m\%}$ for train and $M_{\Omega_i} \backslash M_{\Omega_i}^{m\%}$ for test, for each $i \in \{a, p, s, n, z\}$ in which $u$ is defined. More formally: for all $u \in \{T, C_e, C_s^\star, \phi_s, \phi_e\}$

$$\text{Train MSE}(u) = \frac{1}{|SD(u)|} \sum_{i \in \text{SD}(u)} \frac{1}{|M_{\Omega_i}^{75\%}|} \sum_{(x,t) \in M_{\Omega_i}^{75\%}} |u_{\text{LS}}(x,t) - u_{\text{PINN}}(x,t)|^2,$$

and:

$$\text{Test MSE}(u) = \frac{1}{|SD(u)|}\sum_{i\in \text{SD}(u)} \frac{1}{|M_{\Omega_i}\backslash M_{\Omega_i}^{75\%}|}\sum_{(x,t)\in M_{\Omega_i}\backslash M_{\Omega_i}^{75\%}} |u_{\text{LS}}(x,t) - u_{\text{PINN}}(x,t)|^2.$$

These MSEs are plotted in Figure 6 below. As one might have feared, we see that the difference in magnitude between the variables affects the training of the variables $T$, $\phi_e$ and $\phi_s$, which are those with the smallest orders of magnitude. Note, however, that this is only true for training with L-BFGS and not with the Adam. This can be explained by Adam's adaptive mechanism, which renormalizes the gradient step on each weight of the neural network by the sliding average of the norms of successive gradients on that weight (*cf.* Definition 3.10 in Section 3.2.1).

To correct this, we are going to adapt the loss by assigning a different weight to each variable in the next section.

### 4.2.4. Data driven approach with weight correction

In this section we show how to compensate for the imbalance between the orders of magnitude of the different variables, which affects the training in Section 4.2.3, by assigning a different weight to each variable. Since the setting of our neural network $\tilde{\Psi}$ of Equation (4.28) rescale the output by the the empirical variances calculated from LIONSIMBA, defined in Equation (4.26), the most logical option seems to rescale the loss according to those weights. More formally, we will rewrite the loss $\mathcal{L}_{75\%}$ of Equation (4.23) as:

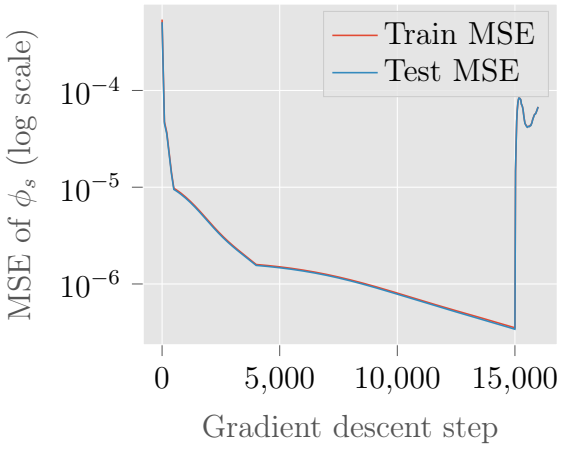
$$\mathcal{L}_{m\%}^{\text{weighted}}(f) := \sum_{i\in\{a,p,s,n,z\}} \frac{1}{|M_{\Omega_i}^{m\%}|}\sum_{(x,t)\in M_{\Omega_i}^{m\%}}\sum_{j=1}^{m_i} \left(w_j^i\right)^{-1}\left(D(f)(x,t)\right)_j^2, \tag{4.30}$$

with $w_j^i$ from Equation (4.26).

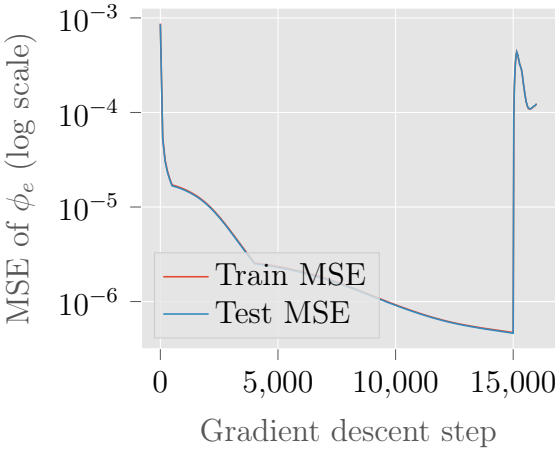
To evaluate more accurately the results, we will use a 4-fold cross-validation method (*cf.* Goodfellow et al. 2016, Algorithm 5.1, page 123). Namely for all $i \in \{a,p,s,n,z\}$, we randomly split each $M_{\Omega_i}$ into a 4-partition $(A_j^i)_{1\leqslant j\leqslant 4}$ where all $A_j^i$ are of the same cardinal, so that they each represent 25% of $\Omega_i$. Given $J \in \{1,2,3,4\}$, we then set $M_{\Omega_i}^{75\%} = \bigcup_{j\in\{1,2,3,4\}\backslash\{J\}} A_j^i$ in loss of Equation (4.30) yielding a loss $\mathcal{L}_J^{\text{train}}$, and train the neural network $\tilde{\Psi}$ of Equation (4.28) according to this loss, while monitoring the associated test loss:

$$\mathcal{L}_J^{\text{test}}(f) := \sum_{i\in\{a,p,s,n,z\}} \frac{1}{|A_J^i|}\sum_{(x,t)\in A_J^i}\sum_{j=1}^{m_i} \left(w_j^i\right)^{-1}\left(D(f)(x,t)\right)_j^2,$$

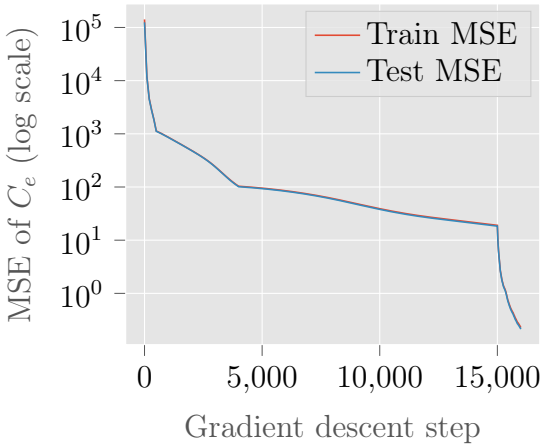
to assess the generalization error (*cf.* Section 3.2.4) of the neural network. Results are plotted in Figure 7 below.



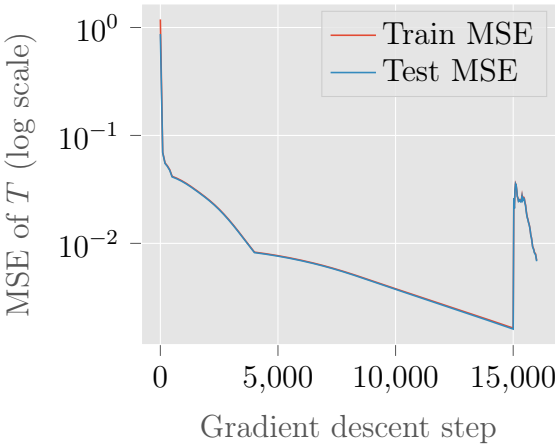

(a) MSE of solid potential ($\phi_s$)



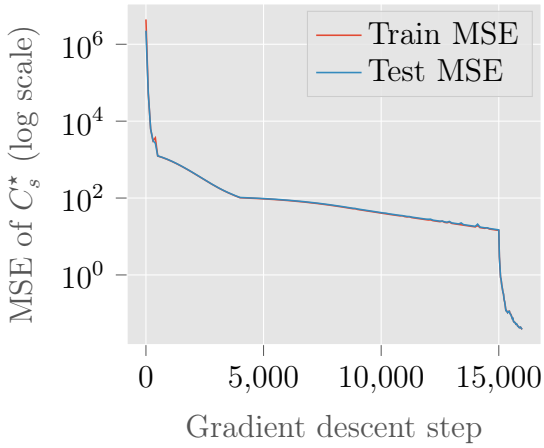

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

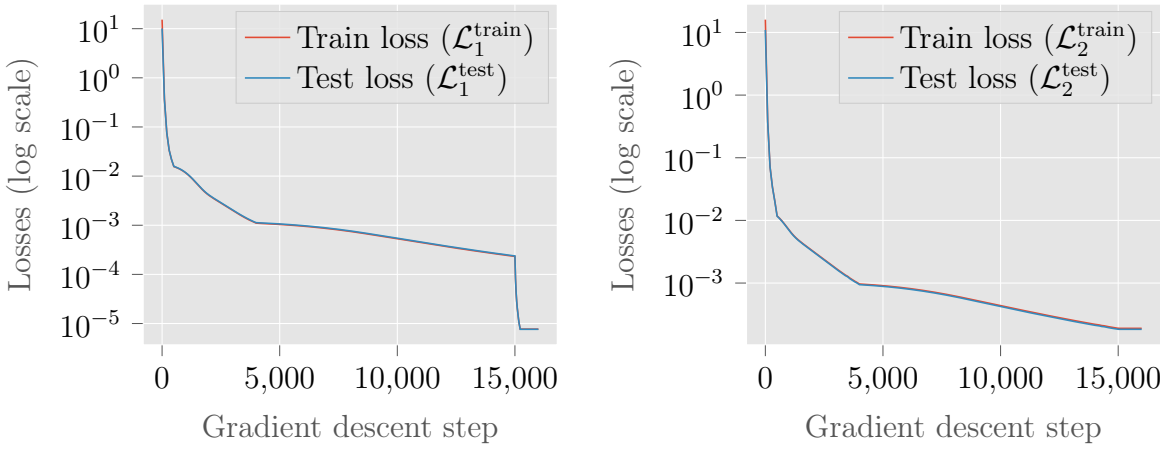



(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

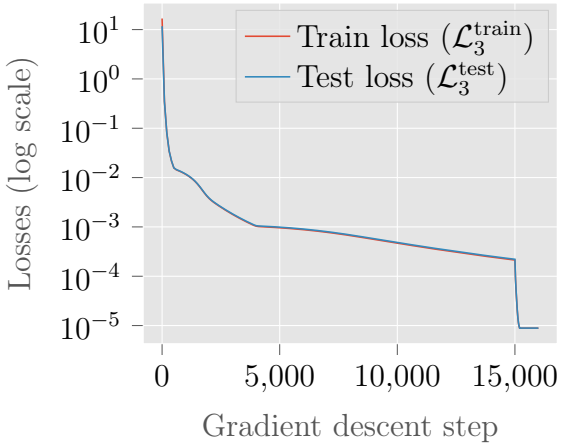
Figure 6: Train and Test MSEs of each of the variables of the model for the data-driven approach, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We can see that the difference in scale between the variables is not a problem for training with ADAM, but it is for training with L-BFGS.



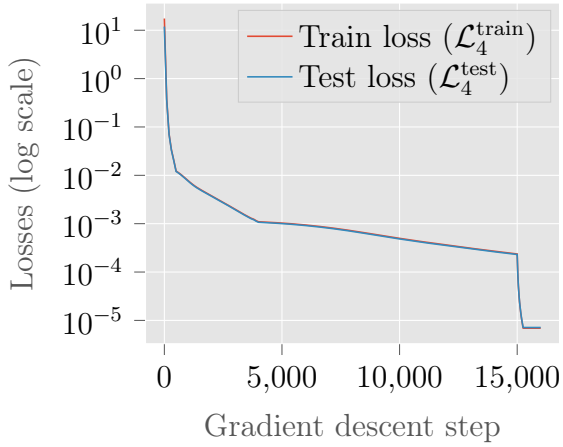

(a) Total loss on 4-fold 1




(b) Total loss on 4-fold 2



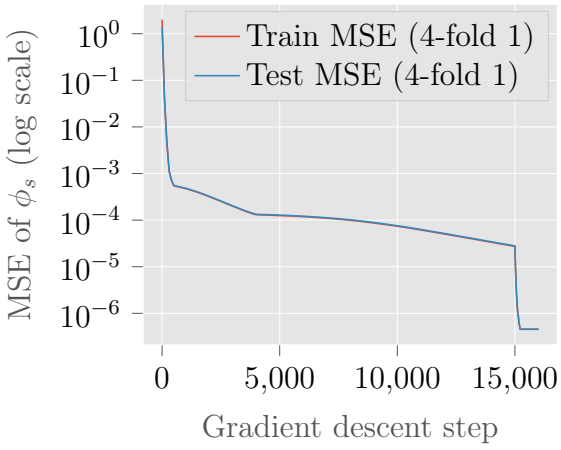

(c) Total loss on 4-fold 3



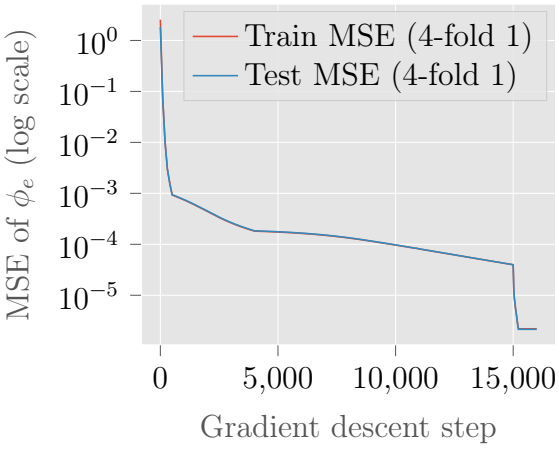

(d) Total loss on 4-fold 4

Figure 7: 4-fold weighted train losses ($\left(\mathcal{L}_J^{\text{train}}\right)_{1\leqslant J\leqslant 4}$) and weighted test losses ($(\mathcal{L}_J^{\text{test}})_{1\leqslant J\leqslant 4}$) for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that train and test losses match perfectly in all 4-folds and lose 6 orders of magnitude during training, except for fold 2, where L-BFGS did not converge. This indicates that the neural network learns and generalizes very well, even if optimization failures remain possible.
We note the effect of scheduling, with clear inflections of the loss slope at the 500th and at the 4000th step, and of the change from Adam to L-BFGS at the 15000th step, with an even clearer change in the loss slope for 4-folds 1, 3 and 4.

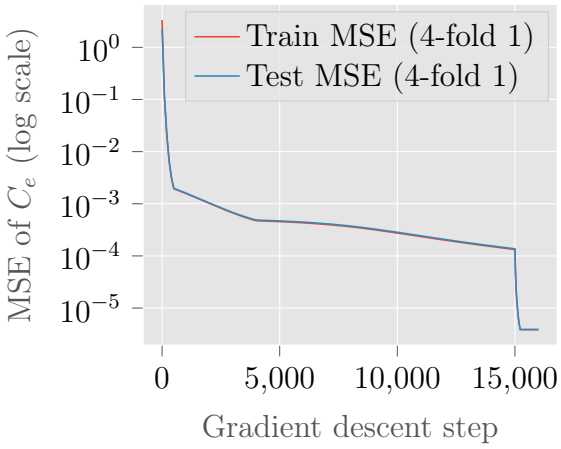
As in the previous experiment, we will plot the MSEs variable by variable to check that convergence does indeed take place for each of the predicted variables. We plot the graphs for the 0-fold below and plot them for all the 4-folds in Appendix F.2 of Appendix F for completeness.



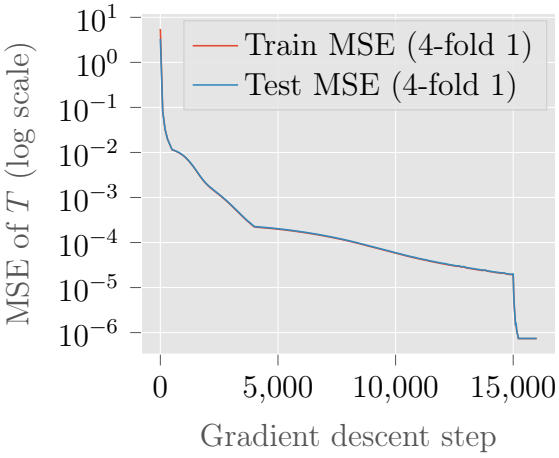

(a) MSE of solid potential ($\phi_s$)



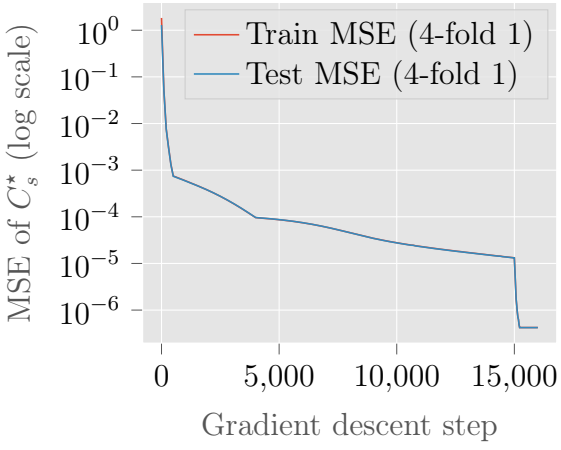

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)




(d) MSE of temperature ($T$)



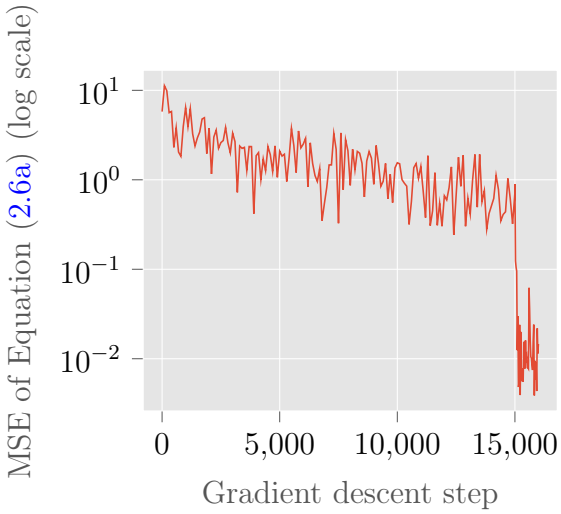

(e) MSE of lithium ions surface concentration in the solid-phase ($C_s^\star$)

Figure 8: Train and Test MSEs of each of the variables of the model for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) on 4-fold 1, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We can see that the difference in scale between the variables is not a problem for training with ADAM, but it is for training with L-BFGS.

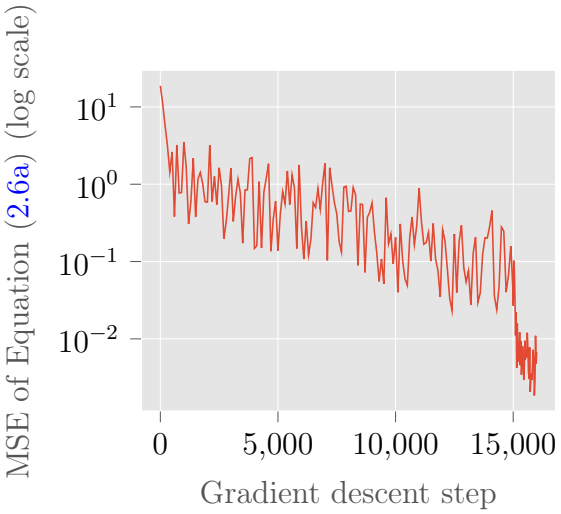
In contrast to the previous case, we can see that all variables have been equally learned, which validates our correction. We can now ask what the generalization of the neural network is to higher orders, *i.e.* whether, beyond its good interpolation capabilities, attested by its performance on test data from LIONSIMBA, the neural network actually learns the PDEs of the P2D model. To verify this, we will plot the following MSEs that measure the extent to which the $D_i$ operators derived from the P2D equations depicted in Equation (4.3) of Section 4.1.1, deviate from 0: for all $i \in \{a, p, s, n, z\}$, for all $1 \leqslant j \leqslant m_i$

$$\text{PDE MSE}(i,j) := \frac{1}{|S_i|} \sum_{(x,t)\in S_i} \left| \left(D_i[u_{\text{PINN}}](x,t)\right)_j \right|^2, \tag{4.31}$$

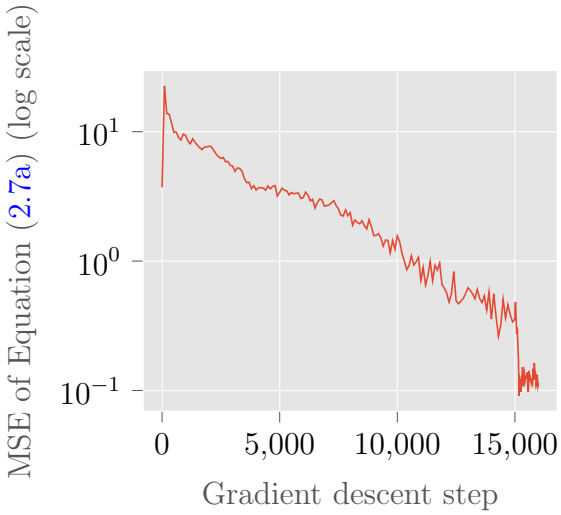
where $S_i$ is an LHS sampling of $\Omega_i$ of cardinal $|S_i|$. The orders of magnitude of these MSEs is not important here, what counts is the learning dynamics, that we will plot for 4-fold 1 below with for all $i \in \{a, p, s, n, z\}$, $|S_i| = 102$.



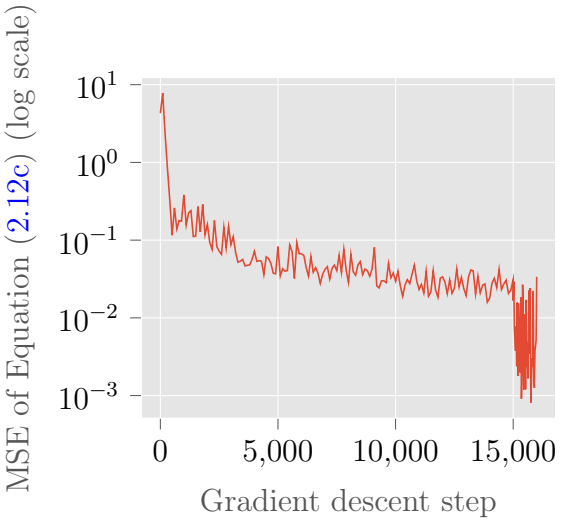

(a) MSE associated to lithium ions accumulation in the electrolyte in anode PDE (Equation (2.6a))



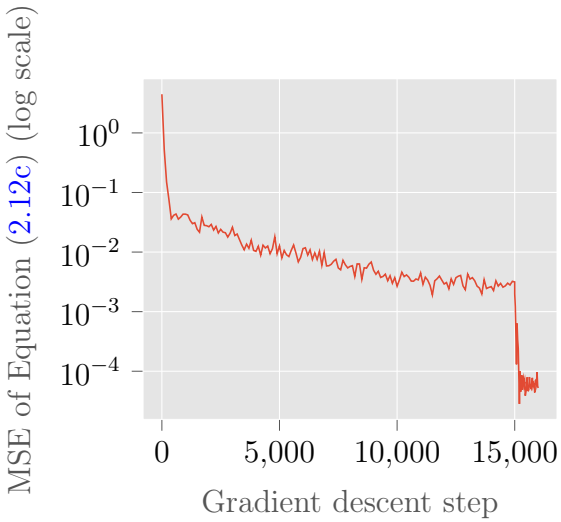

(b) MSE associated to lithium ions accumulation in the electrolyte in cathode PDE (Equation (2.6a))



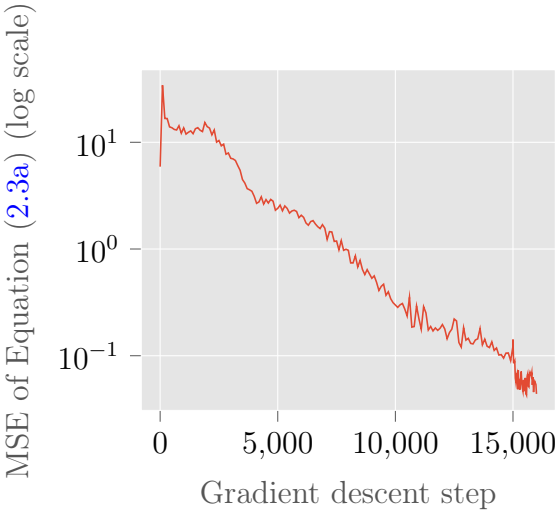

(c) MSE associated to lithium ions accumulation in the electrolyte in separator PDE (Equation (2.7a))



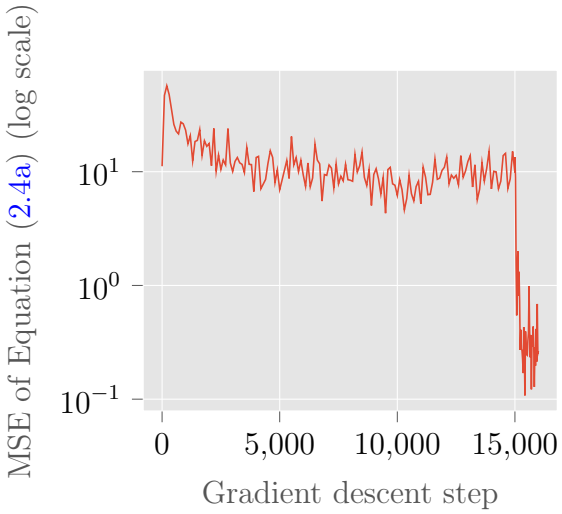

(d) MSE associated to lithium ions intercalation in solid particles in anode PDE (Equation (2.12c))



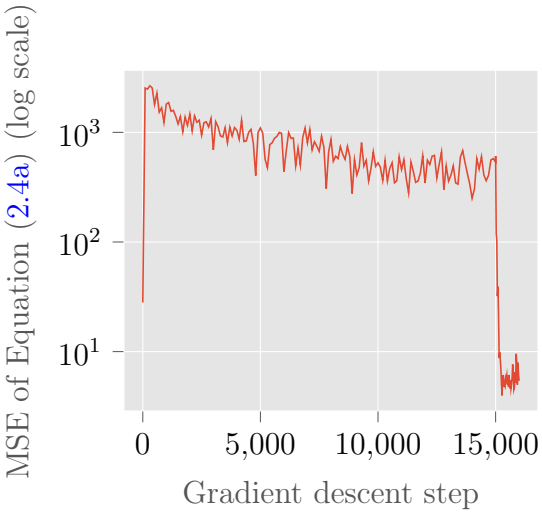

(e) MSE associated to lithium ions intercalation in solid particles in cathode PDE (Equation (2.12c))



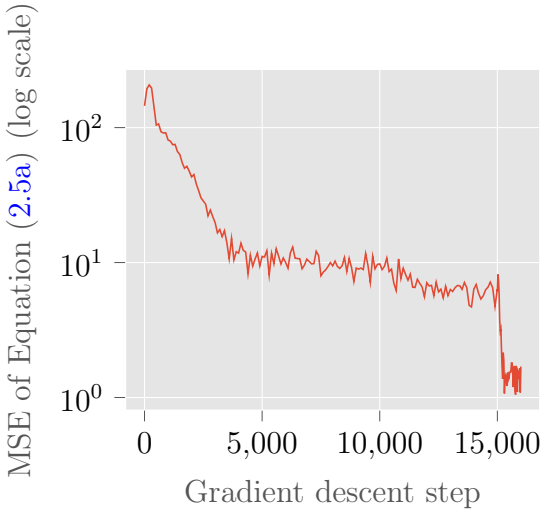

(f) MSE associated to temperature in positive current collector PDE (Equation (2.3a))



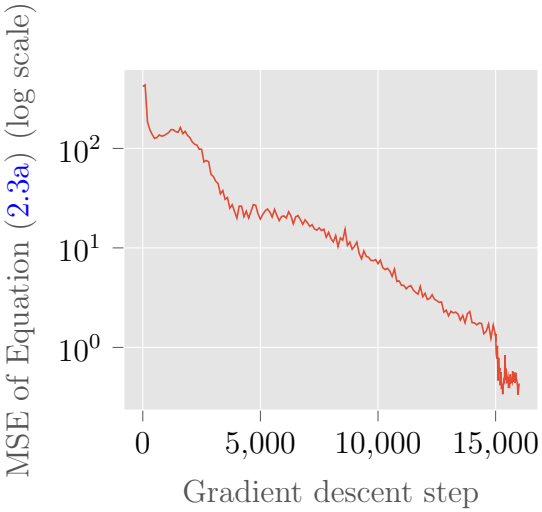

(g) MSE associated to temperature in anode PDE (Equation (2.4a))



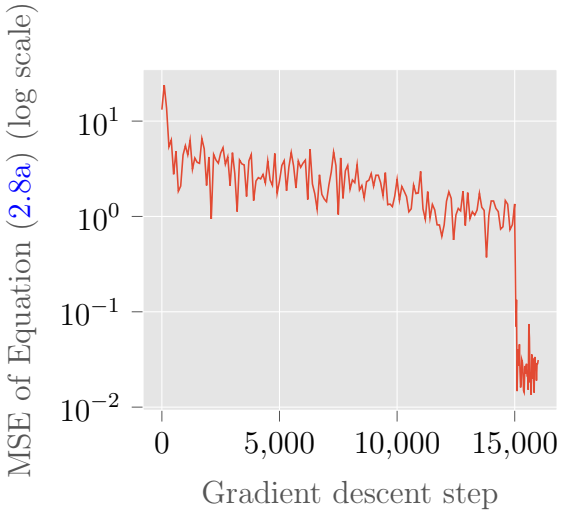

(h) MSE associated to temperature in cathode PDE (Equation (2.4a))



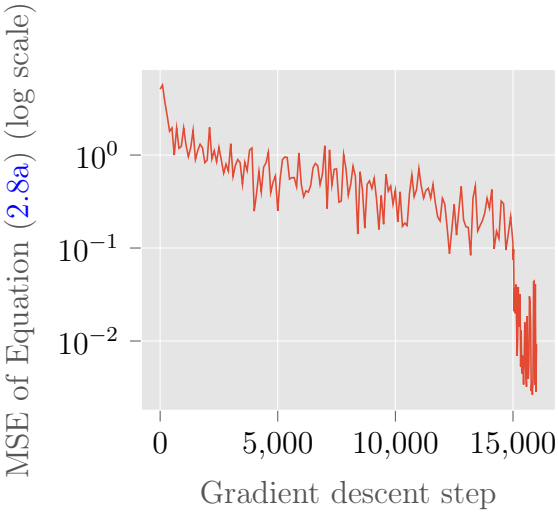

(i) MSE associated to temperature in separator PDE (Equation (2.5a))



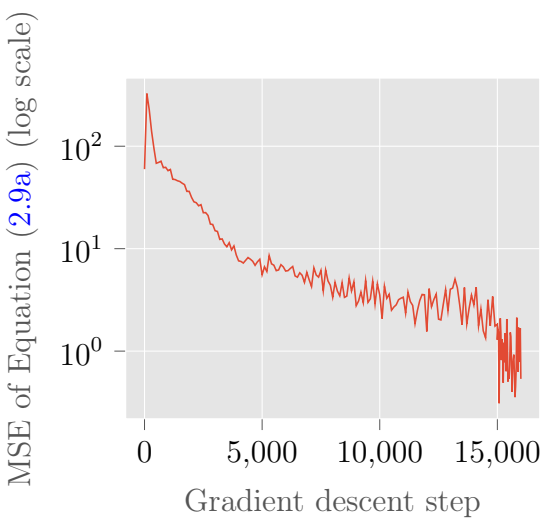

(j) MSE associated to temperature in negative current collector PDE (Equation (2.3a))



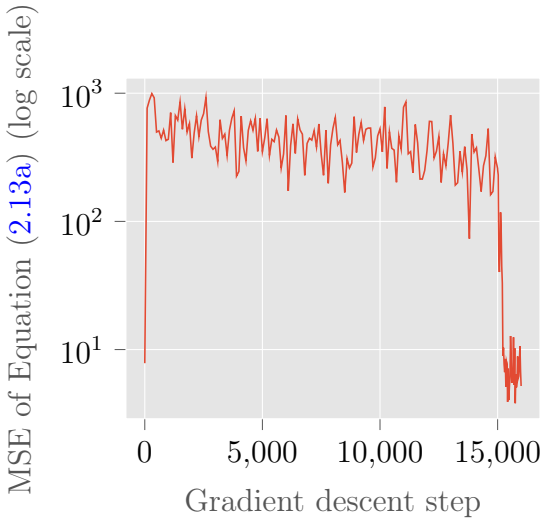

(k) MSE associated to lithium ions transportation in anode PDE (Equation (2.8a))



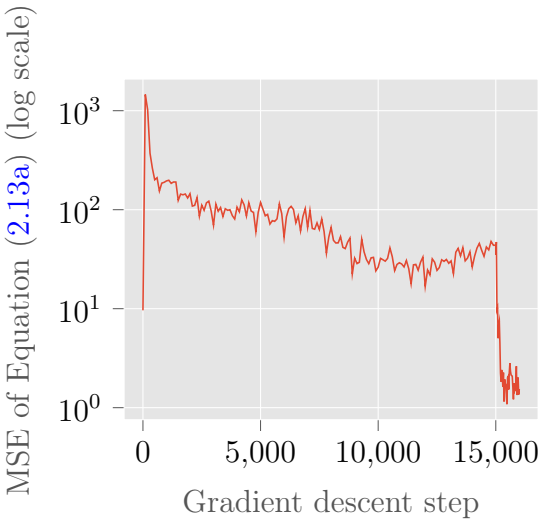

(l) MSE associated to lithium ions transportation in cathode PDE (Equation (2.8a))




(m) MSE associated to lithium ions transportation in separator PDE (Equation (2.9a))




(n) MSE associated to electrons motion in anode PDE (Equation (2.13a))




(o) MSE associated to electrons motion in cathode PDE (Equation (2.13a))

Figure 9: MSEs for each of the PDEs of the model for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) on 4-fold 1, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We can see that performance on each PDE is improved by between two and four orders of magnitude, the improvement being due almost exclusively to L-BFGS in the case of the cathode and anode, which are the most complex to model. This shows both that the model generalizes fairly well to higher orders and highlights the crucial role of L-BFGS.

We observe in Figure 9 that the generalization applies also to higher-orders. However, we wonder whether this remains true if we take much less input data, for example of the order of 1% instead of 75%. This is the subject of the next section.

### 4.2.5. Data driven approach with boundary data

In this section, we will show that learning from the data is not viable anymore if the data are scarce. To this end, we will learn only with the data coming from the two ends domain $\hat{M}_{\Omega_i}^{bc}$ and $M_{\Omega_i}^{bc}$ of each cell section $i \in \{a, p, s, n, z\}$ in LIONSIMBA, as well as from initial time domain $M_{\Omega_i}^{ic}$. More formally, we will set for all $i \in \{a, p, s, n, z\}$ the operators:

$$\hat{B}_i^{bc,\mathrm{dd}} : \begin{cases} \mathcal{C}^2\left(\Omega_i \to \mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\Omega_i \to \mathbb{R}\right)^{m_i} \\ \left(u_j\right)_{1 \leqslant j \leqslant m_i} & \longmapsto & \left(x \in \Omega_i \mapsto \sum_{y \in \hat{M}_{\Omega_i}^{bc}} \delta_y(x) \left(u_j(x) - \upsilon_j^{\mathrm{LS},i}(y)\right)\right)_{1 \leqslant j \leqslant m_i} \end{cases}, \tag{4.32}$$

and:

$$B_i^{bc,\mathrm{dd}} : \begin{cases} \mathcal{C}^2\left(\Omega_i \to \mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\Omega_i \to \mathbb{R}\right)^{m_i} \\ \left(u_j\right)_{1 \leqslant j \leqslant m_i} & \longmapsto & \left(x \in \Omega_i \mapsto \sum_{y \in M_{\Omega_i}^{bc}} \delta_y(x) \left(u_j(x) - \upsilon_j^{\mathrm{LS},i}(y)\right)\right)_{1 \leqslant j \leqslant m_i} \end{cases}, \tag{4.33}$$

as well as the operator:

$$B_i^{ic,\mathrm{dd}} : \begin{cases} \mathcal{C}^2\left(\partial\Omega_i \to \mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\partial\Omega_i \to \mathbb{R}\right)^{m_i} \\ \left(u_j\right)_{1 \leqslant j \leqslant m_i} & \longmapsto & \left(x \in \Omega_i \mapsto \sum_{y \in M_{\Omega_i}^{ic}} \delta_y(x) \left(u_j(x) - \upsilon_j^{\mathrm{LS},i}(y)\right)\right)_{1 \leqslant j \leqslant m_i} \end{cases}, \tag{4.34}$$

where we used the notation of Section 4.1.2 for all $y \in \Omega_i$, and $\delta_y$ is the Dirac delta distribution.

*Remark* 4.3. Strictly speaking, $\hat{B}_i^{bc,\mathrm{dd}}$, $B_i^{bc,\mathrm{dd}}$ are not defined on the boundary of $\Omega$. Nevertheless, as it is the more accurate approximation that we can get of the boundary domain with LIONSIMBA, we will neglect this aspect.

We then define a loss built upon $\hat{B}_i^{bc,\mathrm{dd}}$, $B_i^{bc,\mathrm{dd}}$ and $B_i^{ic,\mathrm{dd}}$ in a similar way as for the weight corrected loss of Equation (4.30), namely:

$$\begin{aligned} \mathcal{L}_{\mathrm{boundary}}^{\mathrm{train}}(f) := \sum_{i \in \{a,p,s,n,z\}} \Bigg( & \frac{1}{|\hat{M}_{\Omega_i}^{bc}|} \sum_{(x,t) \in \hat{M}_{\Omega_i}^{bc}} \sum_{j=1}^{m_i} \left(w_j^i\right)^{-1} \left(\hat{B}_i^{bc,\mathrm{dd}}(f)(x,t)\right)_j^2 \\ & + \frac{1}{|M_{\Omega_i}^{bc}|} \sum_{(x,t) \in M_{\Omega_i}^{bc}} \sum_{j=1}^{m_i} \left(w_j^i\right)^{-1} \left(B_i^{bc,\mathrm{dd}}(f)(x,t)\right)_j^2 \\ & + \frac{1}{|M_{\Omega_i}^{ic}|} \sum_{(x,t) \in M_{\Omega_i}^{ic}} \sum_{j=1}^{m_i} \left(w_j^i\right)^{-1} \left(B_i^{ic,\mathrm{dd}}(f)(x,t)\right)_j^2 \Bigg). \end{aligned} \tag{4.35}$$

As a result, the neural network will only learn with just over 1% of the data, which is a situation quite similar to that faced in industrial applications, where the exact measurement of variables at cell level is difficult.

We also define the test loss as:

$$\mathcal{L}_{\mathrm{boundary}}^{\mathrm{test}}(f) := \frac{1}{|M_\Omega \backslash M_\Omega^{ic,bc}|} \sum_{(x,t) \in M_\Omega \backslash M_\Omega^{ic,bc}} \left\| D^{\mathrm{dd}}(f)(x,t) \right\|_2^2, \tag{4.36}$$

with $D^{\mathrm{dd}}$ as in Equation (4.23), that accounts for the "unseen" data, in order to evaluate capabilities.

*Remark* 4.4. This former loss has also the same weight correction, but we omit it in the notation for the sake of simplicity.

We then train the same neural network as defined in Equation (4.28) according to the loss defined in Equation (4.35), with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and $T_{\max} = 400\,\text{s}$. The training will be performed with 15000 steps of the Adam algorithm, initialized with $(\delta, \beta, \gamma) = (10^{-3}, 0.9, 0.999)$ (*cf.* Definition 3.10), followed by 1000 steps of L-BFGS algorithm. We also apply a scheduling heuristic that decays $\delta$ to $10^{-4}$ from step 500 and to $10^{-5}$ from step 4000 during the Adam training. We plot the test and train loss in Figure 10 below.

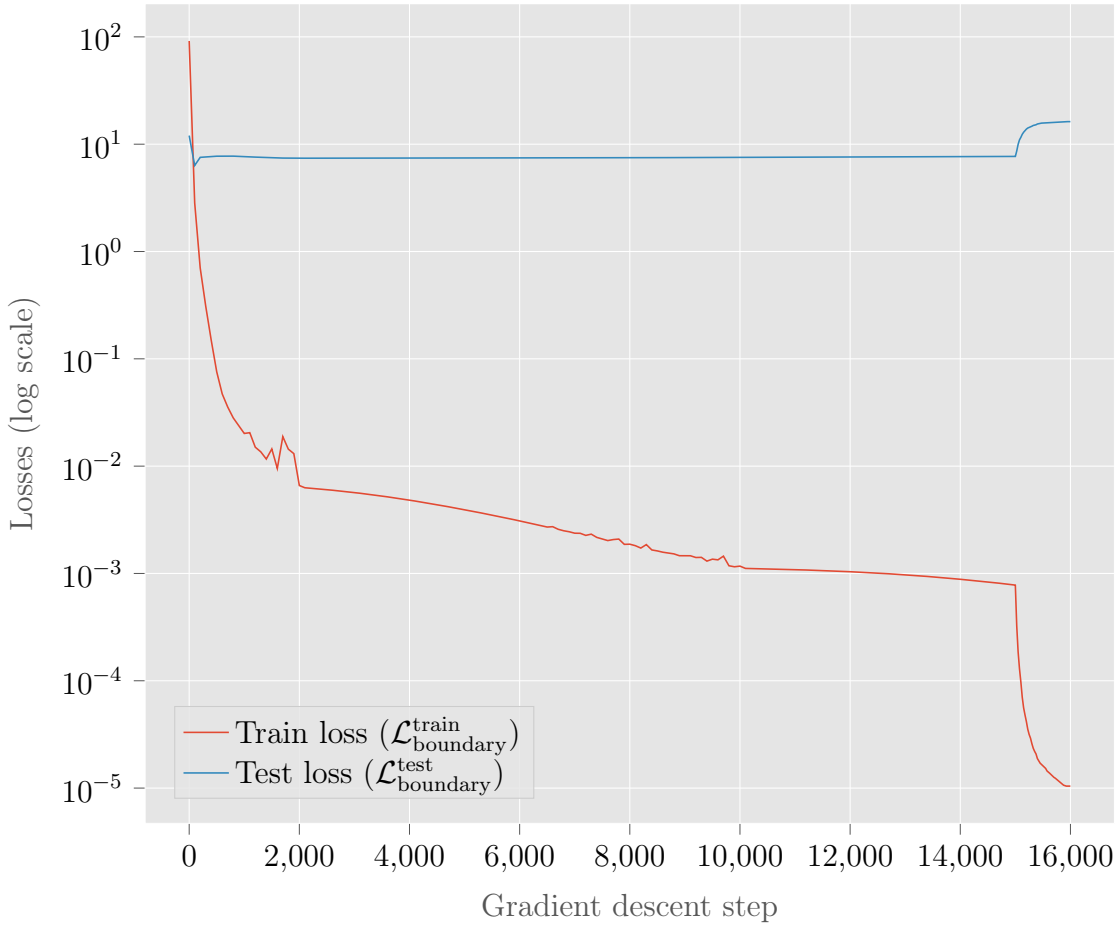
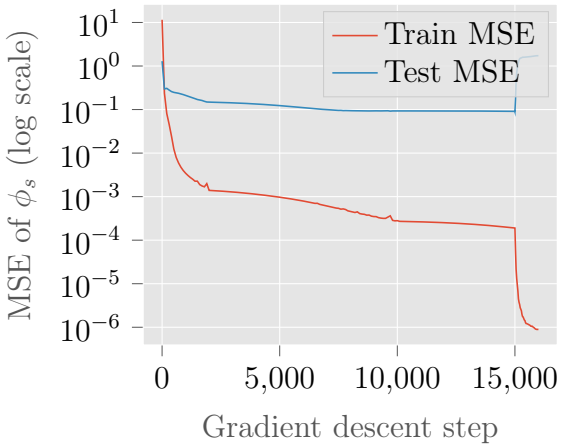
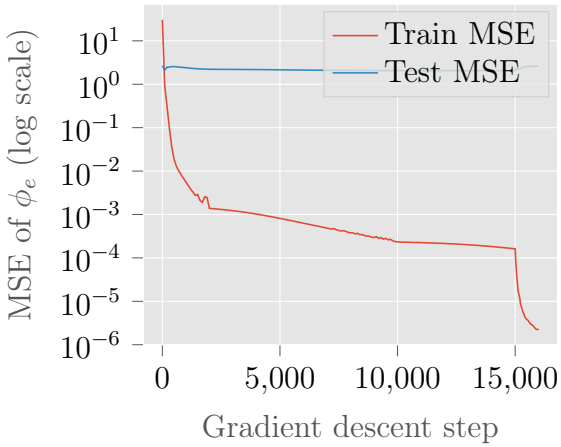



Figure 10: Train loss ($\mathcal{L}^{\text{train}}_{\text{boundary}}$) and Test loss ($\mathcal{L}^{\text{test}}_{\text{boundary}}$) for the data driven approach with boundary data, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that, while train loss lose 7 orders of magnitude during training, the test loss remains essentially constant and get even worse during the L-BFGS phase, which means that the neural network does not generalize well, or even does not generalize at all.

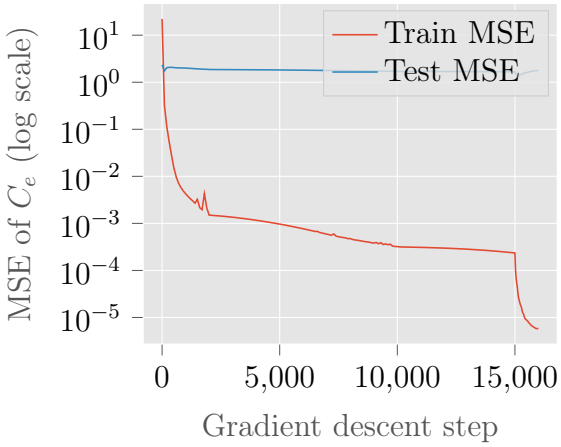
As observed in Figure 10, we see that the neural network doe not generalize in this case. We will also investigate the variable by variable MSEs in Figure 11.



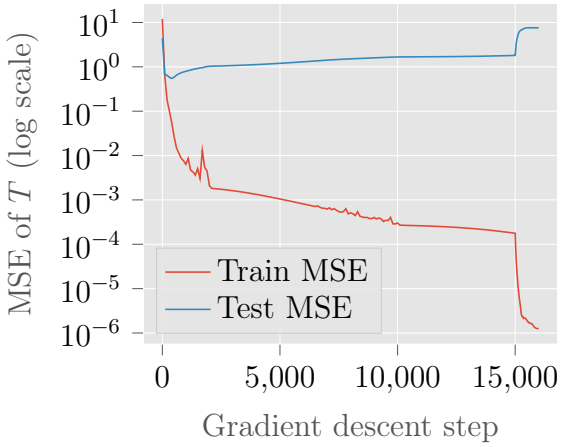

(a) MSE of solid potential ($\phi_s$)



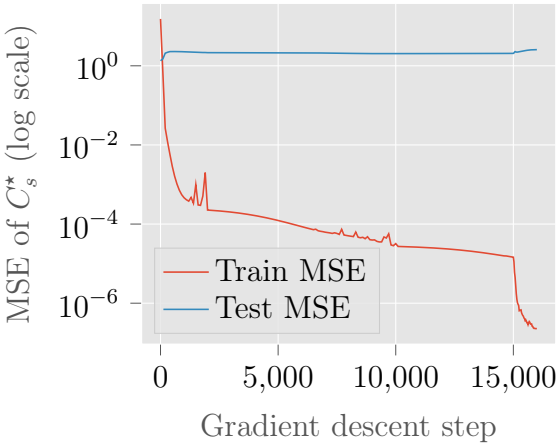

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)




(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

Figure 11: Train and Test MSEs of each of the variables of the model for the data driven approach with boundary data, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We can see that lack of generalization observed in Figure 10 is a phenomenon that affects all variables equally.

These results argue in favor of a different approach that would enable to generalize further. In the following section, we show that the use of PINNs method successfully addresses this problem.

## 4.3. Approximation of the P2D model in intermediate and long timescale

This section presents the first important contribution of our work. We show that with a fraction of the data from LIONSIMBA, the neural network defined in Equation (4.28) learns the equations of the P2D model with high accuracy for a time domain of the order of a hundred seconds. We begin by briefly introducing the losses used, carefully explaining the choice of recalibration weights, before detailing the results obtained for different choices of the applied current density function $I_{\text{app}}$ (*cf.* Table 7). Finally, we will show how to generalize these results to longer simulation times.

### 4.3.1. Loss defintion

First of all, we will take the same boundary condition as the one used in Section 4.2.5, namely: for all $i \in \{a, p, s, n, z\}$ the operators:

$$\hat{B}_i^{bc,\text{dd}} : \begin{cases} \mathcal{C}^2\left(\Omega_i \to \mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\Omega_i \to \mathbb{R}\right)^{m_i} \\ \left(u_j\right)_{1\leqslant j\leqslant m_i} & \longmapsto & \left(x \in \Omega_i \mapsto \sum_{y\in \hat{M}_{\Omega_i}^{bc}} \delta_y(x)\left(u_j(x) - \upsilon_j^{\text{LS},i}(y)\right)\right)_{1\leqslant j\leqslant m_i} \end{cases},$$

and:

$$B_i^{bc,\text{dd}} : \begin{cases} \mathcal{C}^2\left(\Omega_i \to \mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\Omega_i \to \mathbb{R}\right)^{m_i} \\ \left(u_j\right)_{1\leqslant j\leqslant m_i} & \longmapsto & \left(x \in \Omega_i \mapsto \sum_{y\in M_{\Omega_i}^{bc}} \delta_y(x)\left(u_j(x) - \upsilon_j^{\text{LS},i}(y)\right)\right)_{1\leqslant j\leqslant m_i} \end{cases},$$

as well as the operator:

$$B_i^{ic,\text{dd}} : \begin{cases} \mathcal{C}^2\left(\partial\Omega_i \to \mathbb{R}\right)^{m_i} & \longrightarrow & \mathcal{F}\left(\partial\Omega_i \to \mathbb{R}\right)^{m_i} \\ \left(u_j\right)_{1\leqslant j\leqslant m_i} & \longmapsto & \left(x \in \Omega_i \mapsto \sum_{y\in M_{\Omega_i}^{ic}} \delta_y(x)\left(u_j(x) - \upsilon_j^{\text{LS},i}(y)\right)\right)_{1\leqslant j\leqslant m_i} \end{cases},$$

For the operator $D$, we will take the one defined in Equation (4.5) of Section 4.2.5.

However, to define a loss, we will pay particular attention in choosing a weighting mechanism. Indeed, Rahaman et al. (2019) have shown that neural networks are subject to the phenomenon of "spectral bias". In a nutshell, given the Fourier series decomposition of a function $f$ to be approximated, Rahaman et al. (2019) show that a neural network trained by gradient descent will tend to concentrate on learning the low frequencies of $f$ (*i.e.* its first Fourier coefficients), which can result in a significant loss of information if $f$ also carries information in its higher frequencies (*i.e.* if $f$ admits Fourier coefficients of non-negligible values for higher frequencies). Wang et al. (2022c) have shown that this is even more the case when we train our neural network with PINNs methods. To prove this, they derived, thanks to the Neural Tangent Kernel (NTK) introduced by Jacot et al. (2018), an explicit training dynamics in the asymptotic limit where layers are of infinite size. They also propose a methof of correcting this bias by explicitly calculating the NTK, but this is far too costly in computation time for our problem. McClenny and Braga-Neto (2022) have introduced another mechanism that corrects this bias, by applying trainable weights

on each sampled point of the domain and updating them according to an adversarial mechanism somewhat similar to the one introduced by Goodfellow et al. (2014). In the following, we will show how to slightly adapt this mechanism, which will enable us to obtain our results. Let us start by formally introduce the self-adaptive mechanism of McClenny and Braga-Neto (2022).

**Self-adaptive mechanism:** Let be $m_D \in \mathbb{N}_1$ and $(y_k)_{1\leqslant k\leqslant m_D} \in \Omega^{m_D}$ where $(y_k)_{1\leqslant k\leqslant m_D}$ is typically an LHS outcome with $m_D$ samples. Let then be $\nu \in \mathbb{R}^{m_D}$, and a self-adaptive activation $\alpha \in \mathcal{C}^1(\mathbb{R} \to [0, +\infty))$. We can then define the self-adaptive loss:

$$\mathcal{L}^{\mathrm{SA}}(\nu)(f) := \frac{1}{m_D}\sum_{k=1}^{m_D} \alpha(\nu_k) \left\| D(f)(y_k) \right\|_2^2 . \tag{4.37}$$

The idea of McClenny and Braga-Neto (2022) is then, given $\nu \in \mathbb{R}^{m_D}$, to combine the usual training of $\mathcal{L}^{\mathrm{SA}}(\nu) \circ \tilde{\Psi}$ by gradient descent (*cf.* Section 3.2.1), where $\tilde{\Psi}$ is defined in Equation (4.28), with a second training mechanism that, given $\theta \in \mathbb{R}^P$, trains the weights $\nu$ to maximize $\mathcal{L}^{\mathrm{SA}}(\cdot)\left(\tilde{\Psi}(\theta)\right)$. More precisely, given $\theta \in \mathbb{R}^P$, we apply a gradient descent step to the loss:

$$\ell^{\mathrm{SA}\prime} : \left\{ \begin{array}{ccc} \mathbb{R}^{m_D} & \longrightarrow & [0, +\infty) \\ \varpi & \longmapsto & -\mathcal{L}^{\mathrm{SA}}(\varpi)\left(\tilde{\Psi}(\theta)\right) \end{array} \right. .$$

We thus obtain a process in which two sequences $(\nu_n)_{n\in\mathbb{N}_1}$ and $(\tilde{\theta}_n)_{n\in\mathbb{N}_1}$ are iteratively defined, given $n \in \mathbb{N}$, by the successive application of the gradient descent to the classical loss:

$$\ell^{\mathrm{SA}} : \left\{ \begin{array}{ccc} \mathbb{R}^{P} & \longrightarrow & [0, +\infty) \\ \theta & \longmapsto & \mathcal{L}^{\mathrm{SA}}(\nu_n)\left(\tilde{\Psi}(\theta)\right) \end{array} \right. ,$$

yielding $\tilde{\theta}_{n+1}$, followed by a second gradient descent applied to the loss:

$$\ell^{\mathrm{SA}\prime} : \left\{ \begin{array}{ccc} \mathbb{R}^{m_D} & \longrightarrow & [0, +\infty) \\ \nu & \longmapsto & -\mathcal{L}^{\mathrm{SA}}(\nu)\left(\tilde{\Psi}(\tilde{\theta}_{n+1})\right) \end{array} \right. ,$$

yielding $\nu_{n+1}$. The only remaining element to be defined is the choice of $\nu_0$ weights at initialization. This is usually done randomly, with a distribution chosen according to $\alpha$. A typical choice is $\alpha = \mathrm{sigmoid}$ (*cf.* Table 16) with an uniform distribution in $[0, 1]$.

We will now adapt this mechanism, by combining it with a weight correction as introduced in Section 4.2.4. In this case, we cannot rely anymore on weights computed from LIONSIMBA, since LIONSIMBA does not give information about the orders of magnitude of the operator $D$, which is different from the one it uses. We then chose to apply the following heuristic which proves to work in practice. Given $m \in \mathbb{N}_1$, we sample for all $i \in \{a, p, s, n, z\}$, $m$ points $(z_k^i)_{1\leqslant k\leqslant m}$ in $\Omega_i$ with a LHS procedure. Then given the initialization $\theta_0 \in \mathbb{R}^P$ of the neural network, we set the weights: for all $i \in \{a, p, s, n, z\}$, for all $1 \leqslant j \leqslant m_i$

$$\omega_j^{D,i} := \frac{1}{m}\sum_{k=1}^{m} \left( D\left(\tilde{\Psi}(\theta_0)\right)(z_k^i)\right)_j^2 . \tag{4.38}$$

with for all $i \in \{a, p, s, n, z\}$, $m_i$ being the number of equations in section $i$ (*cf.* Section 4.1.1).

Now, given a number of samples $m_D \in \mathbb{N}_1$, LHS sampled points $(y_k^i)_{1\leqslant k\leqslant m_D} \in (\Omega_i)^{m_D}$, a self-adaptive activation $\alpha \in \mathcal{C}^1(\mathbb{R} \to [0, +\infty))$, and weights:

$$\left(\left(\left(\nu_k^{\Omega_i,j}\right)_{1\leqslant j\leqslant m_i}\right)_{1\leqslant k\leqslant m_D}\right)_{i\in\{a,p,s,n,z\}} \in \left(\prod_{i\in\{a,p,s,n,z\}} \mathbb{R}^{m_i}\right)^{m_D}$$

with for all $i \in \{a, p, s, n, z\}$, $m_i$ being again the number of equations in section $i$ (*cf.* Section 4.1.1), we define the self-adaptive loss with weight correction for $D$ as:

$$\mathcal{L}_D^{\mathrm{SA}}(f) := \sum_{i\in\{a,p,s,n,z\}} \frac{1}{m_D} \sum_{k=1}^{m_D} \sum_{j=1}^{m_i} \alpha\left(\nu_k^{\Omega_i,j}\right) \left(\omega_j^{D,i}\right)^{-1} \left(D(f)(y_k^i)\right)_j^2, \tag{4.39}$$

where we omitted the dependence to $\nu$ weights for the sake of simplicity, and typicall used $m = \frac{m}{m_D}$ to compute the weights $\left(\left(\omega_j^{D,i}\right)_{1\leqslant j\leqslant m_i}\right)_{i\in\{a,p,s,n,z\}}$ defined in Equation (4.38) above. We will also extend the self-adaptive mechanism to operators $\hat{B}_i^{bc,\mathrm{dd}}$, $B_i^{bc,\mathrm{dd}}$ and $B_i^{ic,\mathrm{dd}}$ defined above. To this end, we are giving ourselves the following additional self-adaptive weights:

- $\left(\left(\nu_k^{\hat{M}_{\Omega_i}^{bc}}\right)_{1\leqslant k\leqslant|\hat{M}_{\Omega_i}^{bc}|}\right)_{i\in\{a,p,s,n,z\}} \in \prod_{i\in\{a,p,s,n,z\}} \mathbb{R}^{|\hat{M}_{\Omega_i}^{bc}|}$,
- $\left(\left(\nu_k^{M_{\Omega_i}^{bc}}\right)_{1\leqslant k\leqslant|M_{\Omega_i}^{bc}|}\right)_{i\in\{a,p,s,n,z\}} \in \prod_{i\in\{a,p,s,n,z\}} \mathbb{R}^{|M_{\Omega_i}^{bc}|}$,
- $\left(\left(\nu_k^{M_{\Omega_i}^{ic}}\right)_{1\leqslant k\leqslant|M_{\Omega_i}^{ic}|}\right)_{i\in\{a,p,s,n,z\}} \in \prod_{i\in\{a,p,s,n,z\}} \mathbb{R}^{|M_{\Omega_i}^{ic}|}$.

We can then finally define our final loss:

$$\begin{aligned}
\mathcal{L}_{\mathrm{P2D}}^{\mathrm{SA}}(f) := \sum_{i\in\{a,p,s,n,z\}} \Bigg( & \frac{1}{m_D} \sum_{k=1}^{m_D} \sum_{j=1}^{m_i} \alpha\left(\nu_k^{\Omega_i,j}\right) \left(w_j^i\right)^{-1} \left(D(f)(y_k^i)\right)_j^2 \\
& + \frac{1}{|\hat{M}_{\Omega_i}^{bc}|} \sum_{(x,t)\in\hat{M}_{\Omega_i}^{bc}} \sum_{j=1}^{m_i} \alpha\left(\nu_k^{\hat{M}_{\Omega_i}^{bc}}\right) \left(w_j^i\right)^{-1} \left(\hat{B}_i^{bc,\mathrm{dd}}(f)(x,t)\right)_j^2 \\
& + \frac{1}{|M_{\Omega_i}^{bc}|} \sum_{(x,t)\in M_{\Omega_i}^{bc}} \sum_{j=1}^{m_i} \alpha\left(\nu_k^{M_{\Omega_i}^{bc}}\right) \left(w_j^i\right)^{-1} \left(B_i^{bc,\mathrm{dd}}(f)(x,t)\right)_j^2 \\
& + \frac{1}{|M_{\Omega_i}^{ic}|} \sum_{(x,t)\in M_{\Omega_i}^{ic}} \sum_{j=1}^{m_i} \alpha\left(\nu_k^{M_{\Omega_i}^{ic}}\right) \left(w_j^i\right)^{-1} \left(B_i^{ic,\mathrm{dd}}(f)(x,t)\right)_j^2 \Bigg).
\end{aligned} \tag{4.40}$$

Now that our loss is defined, we will take a closer look at the results obtained with it in the next section.

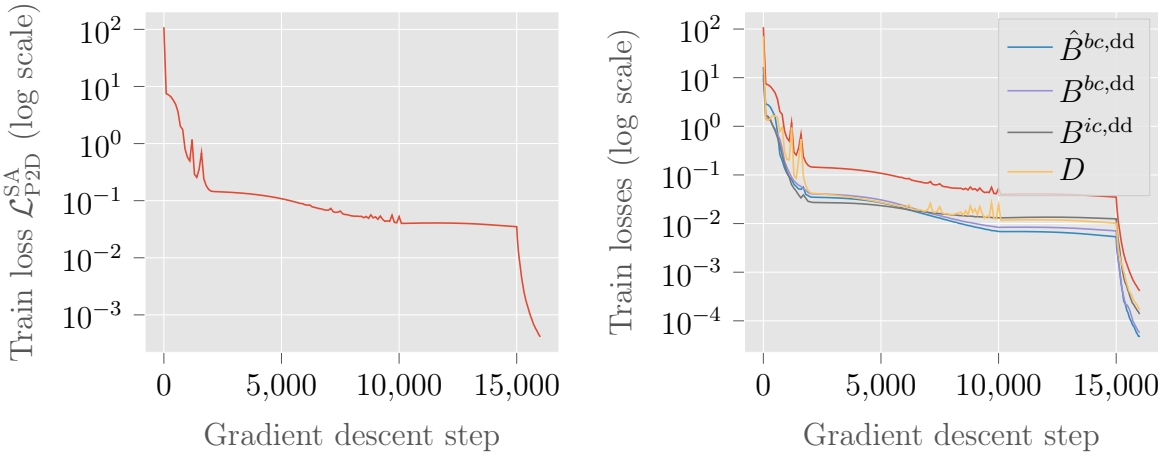
### 4.3.2. Results of the approximation in intermediate timescale

In this section we show that the results obtained when training a neural network with PINNs on the PDEs of the P2D model with initial conditions and boundary conditions data coming from LIONSIMBA are very accurate. As noted in Section 4.2.5, initial conditions and boundary conditions data represents just over one percent of the available data in LIONSIMBA, which is very little for a typical deep learning problem, which shows the interest of the method. More precisely, as we established in Section 4.3.1, we will train the neural network defined in Equation (4.28) with the loss defined in Equation (4.40).

We will first evaluate our results by plotting the training loss as in Figure 5 with a second plot showing the contribution made by each component of the loss, namely the PDEs (operator $D$), each boundary condition (operators $\hat{B}_i^{bc,\mathrm{dd}}$ and $B_i^{bc,\mathrm{dd}}$), and initial conditions (operators $B_i^{ic,\mathrm{dd}}$). We will then plot the MSEs of the different variables as in Figure 6, and the PDEs MSEs as in Figure 9. Finally we will compare the heatmaps of the approximations derived from our model and those derived from LIONSIMBA, and plot the heatmaps of the absolute and relative differences as in **??**.

We will review the results obtained for $T_{\max} = 400\,\mathrm{s}$ with $I_{\mathrm{app}} = 40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7). For the sake of completeness, we also plot in Appendix F.3 results for different values of constant applied current densities[18] $I_{\mathrm{app}}$ (*cf.* Table 7), namely $I_{\mathrm{app}} \in \{-40\,\mathrm{A{\cdot}m^{-2}}, -20\,\mathrm{A{\cdot}m^{-2}}, 20\,\mathrm{A{\cdot}m^{-2}}, 40\,\mathrm{A{\cdot}m^{-2}}\}$, yielding similar results. The training will be performed with 15000 steps of the Adam algorithm, initialized with $(\delta, \beta, \gamma) = (10^{-3}, 0.9, 0.999)$ (*cf.* Definition 3.10), followed by 1000 steps of L-BFGS algorithm. We also apply a scheduling heuristic that decays $\delta$ to $10^{-4}$ from step 2000 and to $10^{-5}$ from step 10000 during the Adam training.

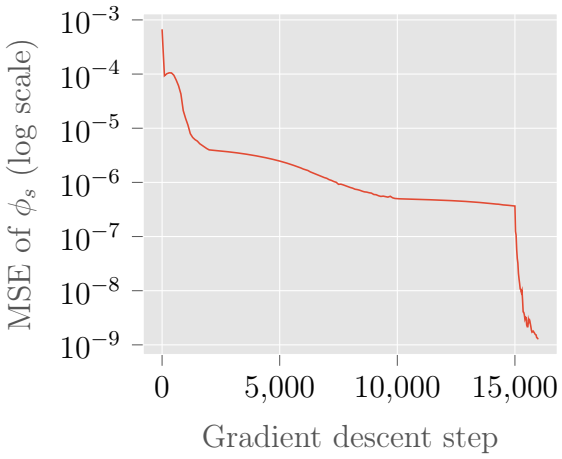
[18] which is the most frequent use case when charging batteries



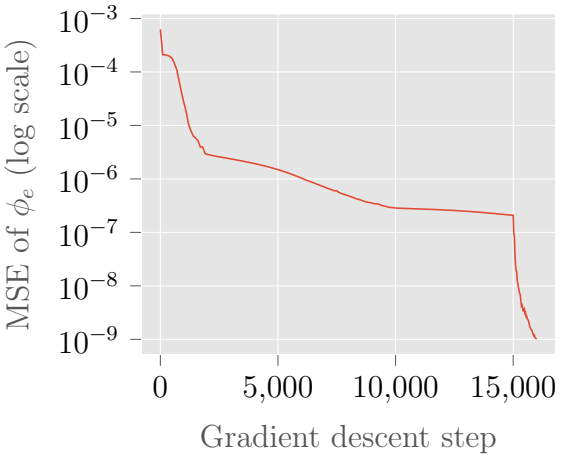

(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

(b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

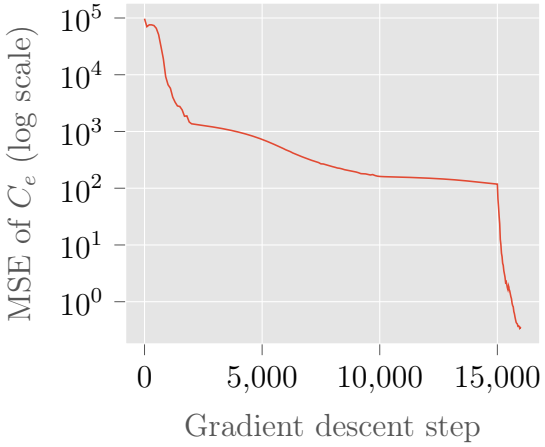
Figure 12: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the learning process succeeded, the loss losing 5 orders of magnitude during training, indicating that the neural network did learn both the boundary conditions data, and the losses from the P2D model PDEs.
We also find that each loss component contributes exactly the same orders of magnitude to the total loss, meaning that each component has been learned equally.
Finally, we note the effect of scheduling, with clear inflections of the loss slope at the 2000th and at the 10000th step, and of the change from Adam to L-BFGS at the 15000th step, with an even clearer change in the loss slope.



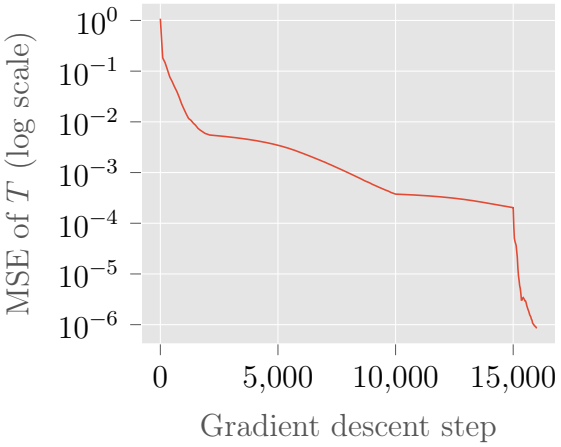

(a) MSE of solid potential ($\phi_s$)



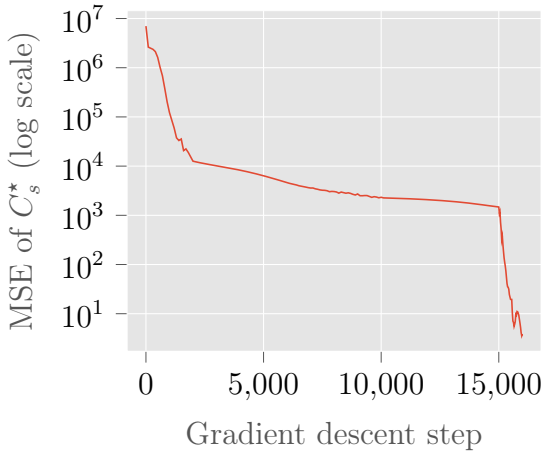

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

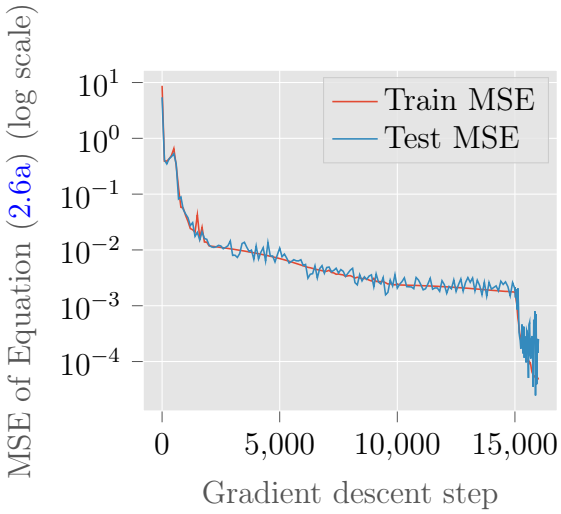



(d) MSE of temperature ($T$)

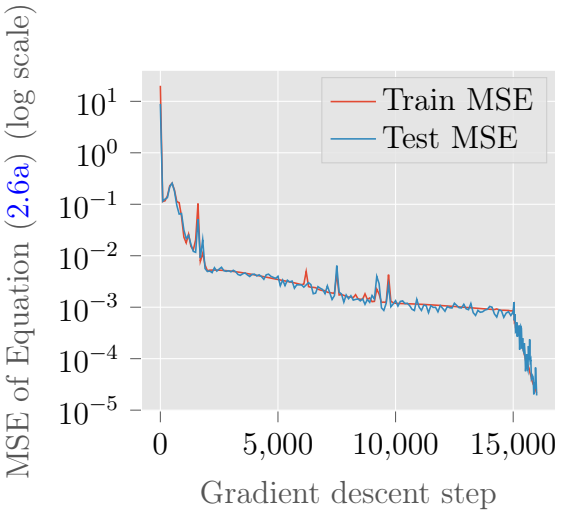



(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

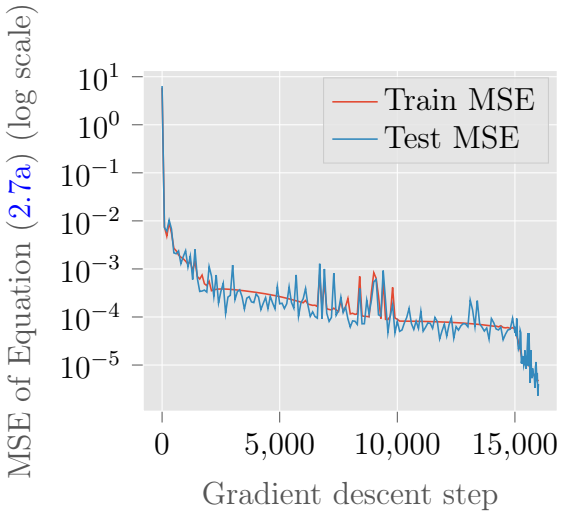
Figure 13: MSEs of each of the variables of the model for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000. We observe that all variables are equally learned by the neural networks, each MSE losing 6 orders of magnitude during the training process, which means that the neural network generalizes very well thanks to the equations enforced in the loss through operator $D$.



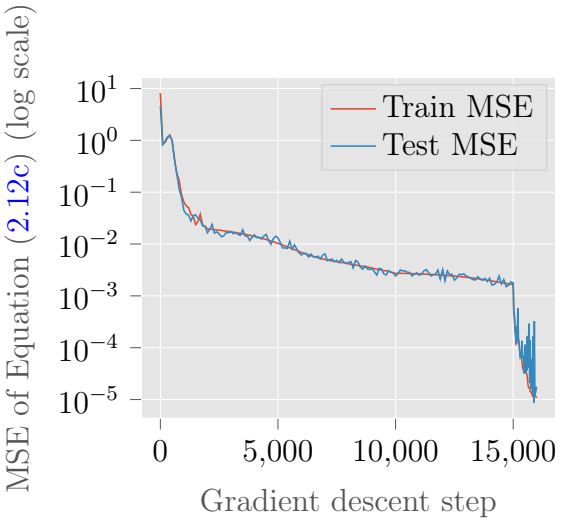

(a) MSE associated to lithium ions accumulation in the electrolyte in anode PDE (Equation (2.6a))



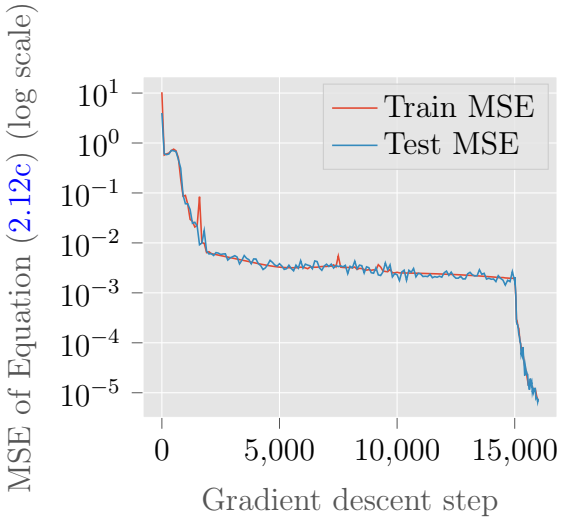

(b) MSE associated to lithium ions accumulation in the electrolyte in cathode PDE (Equation (2.6a))



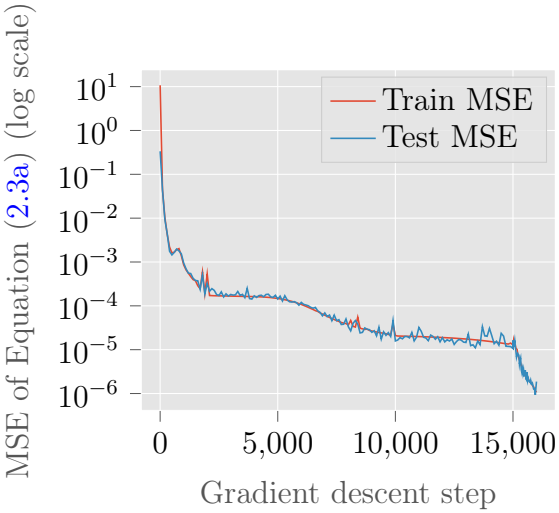

(c) MSE associated to lithium ions accumulation in the electrolyte in separator PDE (Equation (2.7a))



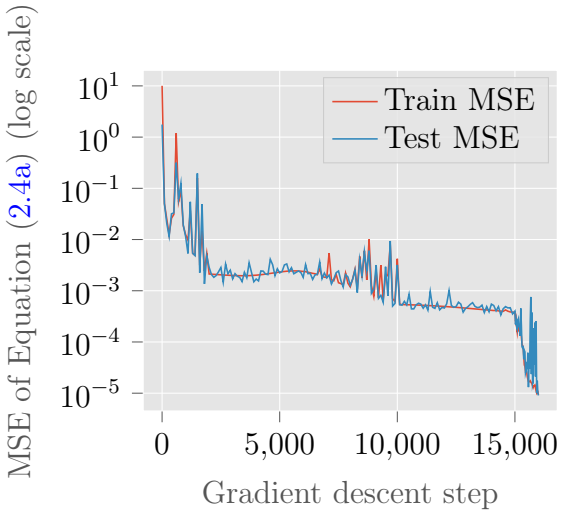

(d) MSE associated to lithium ions intercalation in solid particles in anode PDE (Equation (2.12c))



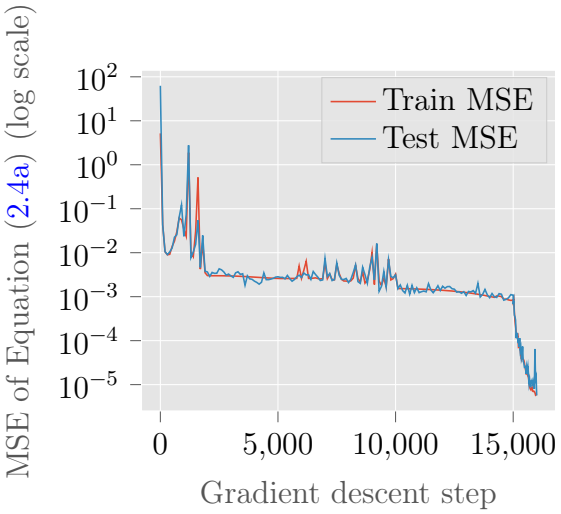

(e) MSE associated to lithium ions intercalation in solid particles in cathode PDE (Equation (2.12c))



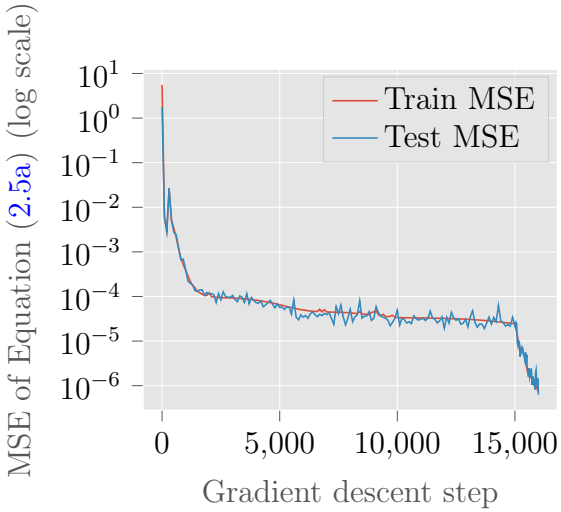

(f) MSE associated to temperature in positive current collector PDE (Equation (2.3a))



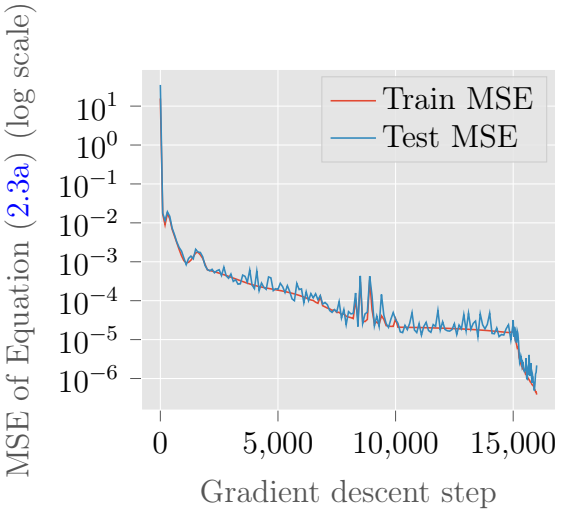

(g) MSE associated to temperature in anode PDE (Equation (2.4a))



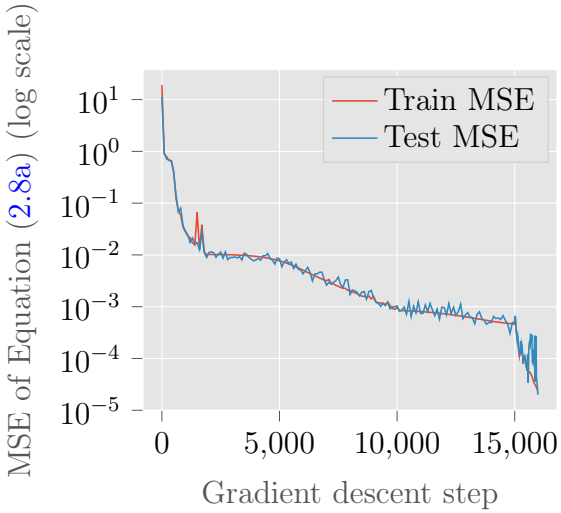

(h) MSE associated to temperature in cathode PDE (Equation (2.4a))



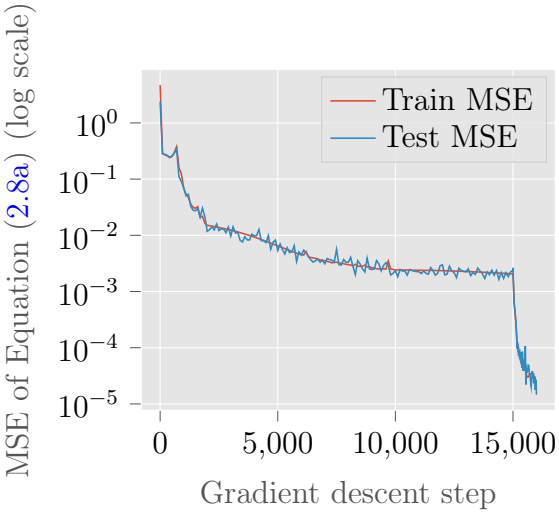

(i) MSE associated to temperature in separator PDE (Equation (2.5a))



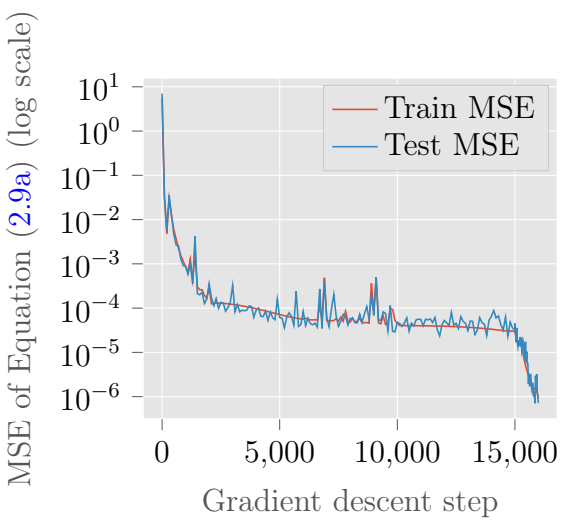

(j) MSE associated to temperature in negative current collector PDE (Equation (2.3a))



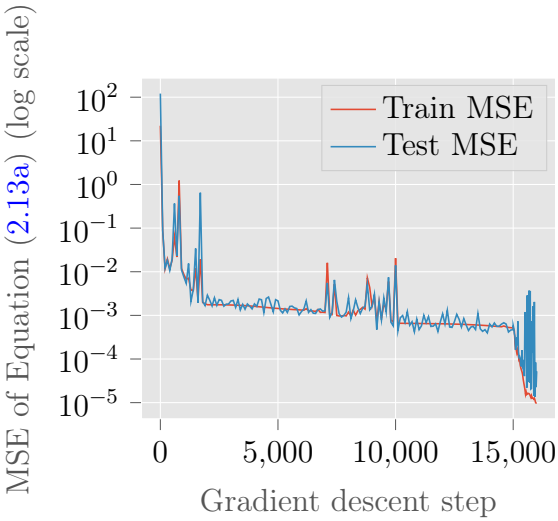

(k) MSE associated to lithium ions transportation in anode PDE (Equation (2.8a))



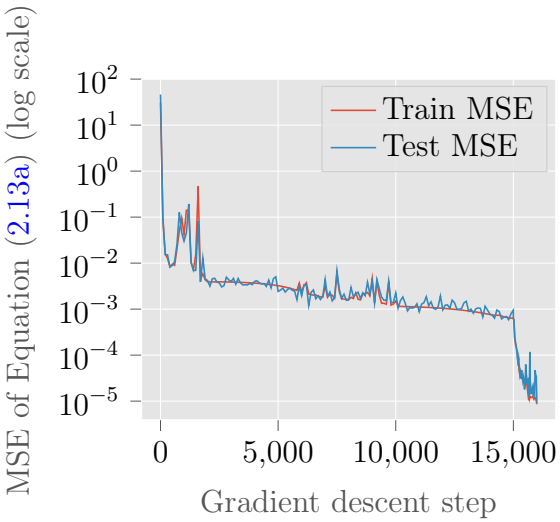

(l) MSE associated to lithium ions transportation in cathode PDE (Equation (2.8a))




(m) MSE associated to lithium ions transportation in separator PDE (Equation (2.9a))




(n) MSE associated to electrons motion in anode PDE (Equation (2.13a))

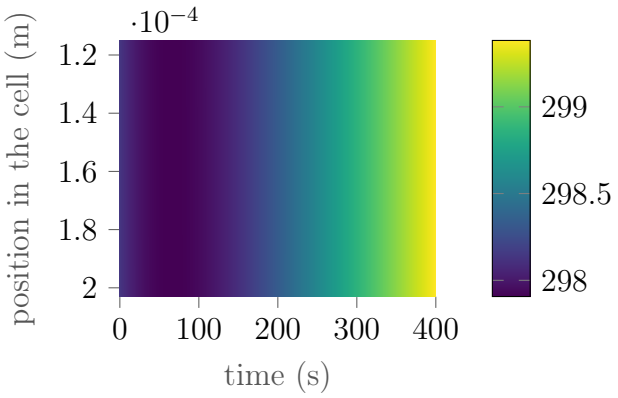



(o) MSE associated to electrons motion in cathode PDE (Equation (2.13a))

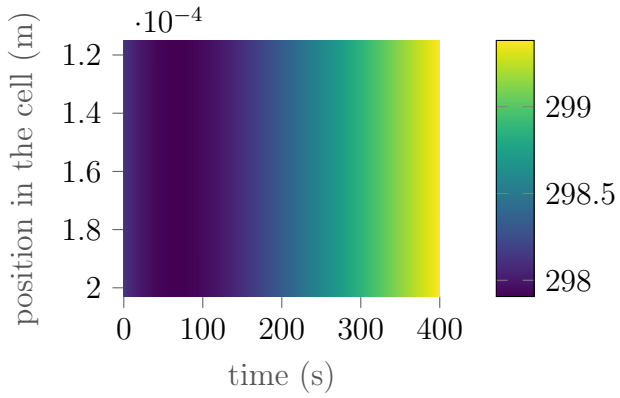
Figure 14: MSEs for each of the PDEs of the model for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
On the one hand, we observe that each equation is equally well learned, each MSE losing between 6 and 7 orders of magnitude during the training process, and on the other hand, that the learning of equations is also very well generalized over the whole domain, which is very satisfactorily, since it means that the physics of the problem has been very well learned by the neural network.

Finally we plot the heatmaps of the approximations derived from our model and those derived from LIONSIMBA, as well as the heatmaps of the absolute and relative differences for all variables in section $n$. For the sake of completeness, we also plot in Appendix F.3 these heatmaps in all sections. We cannot represent them for other values of $I_{\text{app}}$, because

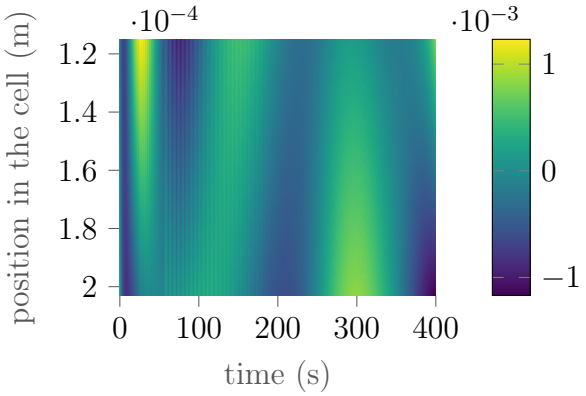
that would add almost 50 pages to this work.



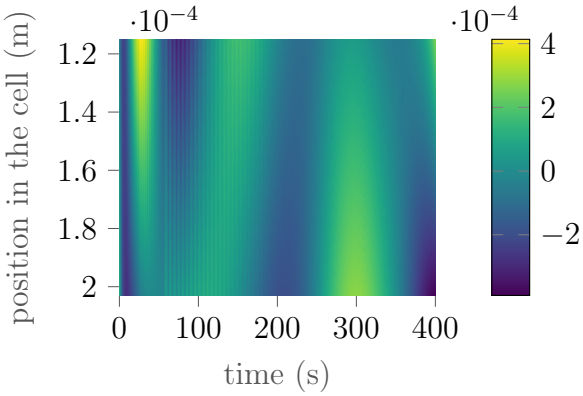

(a) Approximation of the temperature ($T$) in anode with a neural network




(b) Approximation of the temperature ($T$) in anode by LIONSIMBA

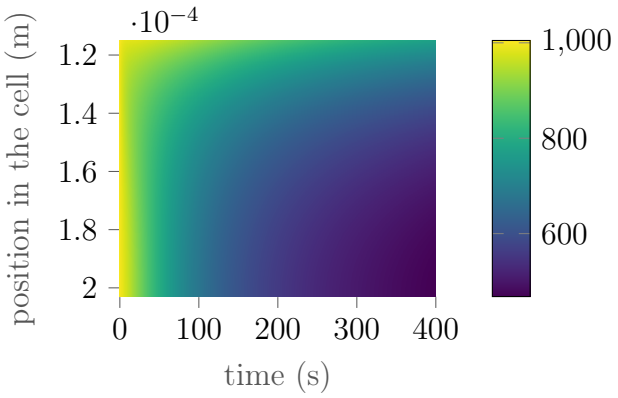



(c) Absolute error $T_{\text{LS}} - T_{\text{PINN}}$ of the temperature in anode

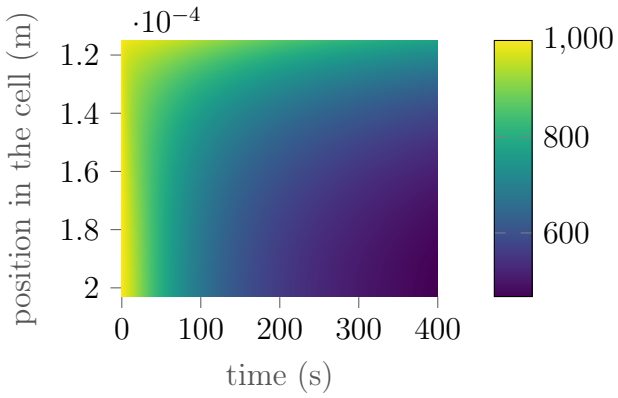



(d) Relative error $\frac{T_{\text{LS}} - T_{\text{PINN}}}{T_{\text{LS}}}$ in % of the temperature in anode

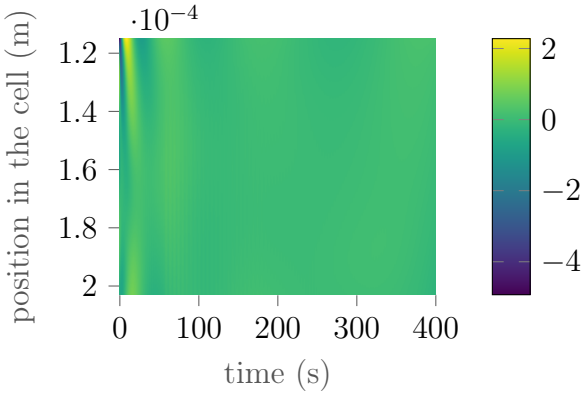
Figure 15: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.0005%.



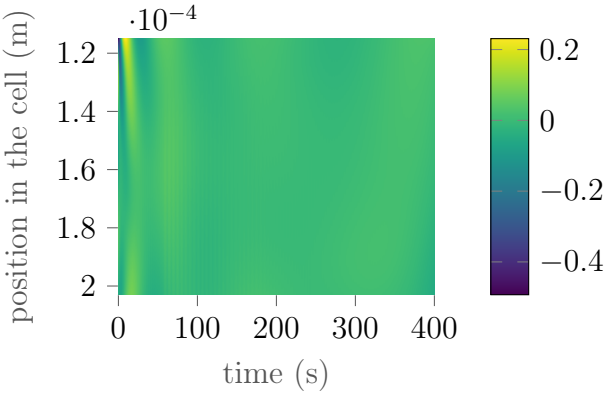

(a) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in anode with a neural network




(b) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in anode by LIONSIMBA

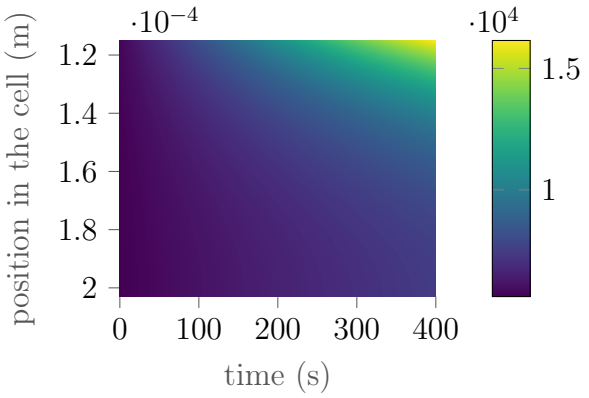



(c) Absolute error $C_{e\text{LS}} - C_{e\text{PINN}}$ of the lithium ions concentration in the electrolyte in anode

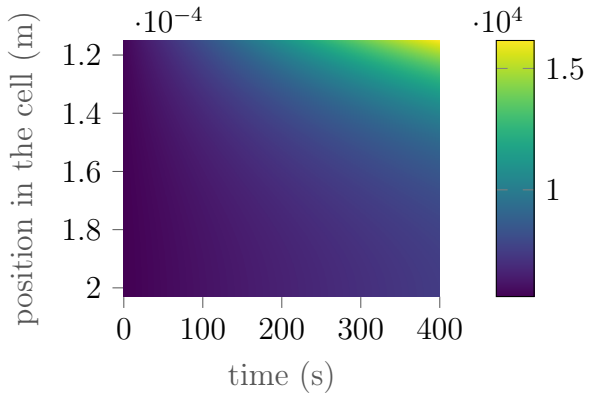



(d) Relative error $\frac{C_{e\text{LS}} - C_{e\text{PINN}}}{C_{e\text{LS}}}$ in % of the lithium ions concentration in the electrolyte in anode

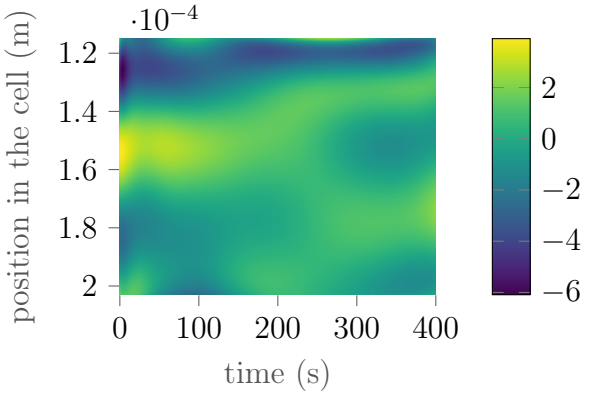
Figure 16: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.5%.



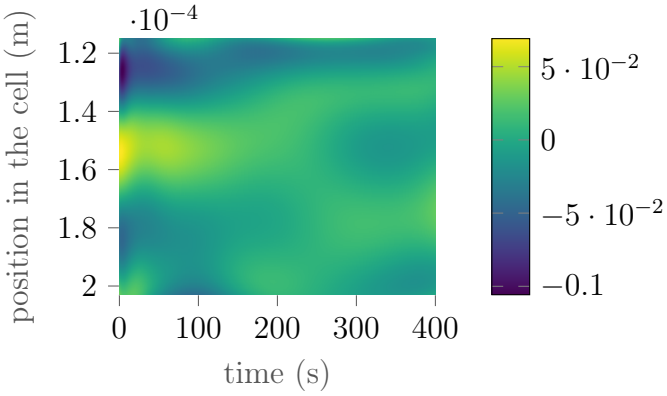

(a) Approximation of the lithium ions surface concentration in the solid-phase ($C_s^\star$) in anode with a neural network




(b) Approximation of the lithium ions surface concentration in the solid-phase ($C_s^\star$) in anode by LIONSIMBA

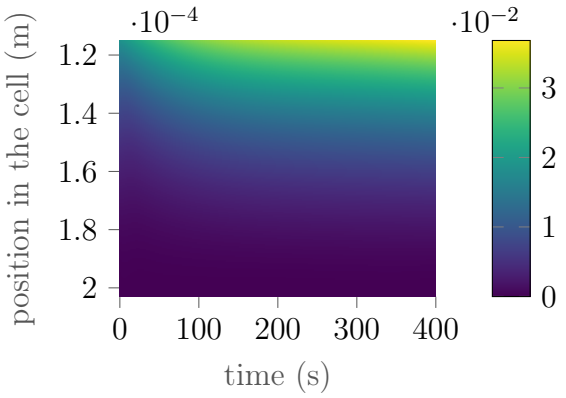



(c) Absolute error $C^\star_{s\,\mathrm{LS}} - C^\star_{s\,\mathrm{PINN}}$ of the lithium ions surface concentration in the solid-phase in anode

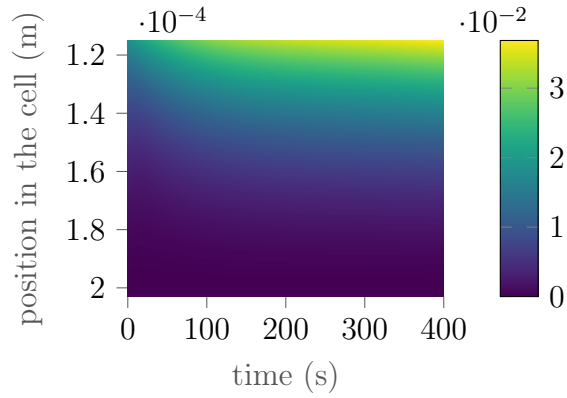



(d) Relative error $\frac{C^\star_{s\,\mathrm{LS}} - C^\star_{s\,\mathrm{PINN}}}{C^\star_{s\,\mathrm{LS}}}$ in % of the lithium ions surface concentration in the solid-phase in anode

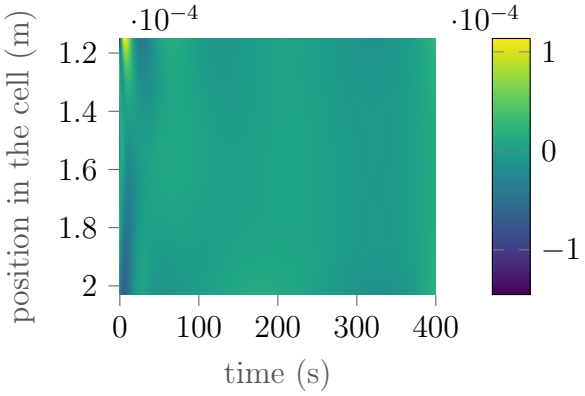
Figure 17: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\mathrm{app}} = 40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\mathrm{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.2%.



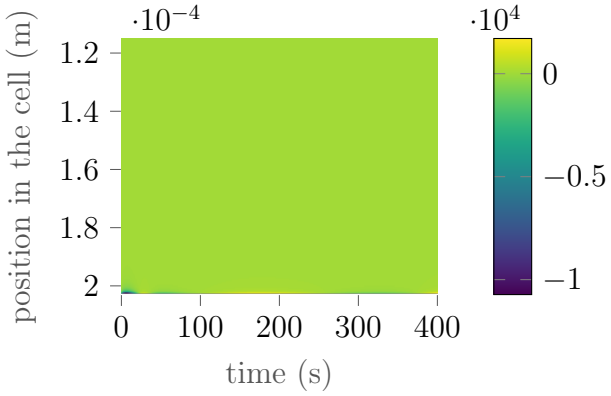

(a) Approximation of the electrolyte potential ($\phi_e$) in anode with a neural network




(b) Approximation of the electrolyte potential ($\phi_e$) in anode by LIONSIMBA

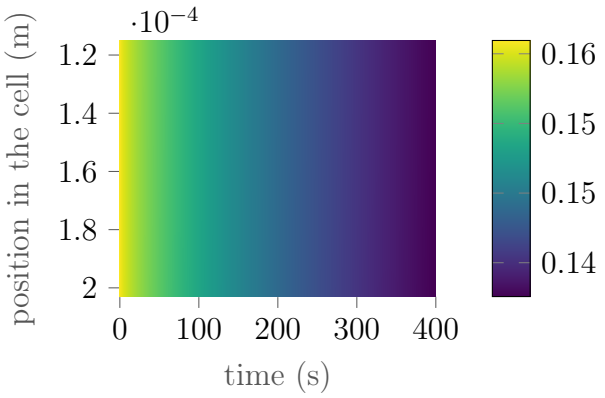



(c) Absolute error $\phi_{e\text{LS}} - \phi_{e\text{PINN}}$ of the electrolyte potential in anode

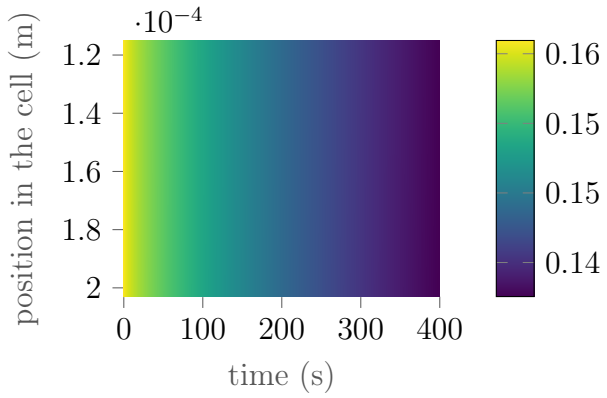



(d) Relative error $\frac{\phi_{e\text{LS}} - \phi_{e\text{PINN}}}{\phi_{e\text{LS}}}$ in % of the electrolyte potential in anode

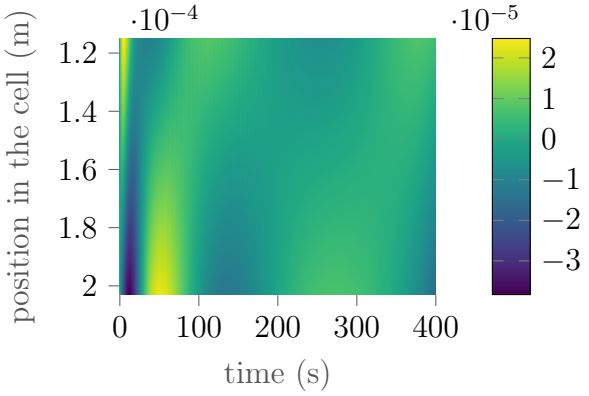
Figure 18: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
Here we see that the relative error with respect to LIONSIMBA seems very high, reaching almost $10^4$%. This is explained by the fact that $\phi_e$ is exactly equal to 0 on the end $x_n$ of the anode (here depicted in the lower part of the graph), as this is a boundary condition of the P2D model (*cf.* Equation (2.8c)). If, on the other hand, we focus on the absolute error, and compare it with the orders of magnitude of $\phi_s$ on the anode depicted in Figure 54b, we find that the effective relative error, outside a neighborhood close to $\{x_n\}[0, T_{\max}]$, is rather of the order of 1%.



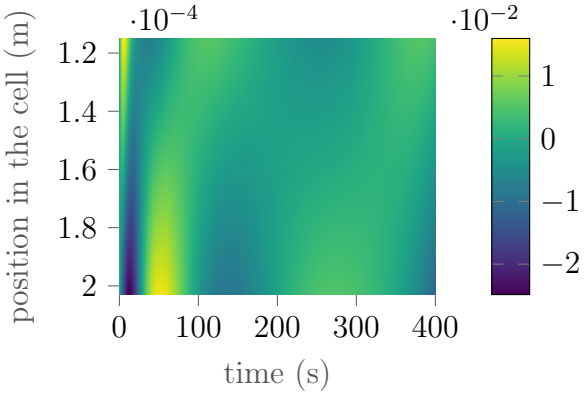

(a) Approximation of the solid potential ($\phi_s$) in anode with a neural network




(b) Approximation of the solid potential ($\phi_s$) in anode by LIONSIMBA




(c) Absolute error $\phi_{s\text{LS}} - \phi_{s\text{PINN}}$ of the solid potential in anode




(d) Relative error $\frac{\phi_{s\text{LS}} - \phi_{s\text{PINN}}}{\phi_{s\text{LS}}}$ in % of the solid potential in anode

Figure 19: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.03%.

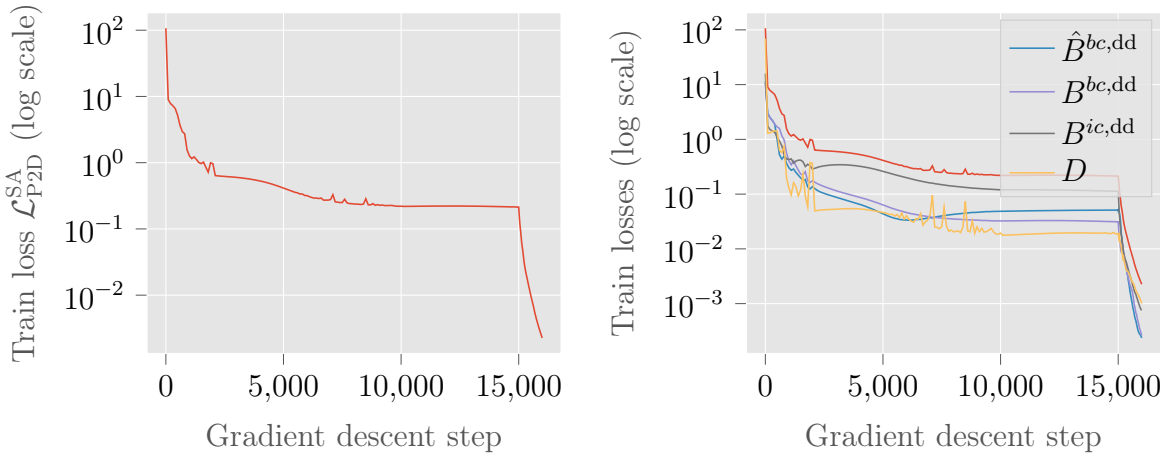
### 4.3.3. Extension of the results to the approximation in long timescale

Due to the limited approximation power of a neural network with finite widths, we cannot directly scale our method to timescales greater than a few hundreds of seconds, 400 seeming to be the empirical limit. Of course, we could take neural networks with larger widths and depth, but this results in a more complex neural network, that is harder to train, which makes it not a viable option. We then chose to extend the intermediate timescale results obtained in Section 4.3.2, by the use of the eXtended PINN (XPINN) technique introduced by Jagtap and Karniadakis (2021). Namely, given the domain $\Omega$ in which we want to approximate the solution of a PDE with a neural network, Jagtap

and Karniadakis (2021) suggest to partition $\Omega$ in pairwise disjoint subdomains $(\Omega_i)_{i\in I}$ and then to solve the problem separately in each $\Omega_i$. Of course this is not possible for every $\Omega_i$, since there is a certain dependency between them. Nevertheless Jagtap and Karniadakis (2021) suggest to begin by approximating the problem's solution on subdomains $\Omega_i$ that contain initial conditions and boundary conditions domain, and then to use those results to approximate the solution of the problem on remaining subdomains $\Omega_i$ by enforcing continuity of the function, or even of derivatives and higher order derivatives.

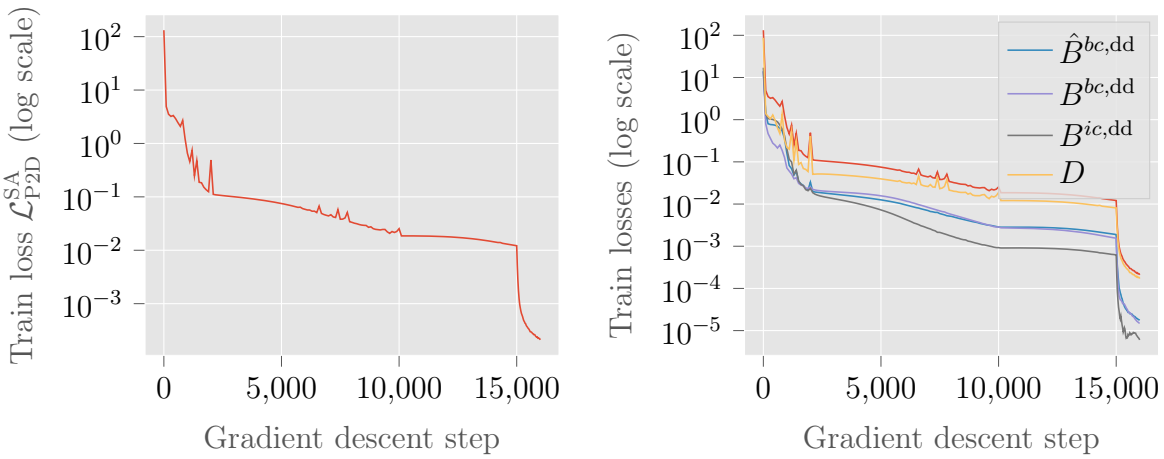
In our case, we will apply this procedure by simply partitioning the P2D domain $\Omega$ along its time axis. More formally, given $\Omega = [x_0, x_z] \times [0, T_{\max}]$, we will partition $\Omega$ into $([X_0, x_z] \times [T_{i-1}, T_i])_{1\leqslant i\leqslant k}$ with $k \in \mathbb{N}_1$, $T_0 = 0$ and for all $1 \leqslant i \leqslant k$, $T_i \in [0, +\infty)$ such that $T_i - T_{i-1}$ be of the order of 100 s. We illustrate this procedure for $T_{\max} = 1200\,\mathrm{s}$, with $T_1 = 400\,\mathrm{s}$, $T_2 = 800\,\mathrm{s}$ and $T_3 = T_{\max} = 1200\,\mathrm{s}$ with an applied current density $I_{\text{app}} = -40\,\mathrm{A{\cdot}m^{-2}}$. The training procedure is chosen exactly as in Section 4.3.2. The results are presented in Figures 20 to 25 below.

We observe that the results obtained show the same accuracy rate as the results in Section 4.3.2, which validates this method for extension to longer time scales.



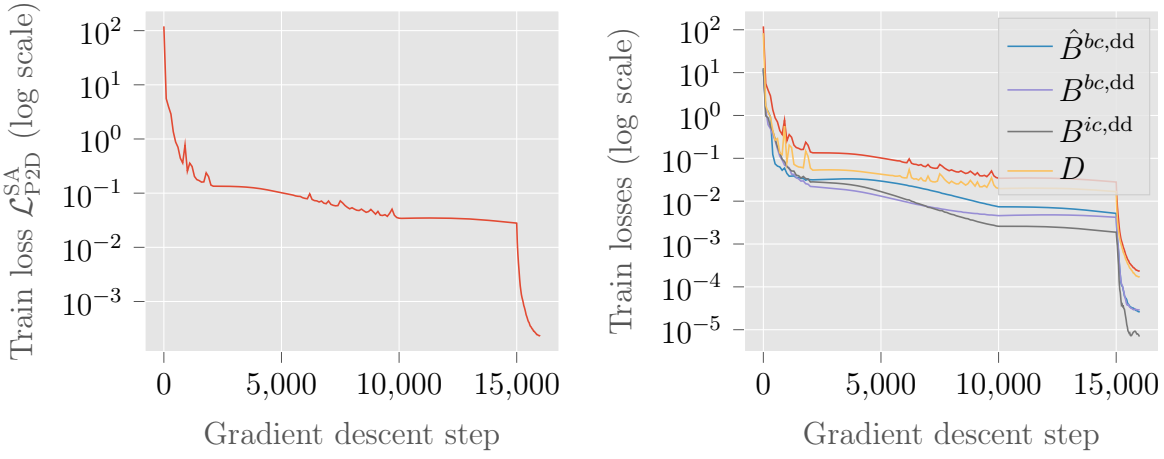

(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ (b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

Figure 20: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the XPINN approach between times 0 s and 400 s, with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7), with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.



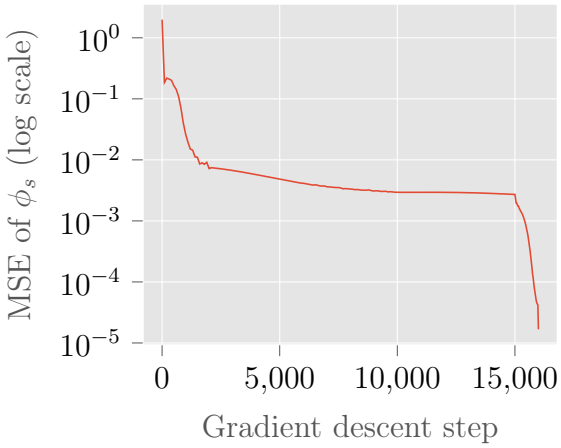

(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ (b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

Figure 21: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the XPINN approach between times 400 s and 800 s, with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7), with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.

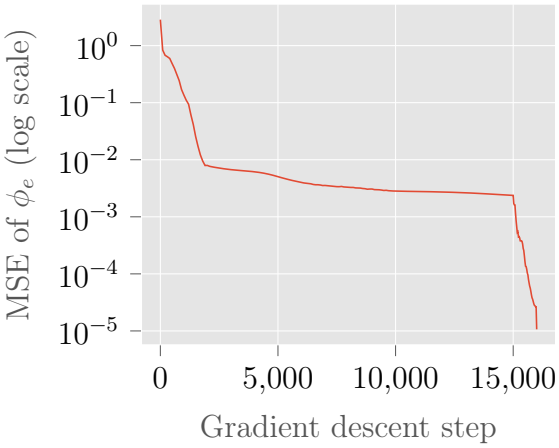



(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ (b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

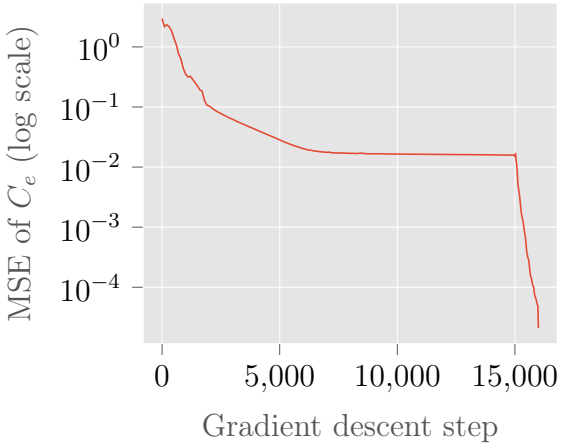
Figure 22: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the XPINN approach between times 800 s and 1200 s, with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7), with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.



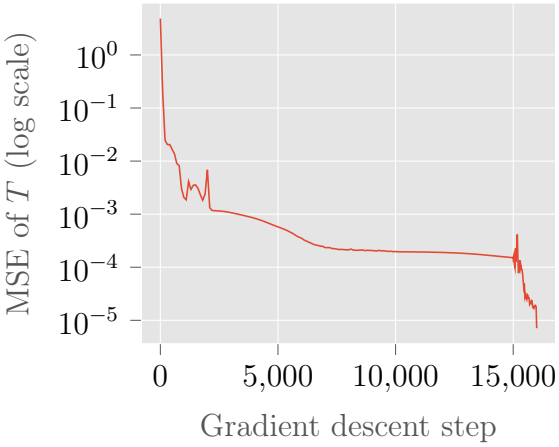

(a) MSE of solid potential ($\phi_s$)



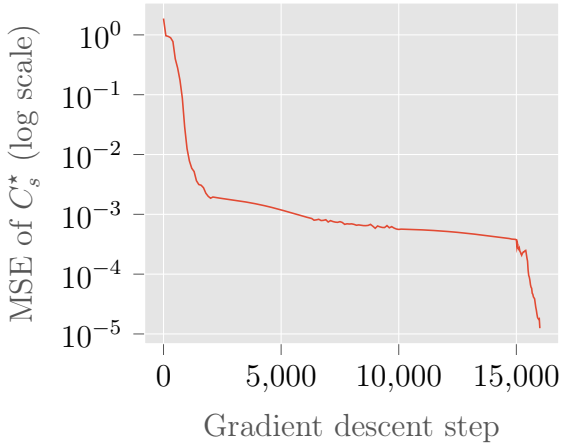

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

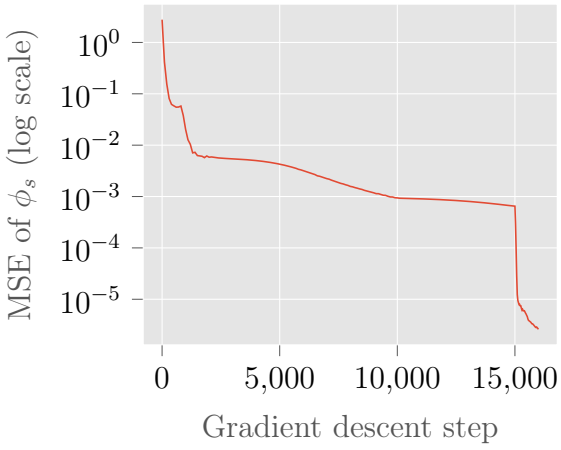



(d) MSE of temperature ($T$)

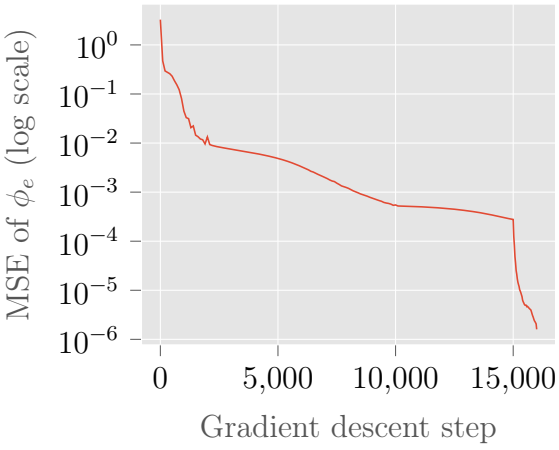



(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

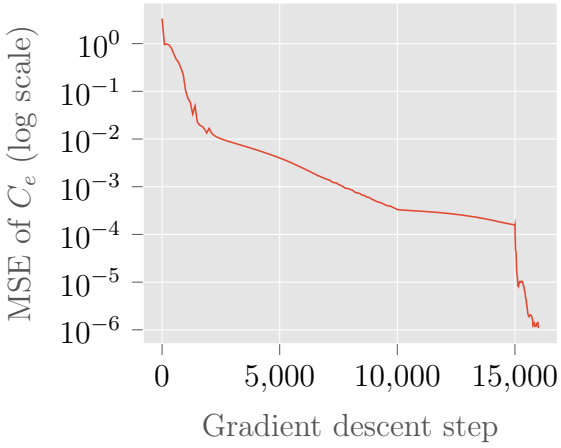
Figure 23: MSEs of each of the variables of the model for the XPINN approach between times 0 s and 400 s, with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7), with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.



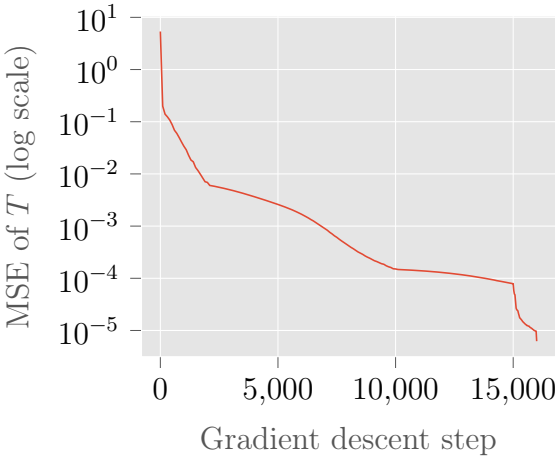

(a) MSE of solid potential ($\phi_s$)



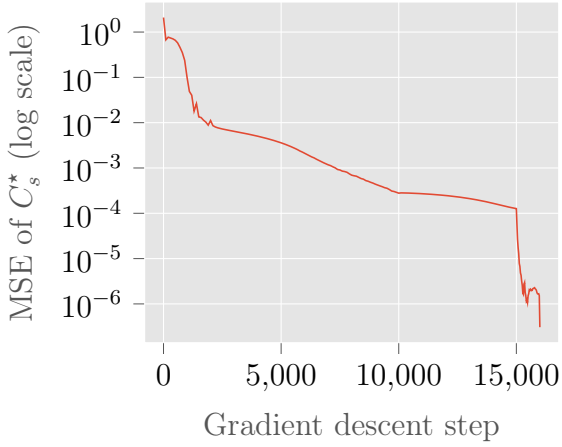

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

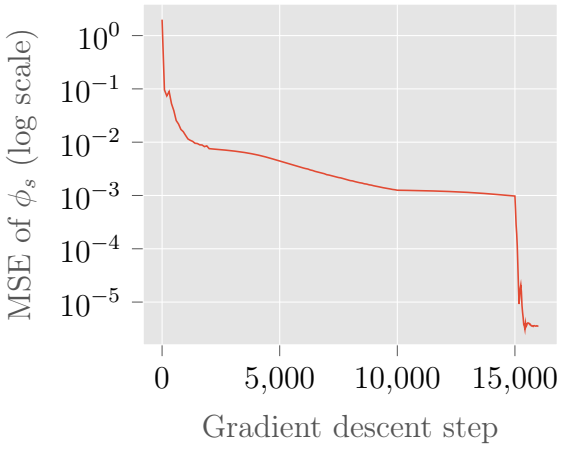



(d) MSE of temperature ($T$)

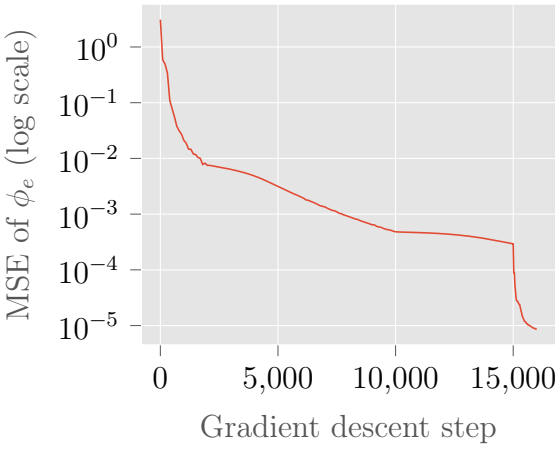



(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

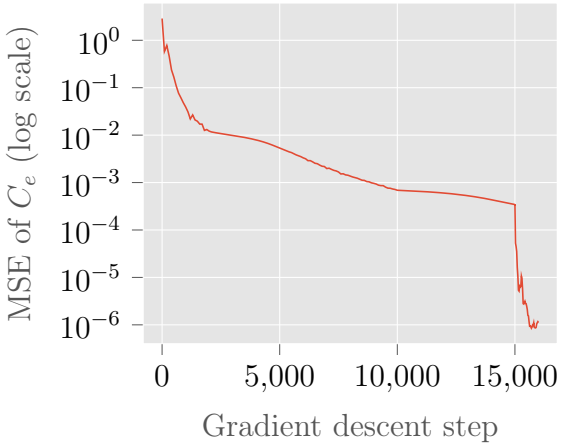
Figure 24: MSEs of each of the variables of the model for the XPINN approach between times 400 s and 800 s, with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7), with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.



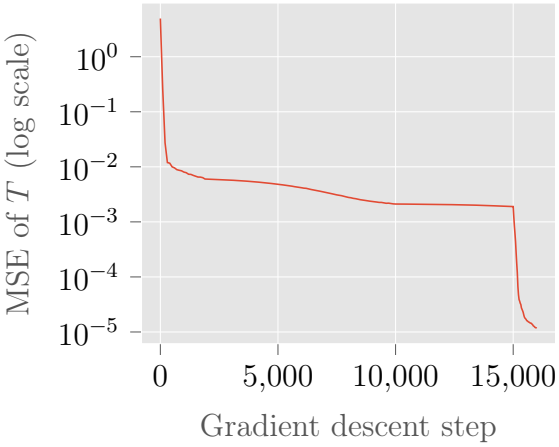

(a) MSE of solid potential ($\phi_s$)



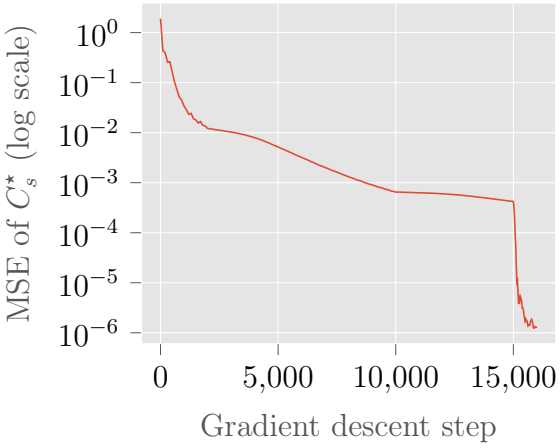

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)




(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

Figure 25: MSEs of each of the variables of the model for the XPINN approach between times 800 s and 1200 s, with boundary conditions data from LIONSIMBA and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7), with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.

## 4.4. Review of the approximation of the full P2D model in short timescale

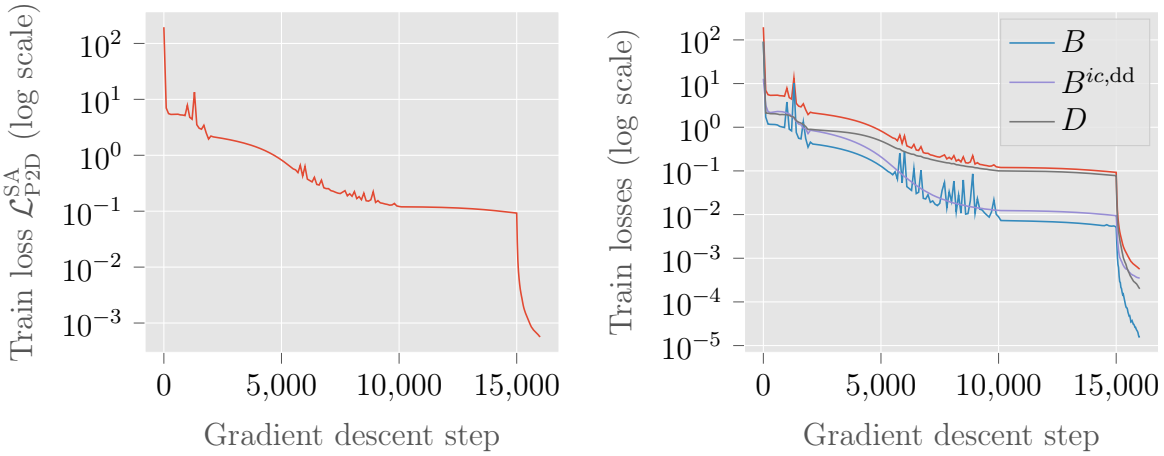
In this section, we will analyze the results obtained by applying a PINNs method to the entire P2D model, *i.e.* with the $D$ operator from the equations, as well as the $B$ operator from the boundary conditions as defined in Equation (4.5) and Equation (4.8) of Section 4.1.1, respectively, keeping as data only the initial conditions represented by the $B_i^{ic,\mathrm{dd}}$ operators, for all $i \in \{a, p, s, n, z\}$, defined in Equation (4.6) of Section 4.2.5.

We will train the neural network defined in Equation (4.28) with a self-adaptive mechanism as in experiments of Section 4.3, yielding a loss built in exactly the same way as the one defined in Equation (4.40), up to replacement of operators $\hat{B}_i^{bc,\mathrm{dd}}$ and $B_i^{bc,\mathrm{dd}}$ by operator $B$ described above.

We will evaluate our results by plotting the training loss as in Figure 39 with a second plot showing the contribution made by each component of the loss, namely the PDEs (operator $D$), the boundary conditions (operator $B$), and initial conditions (operators $B_i^{ic,\mathrm{dd}}$). We will then plot the MSEs of the different variables as in Figure 40, and the PDEs MSEs as in Figure 14.

We will review the results obtained for $T_{\max} = 1\,\mathrm{s}$ with $I_{\mathrm{app}} = 40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7). The training will be performed with 15000 steps of the Adam algorithm, initialized with $(\delta, \beta, \gamma) = (10^{-3}, 0.9, 0.999)$ (*cf.* Definition 3.10), followed by 1000 steps of L-BFGS algorithm. We also apply a scheduling heuristic consisting in decaying $\delta$ to $10^{-4}$ from step 2000 and to $10^{-5}$ from step 10000 during the Adam training.

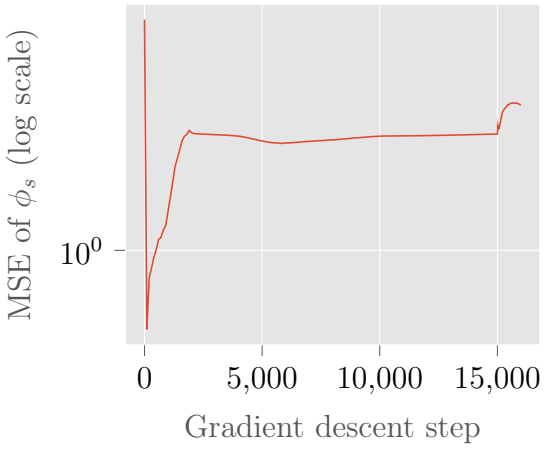
As shown in Figure 26, the learning seems to have been successful. Nevertheless, we see in Figure 27 below that the results are highly contrasted, since the MSEs are accurate for $C_e$, $\phi_e$ and $C_s^\star$ only, with $T$ and $\phi_s$ not learned at all. Given the very short timescale over which the model is being evaluated, this would seem to show that the PINN method applied as such to the P2D model, without any data on which to base it, is not viable as such, and requires adaptation of either the method or the model under consideration.



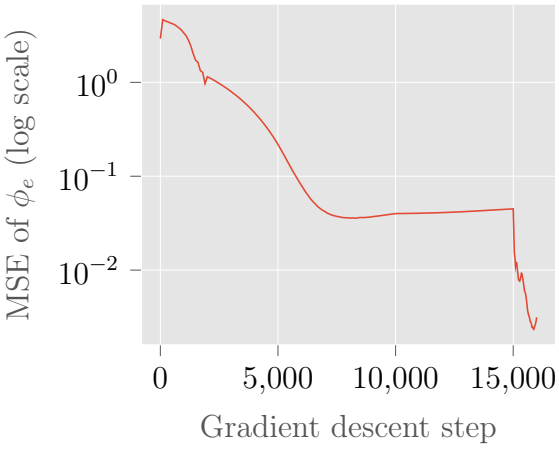

(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

(b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

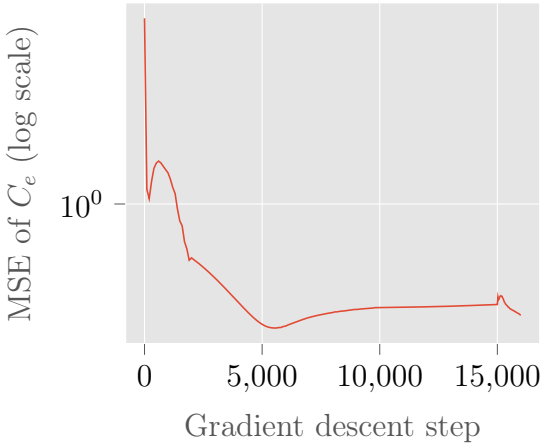
Figure 26: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the PINN approach with boundary conditions learned from the P2D model equations and self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the learning process succeeded, the loss losing 5 orders of magnitude during training, indicating that the neural network did learn both the boundary conditions data, and the losses from the P2D model PDEs and boundary conditions.
We also find that each loss component contributes exactly the same orders of magnitude to the total loss, meaning that each component has been learned equally.
Finally, we note the effect of scheduling, with clear inflections of the loss slope at the 2000th and at the 10000th step, and of the change from Adam to L-BFGS at the 15000th step, with an even clearer change in the loss slope.



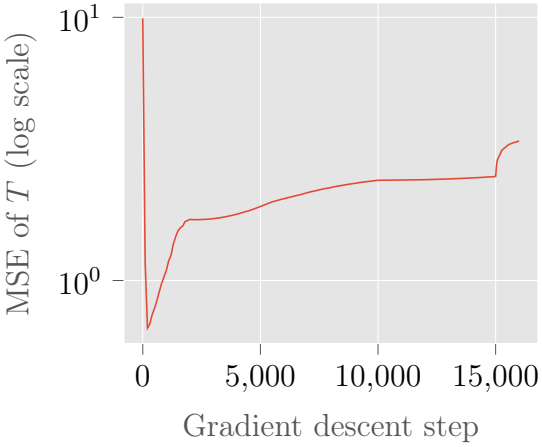

(a) MSE of solid potential ($\phi_s$)



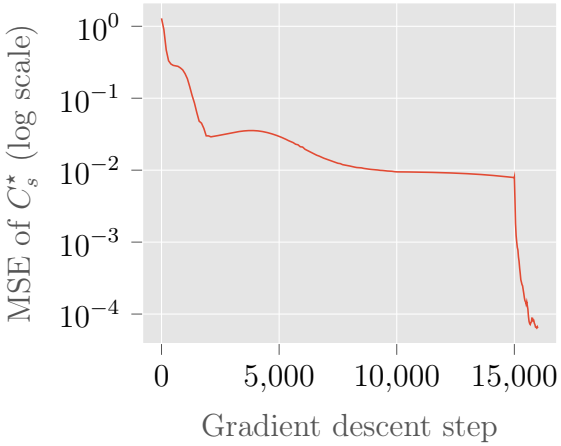

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)




(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

Figure 27: MSEs of each of the variables of the model for the PINN approach with boundary conditions learned from the P2D model equations and self-adaptive mechanism, with $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We observe that $C_s^\star$ is accurately learned by the neural networks, its MSE losing 4 orders of magnitude during the training process, while $\phi_e$ is learned up to 2 orders of magnitude and $C_e$ up to one orders of magnitude. On the other hand, we observe that the learning of $T$ and $\phi_s$ completely fails.

# 5. Conclusion and perspectives

Our work has shown that using PINNs to solve the P2D problem is a valid choice and opens up a very promising avenue, in that it allows very accurate results to be obtained, with a minimal fraction of data, as seen in Sections 4.3.2 and 4.3.3.

However, we have also shown that an approach based entirely based on PINNs with no data to fall back on is not functional as such, as shownin Section 4.4. More research is then needed in order to find other methods, or to find a reformulation of the initial problem that can overcome this difficulty.

We also believe that further theoretical work on PINNs is needed to understand their learning dynamics and to be able to provide explicit convergence conditions and rates, to complement existing results on explicit error estimates (*e.g. cf.* De Ryck et al. 2023; Mishra and Molinaro 2020; De Ryck and Mishra 2021; Shin et al. 2020). We hope that the theoretical framework, formulated in the language of functional analysis, in which we have set out to present this work, will provide a fertile basis for future theoretical work, notably by exploiting and amplifying the tools associated with the Neural Tangent Kernel introduced by Jacot et al. (2018).

Finally, we are firmly convinced that an important path is the use of PINNs in industrial applications, which in itself raises important research questions, since we need to succeed in incorporating the data produced in real time into the learning process, without causing computational costs to explode.

On a more personal note, I am really looking forward to exploring some of these avenues of research in my doctoral work, which starts right now.

# References


Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard (2016): "{TensorFlow}: A System for {Large-Scale} Machine Learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283.

Albawi, S., T. A. Mohammed, and S. Al-Zawi (2017): "Understanding of a Convolutional Neural Network," in *2017 International Conference on Engineering and Technology (ICET)*, Ieee, 1–6.

Amari, S. (1967): "A Theory of Adaptive Pattern Classifiers," *IEEE Transactions on Electronic Computers*, 299–307.

Arora, S., N. Cohen, N. Golowich, and W. Hu (2018): "A Convergence Analysis of Gradient Descent for Deep Linear Neural Networks," *arXiv preprint arXiv:1810.02281*.

Bach, F. and L. Chizat (2021): "Gradient Descent on Infinitely Wide Neural Networks: Global Convergence and Generalization," *arXiv preprint arXiv:2110.08084*.

Baydin, A. G., B. A. Pearlmutter, A. A. Radul, and J. M. Siskind (2018): "Automatic Differentiation in Machine Learning: A Survey," *Journal of Marchine Learning Research*, 18, 1–43.

Berner, J., P. Grohs, G. Kutyniok, and P. Petersen (2021): "The Modern Mathematics of Deep Learning," *arXiv preprint arXiv:2105.04026*, 86–114.

Beyer, H.-G. (2001): *The Theory of Evolution Strategies*, Springer Science & Business Media.

Boursier, E., L. Pillaud-Vivien, and N. Flammarion (2022): "Gradient Flow Dynamics of Shallow ReLU Networks for Square Loss and Orthogonal Inputs," .

Bruggeman, D. a. G. (1935): "Berechnung verschiedener physikalischer Konstanten von heterogenen Substanzen. I. Dielektrizitätskonstanten und Leitfähigkeiten der Mischkörper aus isotropen Substanzen," *Annalen der Physik*, 416, 636–664.

Butcher, J. C. (2016): *Numerical Methods for Ordinary Differential Equations*, John Wiley & Sons.

Chang, L. and J. Xiaoluo (2011): "Kalman Filter Based on SVM Innovation Update for Predicting State-of Health of VRLA Batteries," in *Applied Informatics and Communication*, ed. by D. Zeng, Berlin, Heidelberg: Springer, Communications in Computer and Information Science, 455–463.

Chen, C., Z. Wei, and A. C. Knoll (2021): "Charging Optimization for Li-ion Battery in Electric Vehicles: A Review," *IEEE Transactions on Transportation Electrification*, 8, 3068–3089.

Chizat, L. and F. Bach (2018): "On the Global Convergence of Gradient Descent for Over-Parameterized Models Using Optimal Transport," *Advances in neural information processing systems*, 31.

Chizat, L., E. Oyallon, and F. Bach (2019): "On Lazy Training in Differentiable Programming," *Advances in neural information processing systems*, 32.

Choy, T. C. (2015): *Effective Medium Theory: Principles and Applications*, Oxford University Press.

Cooper, Y. (2018): "The Loss Landscape of Overparameterized Neural Networks," *arXiv preprint arXiv:1804.10200*.

Corliss, G. F. (1988): "Applications of Differentiation Arithmetic," in *Reliability in Computing*, Elsevier, 127–148.

Cuomo, S., V. S. di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli (2022): "Scientific Machine Learning through Physics-Informed Neural Networks: Where We Are and What's Next," .

Davis, P. J. and P. Rabinowitz (2007): *Methods of Numerical Integration*, Courier Corporation.

De Ryck, T., A. D. Jagtap, and S. Mishra (2023): "Error Estimates for Physics Informed Neural Networks Approximating the Navier-Stokes Equations," .

De Ryck, T., S. Lanthaler, and S. Mishra (2021): "On the Approximation of Functions by Tanh Neural Networks," *Neural Networks*, 143, 732–750.

De Ryck, T. and S. Mishra (2021): "Error Analysis for Physics Informed Neural Networks (PINNs) Approximating Kolmogorov PDEs," .

Dickinson, E. J. F. and A. J. Wain (2020): "The Butler-Volmer Equation in Electrochemical Theory: Origins, Value, and Practical Application," *Journal of Electroanalytical Chemistry*, 872, 114145.

Doyle, M., T. F. Fuller, and J. Newman (1993): "Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell," *Journal of the Electrochemical society*, 140, 1526.

Dubey, S. R., S. K. Singh, and B. B. Chaudhuri (2022): "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark," *Neurocomputing*, 503, 92–108.

Elbrächter, D., D. Perekrestenko, P. Grohs, and H. Bölcskei (2021): "Deep Neural Network Approximation Theory," .

Emmerich, M., O. M. Shir, and H. Wang (2018): "Evolution Strategies," in *Handbook of Heuristics*, ed. by R. Martí, P. Panos, and M. G. C. Resende, Cham: Springer International Publishing, 1–31.

Evans, L. (2018): *Measure Theory and Fine Properties of Functions*, Routledge.

Evans, L. C. (2010): *Partial Differential Equations*, American Mathematical Soc.

Feragen, A. and T. Nye (2020): "Statistics on Stratified Spaces," in *Riemannian Geometric Statistics in Medical Image Analysis*, Elsevier, 299–342.

Fletcher, R. (2000): *Practical Methods of Optimization*, John Wiley & Sons.

Gilbarg, D., N. S. Trudinger, D. Gilbarg, and N. S. Trudinger (1977): *Elliptic Partial Differential Equations of Second Order*, vol. 224, Springer.

Glorot, X. and Y. Bengio (2010): "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Goldberg, Y. (2022): *Neural Network Methods for Natural Language Processing*, Springer Nature.

Goodfellow, I., Y. Bengio, and A. Courville (2016): *Deep Learning*, MIT press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014): "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, 2672–2680.

Griewank, A. and A. Walther (2008): *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM.

Gu, W. B. and C. Y. Wang (2000): "Thermal-Electrochemical Modeling of Battery Systems," *Journal of The Electrochemical Society*, 147, 2910.

Han, R., C. Macdonald, and B. Wetton (2021): "A Fast Solver for the Pseudo-Two-Dimensional Model of Lithium-Ion Batteries," *arXiv preprint arXiv:2111.09251*.

Hansen, N. (2016): "The CMA Evolution Strategy: A Tutorial," *arXiv preprint arXiv:1604.00772*.

Harris, C. R., K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, and N. J. Smith (2020): "Array Programming with NumPy," *Nature*, 585, 357–362.

Hu, X., S. Li, and H. Peng (2012): "A Comparative Study of Equivalent Circuit Models for Li-ion Batteries," *Journal of Power Sources*, 198, 359–367.

Hunter, J. D. (2007): "Matplotlib: A 2D Graphics Environment," *Computing in science & engineering*, 9, 90–95.

Imry, Y. (2002): *Introduction to Mesoscopic Physics*, Oxford University Press.

Jacot, A., F. Gabriel, and C. Hongler (2018): "Neural Tangent Kernel: Convergence and Generalization in Neural Networks," *Advances in neural information processing systems*, 31.

Jacot, A., F. Ged, B. Şimşek, C. Hongler, and F. Gabriel (2022): "Saddle-to-Saddle Dynamics in Deep Linear Networks: Small Initialization Training, Symmetry, and Sparsity," .

Jagtap, A. D. and G. E. Karniadakis (2021): "Extended Physics-informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations." in *AAAI Spring Symposium: MLPS*, 2002–2041.

Jakubovitz, D., R. Giryes, and M. R. Rodrigues (2019): "Generalization Error in Deep Learning," in *Compressed Sensing and Its Applications: Third International MATHEON Conference 2017*, Springer, 153–193.

Jerrell, M. E. (1997): "Automatic Differentiation and Interval Arithmetic for Estimation of Disequilibrium Models," *Computational Economics*, 10, 295–316.

Jiang, S. and Z. Song (2022): "A Review on the State of Health Estimation Methods of Lead-Acid Batteries," *Journal of Power Sources*, 517, 230710.

Karniadakis, G. E., I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang (2021): "Physics-Informed Machine Learning," *Nature Reviews Physics*, 3, 422–440.

Kim, C., S. Kim, J. Kim, D. Lee, and S. Kim (2021): "Automated Learning Rate Scheduler for Large-batch Training," .

Kingma, D. P. and J. Ba (2014): "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*.

Kumar, S. K. (2017): "On Weight Initialization in Deep Neural Networks," *arXiv preprint arXiv:1704.08863*.

Kumaresan, K., G. Sikha, and R. E. White (2007): "Thermal Model for a Li-Ion Cell," *Journal of The Electrochemical Society*, 155, A164.

Kunkel, P. (2006): *Differential-Algebraic Equations: Analysis and Numerical Solution*, vol. 2, European Mathematical Society.

Lagaris, I. E., A. Likas, and D. I. Fotiadis (1998): "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *IEEE transactions on neural networks*, 9, 987–1000.

LeCun, Y., Y. Bengio, and G. Hinton (2015): "Deep Learning," *nature*, 521, 436–444.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998): "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 86, 2278–2324.

Lemieux, C. (2009): *Monte Carlo and Quasi-Monte Carlo Sampling*, Springer Science & Business Media.

Leshno, M., V. Y. Lin, A. Pinkus, and S. Schocken (1993): "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function," *Neural networks*, 6, 861–867.

Lewkowycz, A. (2021): "How to Decay Your Learning Rate," .

Li, Z. and S. Arora (2019): "An Exponential Learning Rate Schedule for Deep Learning," .

Linnainmaa, S. (1976): "Taylor Expansion of the Accumulated Rounding Error," *BIT Numerical Mathematics*, 16, 146–160.

Liu, D. C. and J. Nocedal (1989): "On the Limited Memory BFGS Method for Large Scale Optimization," *Mathematical Programming*, 45, 503–528.

Liu, Z. L. (2018): "Finite Volume Method," in *Multiphysics in Porous Materials*, ed. by Z. L. Liu, Cham: Springer International Publishing, 385–395.

Lu, L., X. Meng, Z. Mao, and G. E. Karniadakis (2021): "DeepXDE: A Deep Learning Library for Solving Differential Equations," *SIAM Review*, 63, 208–228.

McClenny, L. and U. Braga-Neto (2022): "Self-Adaptive Physics-Informed Neural Networks Using a Soft Attention Mechanism," .

McKay, M. D., R. J. Beckman, and W. J. Conover (2000): "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, 42, 55–61.

McKinney, W. (2010): "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, vol. 445, 51–56.

Methekar, R. (2018): "SOC Estimation with Thermal and Charging Rate Consideration Using Dual Filter Approach for Lithium-Ion Battery," *Journal of Renewable and Sustainable Energy*, 10, 064103.

Mishra, S. and R. Molinaro (2020): "Estimates on the Generalization Error of Physics Informed Neural Networks (PINNs) for Approximating PDEs," *arXiv:2006.16144 [cs, math]*.

Narkhede, M. V., P. P. Bartakke, and M. S. Sutaone (2022): "A Review on Weight Initialization Strategies for Neural Networks," *Artificial intelligence review*, 55, 291–322.

Neal, R. M. (1996): "Priors for Infinite Networks," in *Bayesian Learning for Neural Networks*, Springer, 29–53.

Nesterov, Y. E. (1983): "A Method of Solving a Convex Programming Problem with Convergence Rate O\backslashbigl(K$\hat{2}$\backslashbigr)," in *Doklady Akademii Nauk*, Russian Academy of Sciences, vol. 269, 543–547.

Newman, J. and N. P. Balsara (2021): *Electrochemical Systems*, John Wiley & Sons.

Newman, J. and W. Tiedemann (1975): "Porous-Electrode Theory with Battery Applications," *AIChE Journal*, 21, 25–41.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019): "PyTorch: An Imperative Style, High-Performance Deep Learning Library," .

Qian, N. (1999): "On the Momentum Term in Gradient Descent Learning Algorithms," *Neural networks*, 12, 145–151.

Rahaman, N., A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville (2019): "On the Spectral Bias of Neural Networks," in *International Conference on Machine Learning*, PMLR, 5301–5310.

Raissi, M., P. Perdikaris, and G. Karniadakis (2019): "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," *Journal of Computational Physics*, 378, 686–707.

Ramkumar, M. S., C. S. R. Reddy, A. Ramakrishnan, K. Raja, S. Pushpa, S. Jose, and M. Jayakumar (2022): "Review on Li-Ion Battery with Battery Management System in Electrical Vehicle," *Advances in Materials Science and Engineering*, 2022, e3379574.

Reddy, T. (2010): *Linden's Handbook of Batteries, 4th Edition*, McGraw Hill Professional.

Rosenblatt, F. (1958): "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological review*, 65, 386.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1985): "Learning Internal Representations by Error Propagation," Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science.

Santambrogio, F. (2017): "{Euclidean, Metric, and Wasserstein} Gradient Flows: An Overview," *Bulletin of Mathematical Sciences*, 7, 87–154.

Sarvi, M. and M. A. Masoum (2008): "A Neural Network Model for Ni-Cd Batteries," in *2008 43rd International Universities Power Engineering Conference*, IEEE, 1–5.

Schwab, C. and J. Zech (2021): "Deep Learning in High Dimension: Neural Network Approximation of Analytic Functions in $L\hat{2}(\mathbb{R}\hat{d},\gamma_d)$," *arXiv:2111.07080 [cs, math, stat]*.

See, K. W., G. Wang, Y. Zhang, Y. Wang, L. Meng, X. Gu, N. Zhang, K. C. Lim, L. Zhao, and B. Xie (2022): "Critical Review and Functional Safety of a Battery Management System for Large-Scale Lithium-Ion Battery Pack Technologies," *International Journal of Coal Science & Technology*, 9, 36.

Shin, Y., J. Darbon, and G. E. Karniadakis (2020): "On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs," *Communications in Computational Physics*, 28, 2042–2074.

Simsek, B., F. Ged, A. Jacot, F. Spadaro, C. Hongler, W. Gerstner, and J. Brea (2021): "Geometry of the Loss Landscape in Overparameterized Neural Networks: Symmetries and Invariances," in *International Conference on Machine Learning*, PMLR, 9722–9732.

Singh, P. and D. Reisner (2002): "Fuzzy Logic-Based State-of-Health Determination of Lead Acid Batteries," in *24th Annual International Telecommunications Energy Conference*, 583–590.

Sipser, M. (2012): *Introduction to the Theory of Computation*, Cengage Learning.

Sola, J. and J. Sevilla (1997): "Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems," *IEEE Transactions on nuclear science*, 44, 1464–1468.

Subramanian, V. R., V. D. Diwakar, and D. Tapriyal (2005): "Efficient Macro-Micro Scale Coupled Modeling of Batteries," *Journal of The Electrochemical Society*, 152, A2002.

Summerfield, J. H. and C. N. Curtis (2015): "Modeling the Lithium Ion/Electrode Battery Interface Using Fick's Second Law of Diffusion, the Laplace Transform, Charge Transfer Functions, and a [4, 4] Padé Approximant," *International Journal of Electrochemistry*, 2015, e496905.

Sun, R., D. Li, S. Liang, T. Ding, and R. Srikant (2020): "The Global Landscape of Neural Networks: An Overview," *IEEE Signal Processing Magazine*, 37, 95–108.

Torchio, M., L. Magni, R. B. Gopaluni, R. D. Braatz, and D. M. Raimondo (2016): "LIONSIMBA: A Matlab Framework Based on a Finite Volume Model Suitable for Li-Ion Battery Design, Simulation, and Control," *Journal of The Electrochemical Society*, 163, A1192.

Torchio, M., N. A. Wolff, D. M. Raimondo, L. Magni, U. Krewer, R. B. Gopaluni, J. A. Paulson, and R. D. Braatz (2015): "Real-Time Model Predictive Control for the Optimal Charging of a Lithium-Ion Battery," in *2015 American Control Conference (ACC)*, IEEE, 4536–4541.

Urbain, M. (2009): "Modélisation Électrique et Énergétique Des Accumulateurs Li-Ion. Estimation En Ligne de La SOC et de La SOH," Ph.D. thesis, Institut National Polytechnique de Lorraine.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017): "Attention Is All You Need," *arXiv:1706.03762 [cs]*.

Vetter, J., P. Novák, M. R. Wagner, C. Veit, K.-C. Möller, J. O. Besenhard, M. Winter, M. Wohlfahrt-Mehrens, C. Vogler, and A. Hammouche (2005): "Ageing Mechanisms in Lithium-Ion Batteries," *Journal of power sources*, 147, 269–281.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, and J. Bright (2020): "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature methods*, 17, 261–272.

Voulodimos, A., N. Doulamis, A. Doulamis, and E. Protopapadakis (2018): "Deep Learning for Computer Vision: A Brief Review," *Computational intelligence and neuroscience*, 2018.

Wang, Q., Y. Ma, K. Zhao, and Y. Tian (2022a): "A Comprehensive Survey of Loss Functions in Machine Learning," *Annals of Data Science*, 9, 187–212.

Wang, S., P. Ren, P. Takyi-Aninakwa, S. Jin, and C. Fernandez (2022b): "A Critical Review of Improved Deep Convolutional Neural Network for Multi-Timescale State Prediction of Lithium-Ion Batteries," *Energies*, 15, 5053.

Wang, S., X. Yu, and P. Perdikaris (2022c): "When and Why PINNs Fail to Train: A Neural Tangent Kernel Perspective," *Journal of Computational Physics*, 449, 110768.

Werbos, P. J. (1982): "Applications of Advances in Nonlinear Sensitivity Analysis," in *System Modeling and Optimization*, ed. by R. F. Drenick and F. Kozin, Berlin/Heidelberg: Springer-Verlag, vol. 38, 762–770.

Williams, C. (1996): "Computing with Infinite Networks," *Advances in neural information processing systems*, 9.

Yang, G. (2019): "Scaling Limits of Wide Neural Networks with Weight Sharing: Gaussian Process Behavior, Gradient Independence, and Neural Tangent Kernel Derivation," *arXiv preprint arXiv:1902.04760*.

Zhang, C. P., J. Z. Liu, and S. M. Sharkh (2009): "Identification of Dynamic Model Parameters for Lithium-Ion Batteries Used in Hybrid Electric Vehicles," .

Zhang, D., B. N. Popov, and R. E. White (2000): "Modeling Lithium Intercalation of a Single Spinel Particle under Potentiodynamic Control," *Journal of the Electrochemical Society*, 147, 831.

Zou, C., X. Hu, Z. Wei, T. Wik, and B. Egardt (2017): "Electrochemical Estimation and Control for Lithium-Ion Battery Health-Aware Fast Charging," *IEEE Transactions on Industrial Electronics*, 65, 6635–6645.

# A. Summary of the equations

Current collectors $i \in \{a, z\}$ — Boundary conditions

$$\rho_i C_{p,i} \partial_t T(x,t) = \lambda_i \partial_{x^2} T(x,t) + \frac{I^2_{\text{app}}(t)}{\sigma_{\text{eff},i}} \quad (2.3a)$$

$$(2.3b) \quad -\lambda_a \partial_t T(x,t)\Big|_{x=x_0^+} = h[T_{\text{ref}} - T(x,t)]$$

$$(2.3c) \quad -\lambda_z \partial_t T(x,t)\Big|_{x=x_z^-} = h[T(x,t) - T_{\text{ref}}]$$

Electrodes $i \in \{p, n\}$

$$\rho_i C_{p,i} \partial_t T(x,t) = \lambda_i \partial_{x^2} T(x,t) + Q_{\text{ohm},i} + Q_{\text{rxn},i} + Q_{\text{rev},i} \quad (2.4a)$$

$$(2.4b) \quad -\lambda_a \partial_x T(x,t)\Big|_{x=x_a^-} = -\lambda_p \partial_x T(x,t)\Big|_{x=x_a^+}$$

$$(2.4c) \quad -\lambda_n \partial_x T(x,t)\Big|_{x=x_n^-} = -\lambda_z \partial_x T(x,t)\Big|_{x=x_n^+}$$

$$\epsilon_i \partial_t C_e(x,t) = \partial_x \left[D_{\text{eff},i} \partial_x C_e(x,t)\right] + a_i(1 - t_+) j_i \quad (2.6a)$$

$$(2.6b) \quad \partial_x C_e(x,t)\Big|_{x=x_a^+, x_n^-} = 0$$

$$a_i F j_i(x,t) = \partial_x \left(\kappa_{\text{eff},i} \left[\Upsilon T(x,t) \frac{\partial_x C_e(x,t)}{C_e(x,t)} - \partial_x \phi_e(x,t)\right]\right) \quad (2.8a)$$

$$(2.8b) \quad \partial_x \phi_e(x,t)\Big|_{x=x_a^+} = 0$$

$$(2.8c) \quad \phi_e(x,t)\Big|_{x=x_n^-} = 0$$

$$\partial_t C_s^*(x,t) = -3 \frac{j_i}{R_{p,i}} - \frac{R_{p,i}}{5} \frac{\partial_t j_i D^s_{\text{eff},i} - \partial_t D^s_{\text{eff},i} j_i}{\left(D^s_{\text{eff},i}\right)^2} \quad (2.12c)$$

$$a_i F j_i(x,t) = \sigma_{\text{eff},i} \partial_{x^2} \phi_s(x,t) \quad (2.13a)$$

$$(2.13b) \quad \sigma_{\text{eff},i} \partial_x \phi_s(x,t)\Big|_{x=x_p^-, x_s^+} = 0$$

$$(2.13c) \quad \sigma_{\text{eff},i} \partial_x \phi_s(x,t)\Big|_{x=x_a^+, x_n^-} = -I_{\text{app}}(t)$$

Separator

$$\rho_s C_{p,s} \partial_t T(x,t) = \lambda_i \partial_{x^2} T(x,t) + Q_{\text{ohm},s} \quad (2.5a)$$

$$(2.5b) \quad -\lambda_p \partial_x T(x,t)\Big|_{x=x_p^-} = -\lambda_s \partial_x T(x,t)\Big|_{x=x_p^+}$$

$$(2.5c) \quad -\lambda_s \partial_x T(x,t)\Big|_{x=x_s^-} = -\lambda_n \partial_x T(x,t)\Big|_{x=x_s^+}$$

$$\epsilon_s \partial_t C_e(x,t) = \partial x \left[D_{\text{eff},s} \partial_x C_e(x,t)\right] \quad (2.7a)$$

$$(2.7b) \quad -D_{\text{eff},p} \partial_x C_e(x,t)\Big|_{x=x_p^-} = -D_{\text{eff},s} \partial_x C_e(x,t)\Big|_{x=x_p^+}$$

$$(2.7c) \quad -D_{\text{eff},s} \partial_x C_e(x,t)\Big|_{x=x_s^-} = -D_{\text{eff},n} \partial_x C_e(x,t)\Big|_{x=x_s^+}$$

$$0 = \partial_x \left(\kappa_{\text{eff},s} \left[\Upsilon T(x,t) \frac{\partial_x C_e(x,t)}{C_e(x,t)} - \partial_x \phi_e(x,t)\right]\right) \quad (2.9a)$$

$$(2.9b) \quad -\kappa_{\text{eff},p} \partial_x \phi_e(x,t)\Big|_{x=x_p^-} = -\kappa_{\text{eff},s} \partial_x \phi_e(x,t)\Big|_{x=x_p^+}$$

$$(2.9c) \quad -\kappa_{\text{eff},s} \partial_x \phi_e(x,t)\Big|_{x=x_s^-} = -\kappa_{\text{eff},n} \partial_x \phi_e(x,t)\Big|_{x=x_s^+}$$

Table 14: Main equations of the P2D model (adapted from Torchio et al. 2016)

Ionix flux $i \in \{p, n\}$

$$j_i = 2k_{\text{eff}} \sqrt{C_e(x,t) \left(C^{\max}_{s,i} - C_s(x,t,r)\big|_{r=R^-_{p,i}}\right) C_s(x,t,r)\big|_{r=R^-_{p,i}}} \sinh\left(\frac{0.5F}{RT(x,t)} \eta_i\right) \quad (2.14)$$

Surface overpotential $i \in \{p, n\}$

$$\eta_i = \phi_s(x,t) - \phi_e(x,t) - U_i \quad (2.15)$$

Open Circuit Potential $i \in \{p, n\}$

$$U_i = U_{\text{ref},i} + (T(x,t) - T_{\text{ref}}) \partial_T U_i|_{T_{\text{ref}}} \quad (2.16)$$

Entropy Change

$$\partial_T U_p|_{T_{\text{ref}}} = -0.001 \left(\frac{0.199521039 - 0.928373822\theta_p + 1.3645506890000030\theta_p^2 - 0.61154489399999980\theta_p^3}{1 - 5.6614798869999970\theta_p + 11.4763619100\theta_p^2 - 9.8243121359999980\theta_p^3 + 3.0467550630\theta_p^4}\right) \quad (2.17)$$

$$\partial_T U_n|_{T_{\text{ref}}} = 0.001 \left( \frac{\begin{array}{c} 0.005269056 + 3.299265709\theta_n - 91.79325798\theta_n^2 + 1004.911008\theta_n^3 - 5812.278127\theta_n^4 \\ + 19329.7549\theta_n^5 - 37147.8947\theta_n^6 + 38379.181278\theta_n^7 - 16515.053088\theta_n^8 \end{array}}{\begin{array}{c} 1 - 48.09287227\theta_n + 1017.234804\theta_n^2 - 10481.804198\theta_n^3 + 59431.3\theta_n^4 \\ - 195881.6488\theta_n^5 + 374577.3152\theta_n^6 - 385821.1607\theta_n^7 + 165705.8597\theta_n^8 \end{array}} \right) \tag{2.18}$$

$$\theta_i = \frac{C_s^*(x,t)}{C_{s,i}^{\max}} \qquad i \in \{p, n\} \tag{2.21}$$

Open circuit reference potential

$$U_{\text{p,ref}} = \frac{-4.656+88.669\theta_p^2-401.119\theta_p^4+342.909\theta_p^6-462.471\theta_p^8+433.434\theta_p^{10}}{-1+18.933\theta_p^2-79.532\theta_p^4+37.311\theta_p^6-73.083\theta_p^8+95.96\theta_p^{10}} \tag{2.19}$$

$$U_{\text{n,ref}} = 0.7222 + 0.1387\theta_n + 0.029\theta_n^{0.5} - \frac{0.0172}{\theta_n} + \frac{0.0019}{\theta^{1.5}} + 0.2808e^{0.9-15\theta_n} - 0.7984e^{0.4465\theta_n - 0.4108} \tag{2.20}$$

Heat source terms (electrodes) $\quad i \in \{p, n\}$

$$Q_{\text{ohm},i} = \sigma_{\text{eff},i} \left(\partial_x \phi_s(x,t)\right)^2 + \kappa_{\text{eff},i} \left(\partial_x \phi_e(x,t)\right)^2 + \frac{2\kappa_{\text{eff},i} RT(x,t)}{F}(1-t_+)\partial_x \ln C_e(x,t) \partial_x \phi_e(x,t) \tag{2.26}$$

$$Q_{\text{rxn},i} = Fa_i j(x,t)\eta_i(x,t) \tag{2.28}$$

$$Q_{\text{rev},i} = Fa_i j(x,t)T(x,t)\partial_T U_i|_{T_{\text{ref}}} \tag{2.29}$$

Heat source terms (separator)

$$Q_{\text{ohm},s} = \kappa_{\text{eff},s} \left(\partial_x \phi_e(x,t)\right)^2 + \frac{2\kappa_{\text{eff},s} RT(x,t)}{F}(1-t_+)\partial_x \ln C_e(x,t) \partial_x \phi_e(x,t) \tag{2.27}$$

Effective coefficients

$$D_{\text{eff},i} = \epsilon_i^{\text{brugg}_i} \times 10^{-4} \times 10^{-4.43 - \frac{54}{T(x,t)-229-5\times 10^{-3} C_e(x,t)} - 0.22\times 10^{-3} C_e(x,t)} \qquad i \in \{p, s, n\} \tag{2.22}$$

$$\begin{aligned} \kappa_{\text{eff},i} = \epsilon_i^{\text{brugg}_i} \times 10^{-4} \times C_e(x,t) \big(&-10.5 + 0.668\times 10^{-3} C_e(x,t) + 0.494\times 10^{-6} C_e(x,t)^2 + \\ &T(x,t)(0.074 - 1.78\times 10^{-5} C_e(x,t) - 8.86\times 10^{-10} C_e(x,t)^2) + \\ &T(x,t)^2(-6.96\times 10^{-5} + 2.8\times 10^{-8} C_e(x,t))\big)^2 \end{aligned} \qquad i \in \{p, s, n\} \tag{2.24}$$

$$k_{\text{eff}} = k_i e^{\frac{E_{a,i}^k}{R}\left(\frac{1}{T(x,t)} - \frac{1}{T_{\text{ref}}}\right)} \qquad i \in \{p, n\} \tag{2.25}$$

$$D_{\text{eff},i}^s = D_i^s e^{\frac{E_{a,i}^{D^s}}{R}\left(\frac{1}{T(x,t)} - \frac{1}{T_{\text{ref}}}\right)} \qquad i \in \{p, n\} \tag{2.23}$$

$$\sigma_{\text{eff},i} = \sigma_i(1 - \epsilon_i - \epsilon_{f,i}) \qquad i \in \{p, n\} \tag{2.2}$$

Other coefficients

$$\Upsilon = \frac{2(1-t_+)R}{F} \tag{2.1}$$

Table 15: Secondary variables of the P2D model (adapted from Han et al. 2021)

# B. Most common activation functions

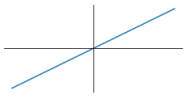
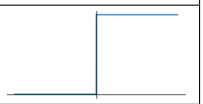
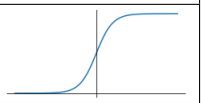
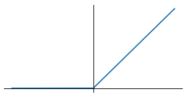
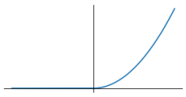
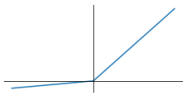
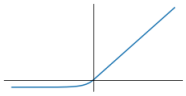
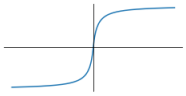
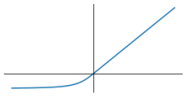
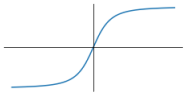
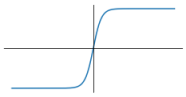
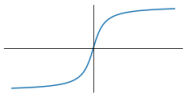
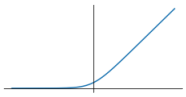
| Name | Given as a function of $x \in \mathbb{R}$ by | Plot |
|---|---|---|
| linear | $x$ |  |
| Heaviside / step function | $\mathbb{1}_{(0,\infty)}(x)$ |  |
| logistic function / sigmoid | $\frac{1}{1+e^{-x}}$ |  |
| rectified linear unit (ReLU) | $\max\{0, x\}$ |  |
| power rectified linear unit | $\max\{0, x\}^k$ for $k \in \mathbb{N}$ |  |
| parametric ReLU (PReLU) | $\max\{ax, x\}$ for $a \geqslant 0$, $a \neq 1$ |  |
| exponential linear unit (ELU) | $x \cdot \mathbb{1}_{[0,\infty)}(x) + (e^x - 1) \cdot \mathbb{1}_{(-\infty,0)}(x)$ |  |
| softsign | $\frac{x}{1+\lvert x\rvert}$ |  |
| inverse square root linear unit | $x \cdot \mathbb{1}_{[0,\infty)}(x) + \frac{x}{\sqrt{1+ax^2}} \cdot \mathbb{1}_{(-\infty,0)}(x)$ for $a > 0$ |  |
| inverse square root unit | $\frac{x}{\sqrt{1+ax^2}}$ for $a > 0$ |  |
| tanh | $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ |  |
| arctan | $\arctan(x)$ |  |
| SoftPlus[19] | $\ln(1 + e^x)$ |  |
| Gaussian | $e^{-x^2/2}$ |  |

Table 16: List of commonly used activation functions (taken from Berner et al. 2021).

[19] SoftPlus is a smooth approximation of the ReLu, as well as a primitive function of the sigmoid.

# C. Illustration of the different differentiation methods

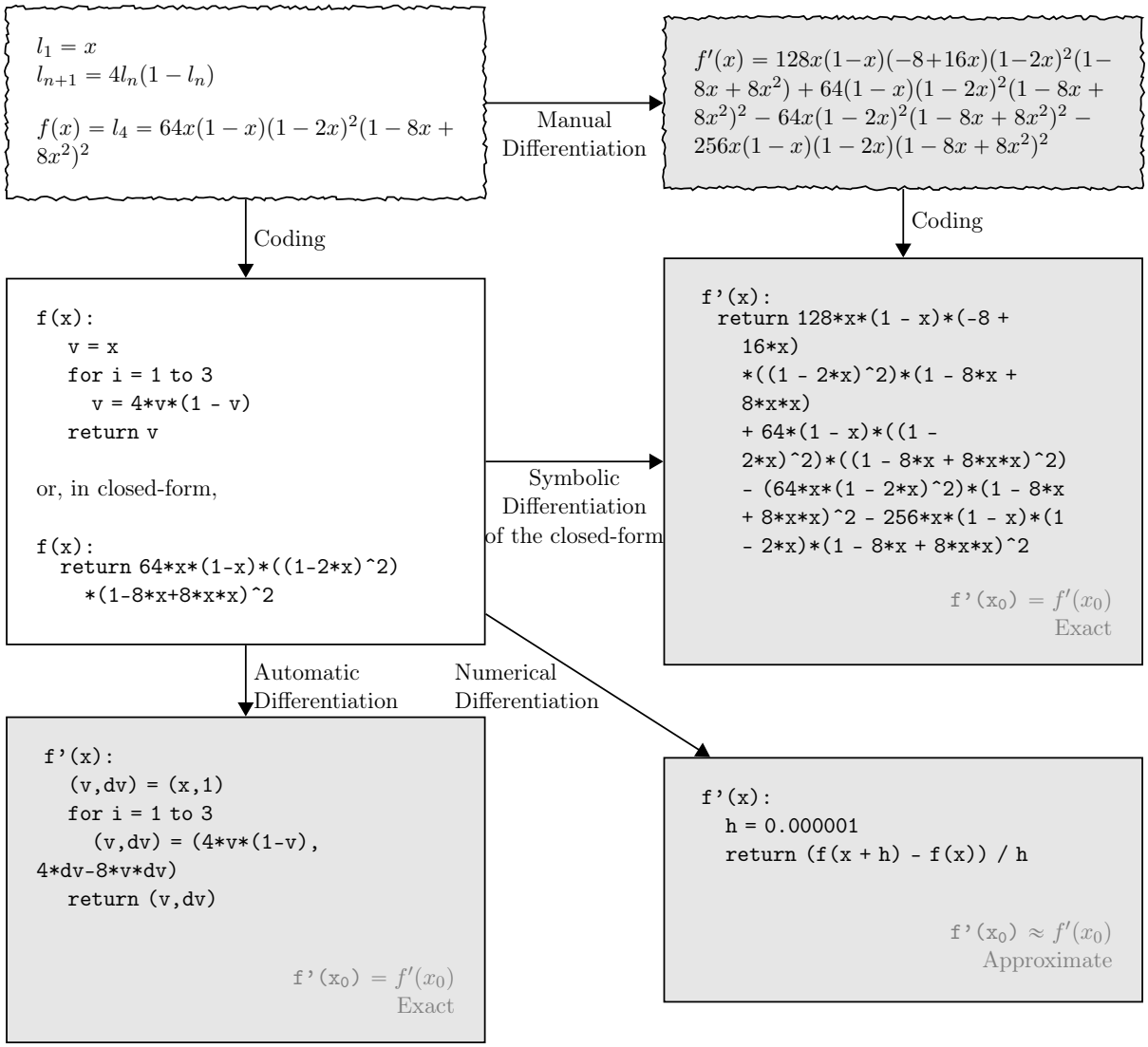



Figure 28: Illustration of the different differentiation methods on the example of a truncated logistic map (upper left) taken from Baydin et al. (2018).
Symbolic differentiation (center right) gives exact results but requires closed-form input and suffers from expression swell; numerical differentiation (lower right) has problems of accuracy due to round-off and truncation errors; automatic differentiation (lower left) for its part is as accurate as symbolic differentiation while being applicable to algorithmically computed functions (in particular making use of branching and looping operations).

# D. Introduction to the Neural Tangent Kernel (NTK)

This section presents part of the work developed in Jacot et al. (2018).

## D.1. Neural Tangent Kernel

Let be $\Psi : \mathbb{R}^P \to \mathcal{F}(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})$ a neural network architecture as in Definition 3.2. We then define the NTK:

**Definition D.1** (Neural Tangent Kernel (NTK))**.** Let be $\theta \in \mathbb{R}^P$. Then the Neural Tangent Kernel (NTK) of $\Psi$ at point $\theta$ is the function:

$$K_\theta : \left\{ \begin{array}{ccc} \mathbb{R}^{l_0} \times \mathbb{R}^{l_0} & \longrightarrow & \mathbb{R}^{l_d^2} \\ (x, y) & \longmapsto & \sum_{p=1}^P \partial_p \Psi(\theta)(x) \otimes \partial_p \Psi(\theta)(y) \end{array} \right. \tag{D.1}$$

## D.2. Training of an MLP with respect to the NTK

Let us recall the definition of the theoretical gradient flow:

**Definition D.2** (Theoretical gradient flow of an MLP)**.** Let fix $\theta_0 \in \mathbb{R}^P$, and a Loss $\mathcal{L} : \mathcal{F} \to \mathbb{R}$. An MLP follows the theoretical gradient flow if the function $\theta \in \mathcal{C}^1\left(\mathbb{R} \to \mathbb{R}^P\right)$ follows the ODE:

$$\begin{cases} \theta(0) & = \theta_0 \\ \frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) & = -\mathrm{d}\mathcal{L}|_{\Psi(\theta(t))}(\partial_p \Psi(\theta(t))), \quad \forall 1 \leqslant p \leqslant P, \forall t \in (0, +\infty) \end{cases}$$

Jacot et al. (2018) claim that this gradient flow is equivalent to the following flow, defined thanks to the NTK:

**Definition D.3** (Theoretical neural tangent kernel flow of an MLP)**.** Let fix $\theta_0 \in \mathbb{R}^P$, a Banach space $\mathcal{B}$, and a Loss $\mathcal{L} : \mathcal{B} \to \mathbb{R}$. An MLP follows the theoretical neural tangent kernel flow if the function $\theta : \mathbb{R} \to \mathbb{R}^P$ follows the ODE:

$$\begin{cases} \theta(0) & = \theta_0 \\ \partial_t \Psi(\theta) & = -\nabla_{K_\theta} \mathcal{L}|_{\Psi(\theta)} \end{cases} \tag{D.2}$$

where $\nabla_{K_\theta} \mathcal{L}|_{\Psi(\theta)} := y \in \mathbb{R}^{l_0} \mapsto \left(\mathrm{d}\mathcal{L}|_{\Psi(\theta)}(K_\theta(\cdot, y)_{\cdot,i})\right)_{1 \leqslant i \leqslant l_d}$

In the following, we will show that this equivalence happens only under some assumptions. Namely, we will prove the following proposition:

**Proposition D.1.** *Let be $\theta \in \mathcal{C}^1\left(\mathbb{R} \to \mathbb{R}^P\right)$ and let us fix $\theta_0 \in \mathbb{R}^P$ and a Loss $\mathcal{L} : \mathcal{B} \to \mathbb{R}$. Then:*

- *if $\theta$ follows the theoretical gradient flow of Equation (3.3), then $\theta$ also follows the theoretical neural tangent kernel flow of Equation (D.2).*
- *Conversely, if $\theta$ follows the theoretical neural tangent kernel flow of Equation (D.2) and if for all $t \in [0, +\infty)$, $(\partial_p \Psi(\theta(t)))_{1 \leqslant p \leqslant P}$ remains of rank $P$ as a vector family in $\mathcal{B}$, then $\theta$ also follows the theoretical gradient flow of Equation (3.3).*

*Proof.* First rewrite the LHS of Equation (D.2): $\partial_t \Psi(\theta) = \sum_{p=1}^{P} \partial_p \Psi(\theta) \frac{\mathrm{d}}{\mathrm{d}t}\theta_p$

Let us now rewrite the definition of $\nabla_{K_\theta}\mathcal{L}|_{\Psi(\theta)}$:

$$\begin{aligned}\forall 1 \leqslant i \leqslant l_d : \mathrm{d}\mathcal{L}|_{\Psi(\theta)}(K_\theta(\cdot, x)_{\cdot,i}) &= \mathrm{d}\mathcal{L}|_{\Psi(\theta)} \left( \left( \sum_{p=1}^{P} \partial_p \Psi(\theta)(\cdot) \otimes \partial_p \Psi(\theta)(x) \right)_{\cdot,i} \right) \\ &= \mathrm{d}\mathcal{L}|_{\Psi(\theta)} \left( \sum_{p=1}^{P} \partial_p \Psi(\theta)(\cdot) \left(\partial_p \Psi(\theta)(x)\right)_i \right) \\ &\overset{\mathrm{d}\mathcal{L}|_{\Psi(\theta)} \text{ linear}}{=} \sum_{p=1}^{P} \left(\partial_p \Psi(\theta)(x)\right)_i \mathrm{d}\mathcal{L}|_{\Psi(\theta)} \left(\partial_p \Psi(\theta)(\cdot)\right).\end{aligned}$$

This gives for the RHS of Equation (D.2):

$$-\nabla_{K_\theta}\mathcal{L}|_{\Psi(\theta)} = \sum_{p=1}^{P} \partial_p \Psi(\theta) \left( -\mathrm{d}\mathcal{L}|_{\Psi(\theta)} \left(\partial_p \Psi(\theta)\right) \right),$$

yielding

$$\sum_{p=1}^{P} \partial_p \Psi(\theta) \left( \frac{\mathrm{d}}{\mathrm{d}t}\theta_p + \mathrm{d}\mathcal{L}|_{\Psi(\theta)} \left(\partial_p \Psi(\theta)\right) \right) = 0. \tag{D.3}$$

This shows that if $\theta$ is a solutions to Equation (3.3), then $\theta$ is also a solution to Equation (D.2). Conversely, if for all $t \in [0, +\infty)$, $\left(\partial_p \Psi(\theta(t))\right)_{1 \leqslant p \leqslant P}$ remains of rank $P$ as a vector family, we have: for all $t \in [0, +\infty)$

$$\sum_{p=1}^{P} \partial_p \Psi(\theta(t)) \left( \frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) + \mathrm{d}\mathcal{L}|_{\Psi(\theta(t))} \left(\partial_p \Psi(\theta(t))\right) \right) = 0,$$

implies that: for all $1 \leqslant p \leqslant P$

$$\frac{\mathrm{d}}{\mathrm{d}t}\theta_p(t) + \mathrm{d}\mathcal{L}|_{\Psi(\theta(t))} \left(\partial_p \Psi(\theta(t))\right) = 0,$$

which means that $\theta$ is also a solution to Equation (D.2). □

One may wonder for which $\theta \in \mathbb{R}^P$ is the family of vectors $(\partial_p \Psi(\theta))_{1 \leqslant p \leqslant P}$ not of rank $P$. This is a difficult problem, *a priori* depending on the underlying architecture, but this phenomenon can happen. Indeed, if we imagine an architecture whose widths are of the form $(a, 1, 1, b)$ with $a, b \in \mathbb{N}_1$, then and if we set to zero the weight and bias between layers 1 and 2 of size 1, and denote $i$ and $j$ their coordinates in parameter $\theta \in \mathbb{R}^P$, then $\Psi$ will be a constant function with respect to the other coordinates of parameter $\theta$, and so in particular for all $1 \leqslant p \leqslant P$ such that $p \neq i$ and $p \neq j$, we will have $\partial_p \Psi = 0$ and so in particular the family $\left(\partial_p \Psi(\theta(t))\right)_{1 \leqslant p \leqslant P}$ will not be of rank $P$. Note, however, that if we assume that there exists a neighborhood $V$ of $\theta$ in $\mathbb{R}^P$ such that $\Psi(V)$ is a submanifold, the former example corresponds exactly to the fact that the submanifold $\Psi(V)$ is also of lower dimension at the point $\Psi(\theta)$. In fact, we can even see that $\left(\partial_p \Psi(\theta(t))\right)_{1 \leqslant p \leqslant P}$ is a generating family of the tangent space of $\Psi(V)$ at the point $\theta$ and thus $\Psi(V)$ is exactly of dimension given by the rank of $\left(\partial_p \Psi(\theta(t))\right)_{1 \leqslant p \leqslant P}$. With this observation in mind, we propose to reinterpret the NTK in the next section.

## D.3. Connection with Riemmanian metrics of (pseudo-)manifolds

Let be $\mu$ a finite measure on $\mathbb{R}^{l_0}$, $\Psi \in \mathcal{C}^\infty\left(\mathbb{R}^P \to \mathrm{L}^2_\mu(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})\right)$ a neural network architecture as in Definition 3.2, with $\mathrm{L}^2_\mu(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})$ being the space of $\mu$ integrable functions, and $\theta \in \mathbb{R}^P$. As in the former section, let suppose that there exist a neighborhood $V$ of $\theta$ in $\mathbb{R}^P$ such that $\Psi(V)$ is a submanifold. Then the tensor:

$$T_{\mathrm{NTK}} : \left\{ \begin{array}{ccc} V \times \mathrm{L}^2_\mu(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})^2 & \longrightarrow & \mathbb{R} \\ (\theta, f, g) & \longmapsto & \int_{\mathbb{R}^{l_0}} \int_{\mathbb{R}^{l_0}} f(x)^T K_\theta(x,y) g(y) \mu(\mathrm{d}x) \mu(\mathrm{d}y) \end{array} \right. \tag{D.4}$$

is a Riemmanian-metric on the tangent bundle of $\Psi(V)$, where $K_\theta$ is the NTK defined in Appendix D.1.

*Proof.* The regularity of $T_{\mathrm{NTK}}$ with respect to variables $\theta$ comes from the regularity of $\Psi$. Symmetry in variables $f$ and $g$ is clear. To check positivity, let bet $f \in \mathrm{L}^2_\mu(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})$. Then: for all $\theta \in \mathbb{R}^P$

$$\begin{aligned} T_{\mathrm{NTK}}(\theta, f, f) &= \int_{\mathbb{R}^{l_0}} \int_{\mathbb{R}^{l_0}} \sum_{p=1}^P f(x)^T \partial_p \Psi(\theta)(x) \otimes \partial_p \Psi(\theta)(y) f(y) \mu(\mathrm{d}x) \mu(\mathrm{d}y) \\ &= \int_{\mathbb{R}^{l_0}} \int_{\mathbb{R}^{l_0}} \sum_{p=1}^P \langle f(x) | \partial_p \Psi(\theta)(x) \rangle_{\mathbb{R}^{l_d}} \langle f(y) | \partial_p \Psi(\theta)(y) \rangle_{\mathbb{R}^{l_d}} \mu(\mathrm{d}x) \mu(\mathrm{d}y) \\ &\overset{\mu \text{ finite}}{=} \sum_{p=1}^P \int_{\mathbb{R}^{l_0}} \int_{\mathbb{R}^{l_0}} \langle f(x) | \partial_p \Psi(\theta)(x) \rangle_{\mathbb{R}^{l_d}} \langle f(y) | \partial_p \Psi(\theta)(y) \rangle_{\mathbb{R}^{l_d}} \mu(\mathrm{d}x) \mu(\mathrm{d}y) \\ &= \sum_{p=1}^P \left( \int_{\mathbb{R}^{l_0}} \langle f(x) | \partial_p \Psi(\theta)(x) \rangle_{\mathbb{R}^{l_d}} \mu(\mathrm{d}x) \right)^2 \geqslant 0 \end{aligned} \tag{D.5}$$

Finally to check that this tensor is definite on $\Psi(V)$ bundle, from Equation (D.5), we get that: for all $\theta \in \mathbb{R}^P$ and for all $f \in \mathrm{L}^2_\mu(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})$

$$\begin{aligned} T_{\mathrm{NTK}}(\theta, f, f) = 0 &\iff \sum_{p=1}^P \left( \int_{\mathbb{R}^{l_0}} \langle f(x) | \partial_p \Psi(\theta)(x) \rangle_{\mathbb{R}^{l_d}} \mu(\mathrm{d}x) \right)^2 = 0 \\ &\iff \forall 1 \leqslant p \leqslant P \ : \ \int_{\mathbb{R}^{l_0}} \langle f(x) | \partial_p \Psi(\theta)(x) \rangle_{\mathbb{R}^{l_d}} \mu(\mathrm{d}x) = 0 \\ &\iff \forall 1 \leqslant p \leqslant P \ : \ \langle f | \partial_p \Psi(\theta) \rangle_{\mathrm{L}^2_\mu(\mathbb{R}^{l_0} \to \mathbb{R}^{l_d})} = 0 \\ &\iff f \in \mathrm{Span}\left(\partial_p \Psi(\theta)\right)^\perp_{1 \leqslant p \leqslant P} \end{aligned}$$

But as $\mathrm{Span}\left(\partial_p \Psi(\theta)\right)_{1 \leqslant p \leqslant P}$ is precisely the tangent space of $\Psi(V)$ at point $\Psi(\theta)$, we see that: for all $\theta \in \mathbb{R}^P$ and for all $f$ in the tangent space of $\Psi(V)$ at point $\Psi(\theta)$

$$\begin{aligned} T_{\mathrm{NTK}}(\theta, f, f) = 0 &\iff f \in \mathrm{Span}\left(\partial_p \Psi(\theta)\right)^\perp_{1 \leqslant p \leqslant P} \bigcap \mathrm{Span}\left(\partial_p \Psi(\theta)\right)_{1 \leqslant p \leqslant P} = \{0\} \\ &\iff f = 0, \end{aligned}$$

which concludes our proof. □

We can argue that the assumptions made about the existence of a neighborhood $V$ are far too strong. In fact, we think that Riemanian submanifolds are a too rigid framework for this formalism, so we plan to study in a future work the framework of stratified spaces, also known as pseudo-manifolds. To our knowledge, there is no classical introduction to this subject. However, one could refer to Feragen and Nye (2020).

# E. Complement to the proofs of Shin et al. (2020)

We focus on proving the first inequality of the proof of Theorem 3.4 (Appendix E) in Shin et al. (2020), which is not straightforward. We first remark that $\mathcal{L}[\tilde{e}]$ can be rewritten:

$$\mathcal{L}[\tilde{e}] = \operatorname{div}(A\nabla\tilde{e} + b\tilde{e}) + c \cdot \nabla\tilde{e} + d\tilde{e}$$

Thus

$$-\langle \mathcal{L}[\tilde{e}] | \tilde{e} \rangle_{L^2} = -\underbrace{\int_U \tilde{e} \operatorname{div}(A\nabla\tilde{e} + b\tilde{e})\, d\lambda}_{=:P_1} - \int_U \tilde{e}c \cdot \nabla\tilde{e}\, d\lambda - \int_U d\tilde{e}^2\, d\lambda$$

Then we have by Green's identity and using $\tilde{e}|_{\partial U} = 0$:

$$\begin{aligned} P_1 &= -\int_U (A\nabla\tilde{e} + b\tilde{e}) \cdot \nabla\tilde{e}\, d\lambda + \int_{\partial U} \underbrace{\tilde{e}}_{=0} (A\nabla\tilde{e} + b\tilde{e}) \cdot d\sigma \\ &= -\int_U (\nabla\tilde{e})^T A\nabla\tilde{e}\, d\lambda - \int_U \tilde{e}b \cdot \nabla\tilde{e}\, d\lambda \end{aligned}$$

Yielding :

$$\begin{aligned} -\langle \mathcal{L}[\tilde{e}] | \tilde{e} \rangle_{L^2} &= \int_U (\nabla\tilde{e})^T A\nabla\tilde{e}\, d\lambda + \int_U \tilde{e}b \cdot \nabla\tilde{e}\, d\lambda - \int_U \tilde{e}c \cdot \nabla\tilde{e}\, d\lambda - \int_U d\tilde{e}^2\, d\lambda \\ &\geqslant \int_U \lambda_0 \|\nabla\tilde{e}\|_2^2\, d\lambda + \langle \tilde{e}(b-c) | \nabla\tilde{e} \rangle_{L^2} - \int_U d\tilde{e}^2\, d\lambda \qquad \text{(E.1)} \\ &\geqslant \lambda_0 \|\nabla\tilde{e}\|_{L^2}^2 - \|\tilde{e}(b-c)\|_{L^2} \|\nabla\tilde{e}\|_{L^2} - \int_U d\tilde{e}^2\, d\lambda \qquad \text{(E.2)} \\ &= \left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 - \left\|\tilde{e}(b-c)\lambda_0^{-\frac{1}{2}}\right\|_{L^2} \left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2} - \int_U d\tilde{e}^2\, d\lambda \\ &= \frac{1}{2}\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 + \frac{1}{2}\left(\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 - 2\left\|\tilde{e}(b-c)\lambda_0^{-\frac{1}{2}}\right\|_{L^2} \left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}\right) - \int_U d\tilde{e}^2\, d\lambda \\ &\geqslant \frac{1}{2}\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 - \frac{1}{2}\left\|\tilde{e}(b-c)\lambda_0^{-\frac{1}{2}}\right\|_{L^2}^2 - \int_U d\tilde{e}^2\, d\lambda \\ &= \frac{1}{2}\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 - \frac{1}{2}\int_U \tilde{e}^2 \|b-c\|_2^2 \lambda_0^{-1}\, d\lambda - \int_U d\tilde{e}^2\, d\lambda \\ &= \frac{1}{2}\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 - \frac{1}{2}\int_U \tilde{e}^2 \left(2\|b\|_2^2 + 2\|c\|_2^2 - \|b+c\|_2^2\right)\lambda_0^{-1}\, d\lambda - \int_U d\tilde{e}^2\, d\lambda \qquad \text{(E.3)} \\ &= \frac{1}{2}\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 - \int_U \tilde{e}^2 \left(\|b\|_2^2 + \|c\|_2^2\right)\lambda_0^{-1}\, d\lambda + \frac{1}{2}\underbrace{\left\|\tilde{e}\|b+c\|_2 \lambda_0^{-\frac{1}{2}}\right\|_{L^2}}_{\geqslant 0} - \int_U d\tilde{e}^2\, d\lambda \\ &\geqslant \frac{1}{2}\left\|\lambda_0^{\frac{1}{2}}\nabla\tilde{e}\right\|_{L^2}^2 + \int_U \tilde{e}^2 \underbrace{\left[-\left(\|b\|_2^2 + \|c\|_2^2\right)\lambda_0^{-1} - |d|\right]}_{\geqslant -\lambda_0 v^2}\, d\lambda \qquad \text{(E.4)} \\ &= \frac{\lambda_0}{2}\|\nabla\tilde{e}\|_{L^2}^2 - \int_U \tilde{e}^2 \lambda_0 v^2\, d\lambda = \frac{\lambda_0}{2}\|\nabla\tilde{e}\|_{L^2}^2 - \lambda_0 v^2 \|\tilde{e}\|_{L^2}^2 \end{aligned}$$

where inequality (E.1) comes the fact that $A$ is symmetric with $\min \operatorname{Spec}(A) \geqslant \lambda_0$ by Assumption 3.3.1 ; inequality (E.2) is Cauchy-Schwartz minoration ; equality (E.3) is the parallelogram law and finally (E.4) comes from Assumption 3.3.5.

# F. Additional experiment results content

## F.1. Tools used to perform the experiments

**Hardware:** All the results presented in this report were obtained from experiments carried out on the cluster of the TAU team at the Université Paris-Saclay, on an instance comprising 32 cpu threads and 4 Nvidia® GeForce RTX 2080 Ti graphics cards with 12GB of VRAM memory[20].

Nevertheless, most of research has been conducted on BwUniCluster2.0[21] instances with dedicated Nvidia® tesla V100 architectures[22].

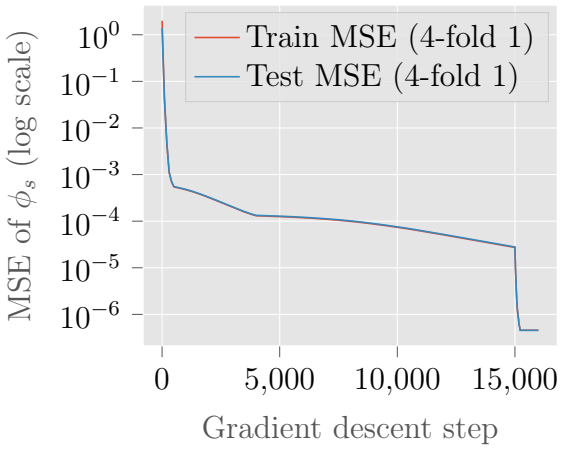
**Software:** All our expriments were coded in python programming language[23]. For neural networks training, we used the pytorch library (Paszke et al., 2019) built upon cuda toolkit[24] for gpu acceleration and some facilities of the PINNs specific library deepxde (Lu et al., 2021), as well as the numpy library (Harris et al., 2020) for any matrix manipulation. For the sampling process (see Section 4.1.1), the scipy (Virtanen et al., 2020) library was used, while data manipulation was achieved thanks to pandas library (McKinney, 2010). The figures below were generated using matplotlib library (Hunter, 2007), tensorboard visualization tool from the tensorflow library (Abadi et al., 2016) and the tikzplotlib pacakge[25].

All the code used to run our experiments will be available at https://github.com/IloneM/NS-PINNs-MA.

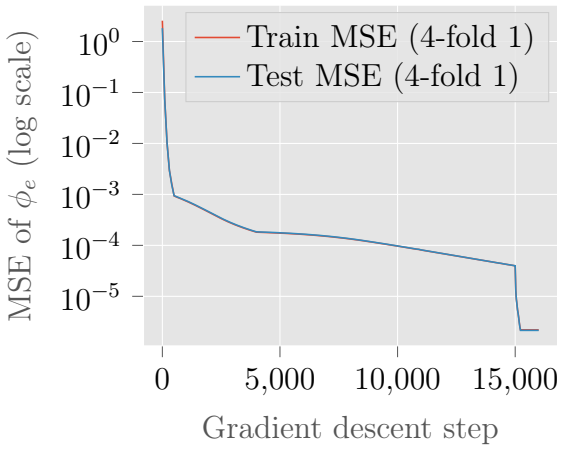
[20] https://www.nvidia.com/en-us/geforce/graphics-cards/compare/?section=compare-20
[21] https://wiki.bwhpc.de/e/Category:BwUniCluster_2.0
[22] https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5-pdf
[23] https://www.python.org/
[24] https://developer.nvidia.com/cuda-toolkit
[25] https://pypi.org/project/tikzplotlib/

## F.2. Complementary experimental results to Section 4.2.3

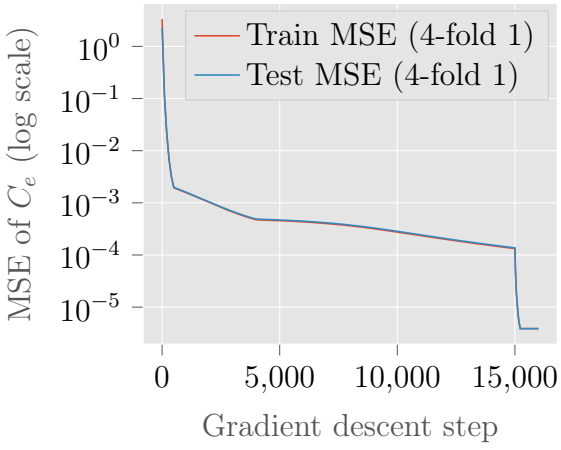
### F.2.1. Complementary MSEs for each variable for all 4-folds



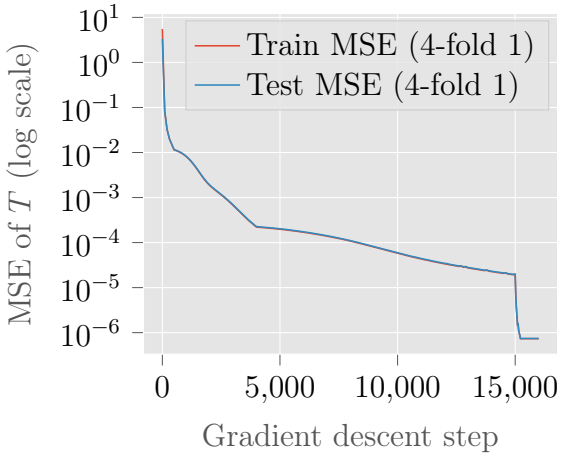

(a) MSE of solid potential ($\phi_s$)



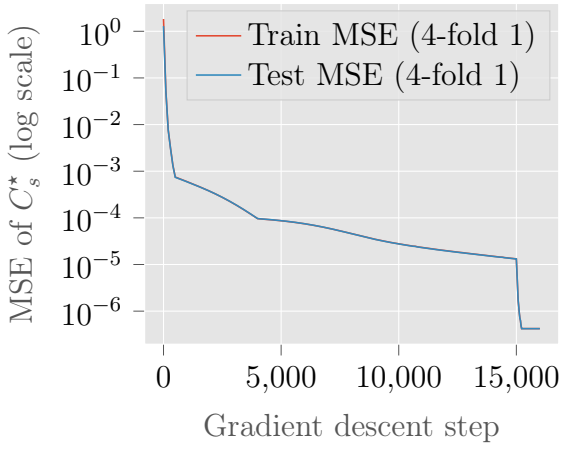

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

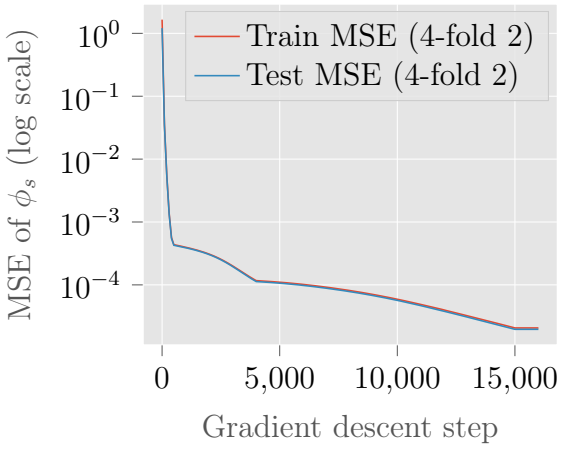



(d) MSE of temperature ($T$)

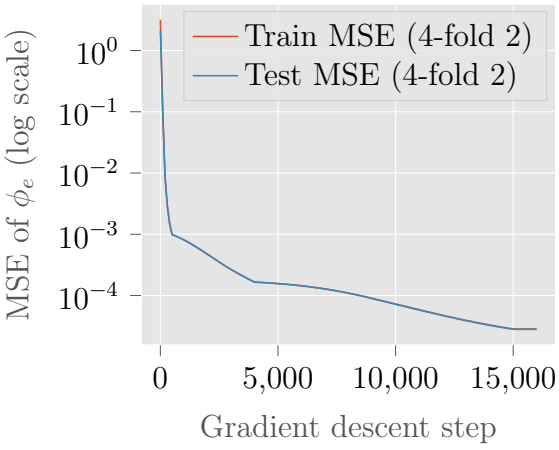



(e) MSE of lithium ions surface concentration in the solid-phase ($C_s^\star$)

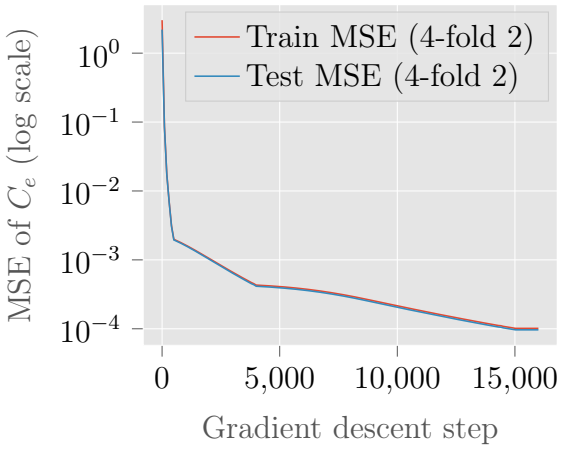
Figure 29: Train and Test MSEs of each of the variables of the model for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ on 4-fold 1, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.



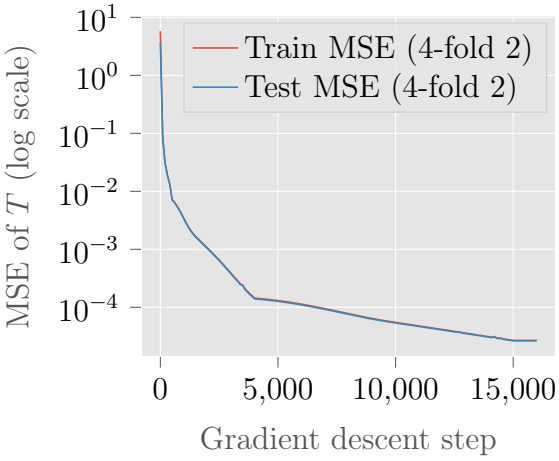

(a) MSE of solid potential ($\phi_s$)



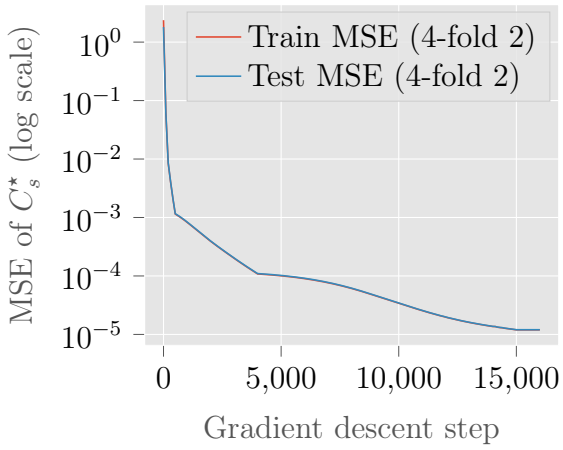

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

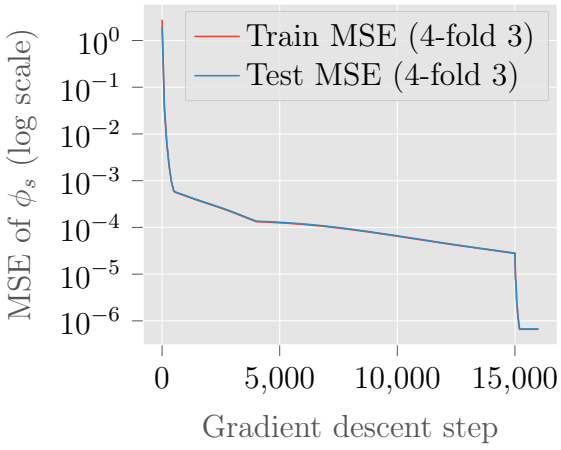



(d) MSE of temperature ($T$)

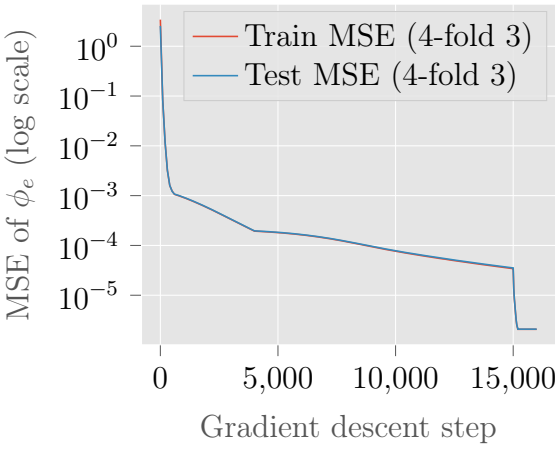



(e) MSE of lithium ions surface concentration in the solid-phase ($C_s^\star$)

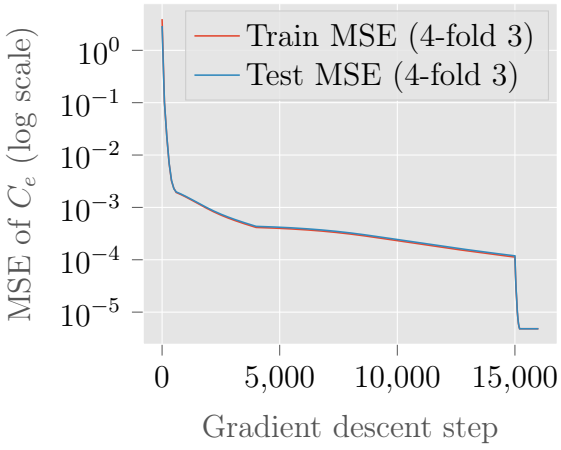
Figure 30: Train and Test MSEs of each of the variables of the model for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ on 4-fold 2, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000. We see that L-BFGS fails to converge for all variables.



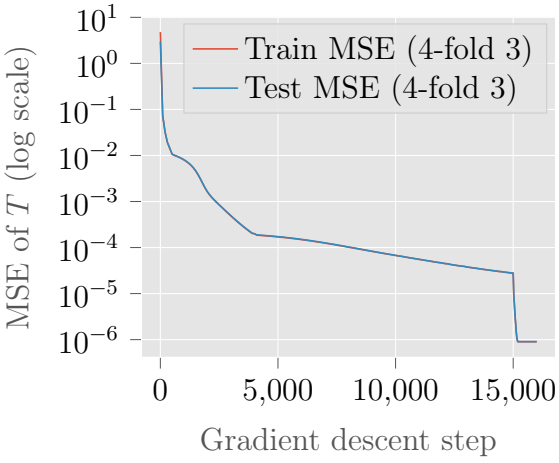

(a) MSE of solid potential ($\phi_s$)



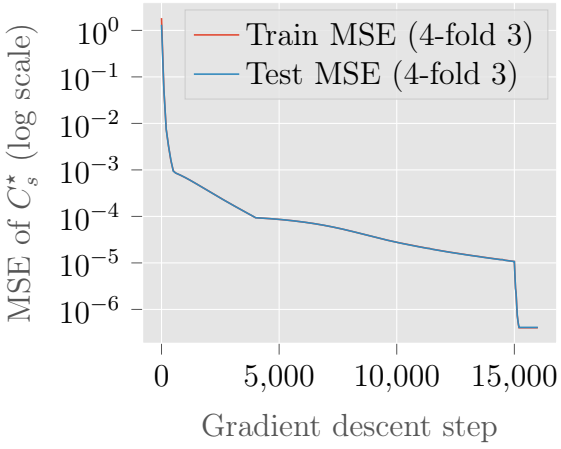

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)

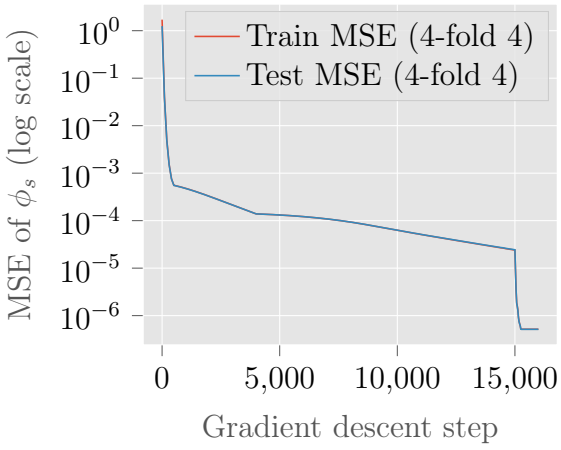



(d) MSE of temperature ($T$)

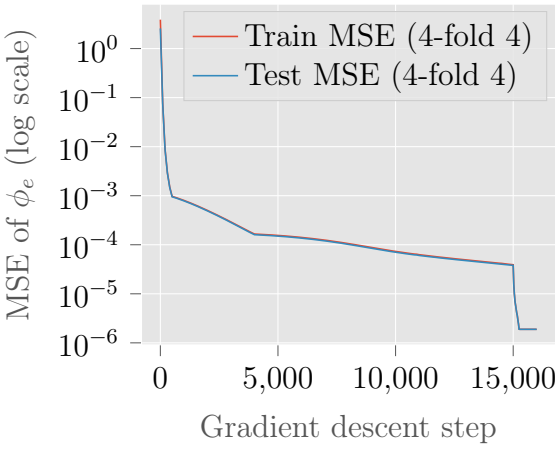



(e) MSE of lithium ions surface concentration in the solid-phase ($C_s^\star$)

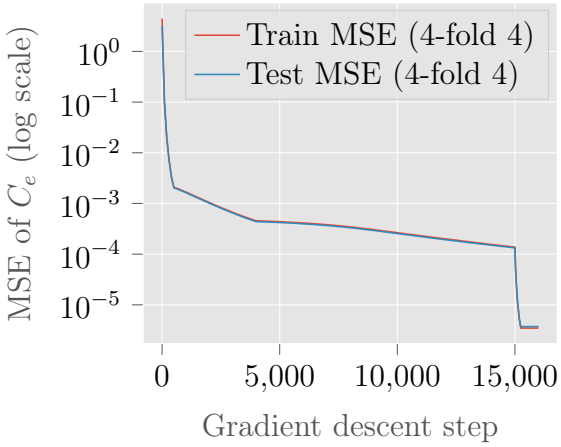
Figure 31: Train and Test MSEs of each of the variables of the model for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ on 4-fold 3, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.



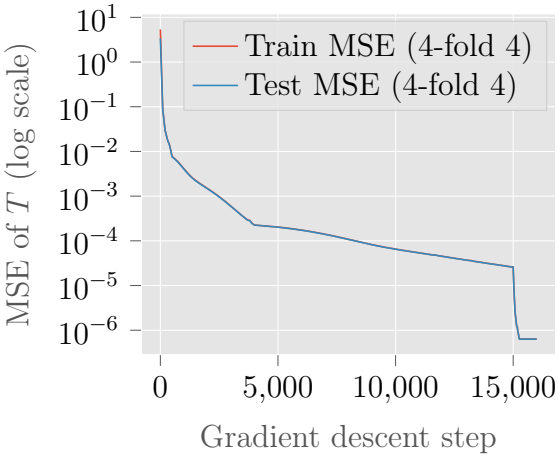

(a) MSE of solid potential ($\phi_s$)



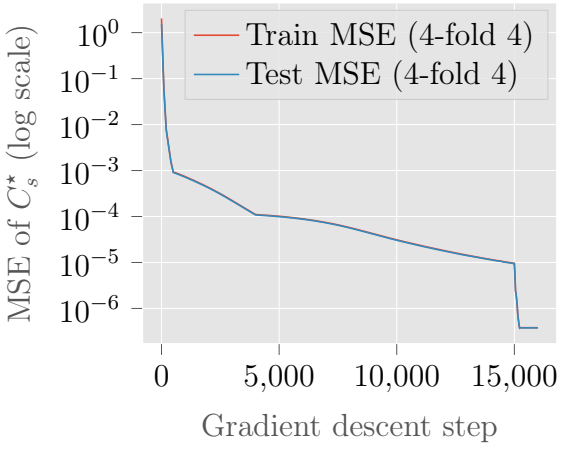

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)




(d) MSE of temperature ($T$)

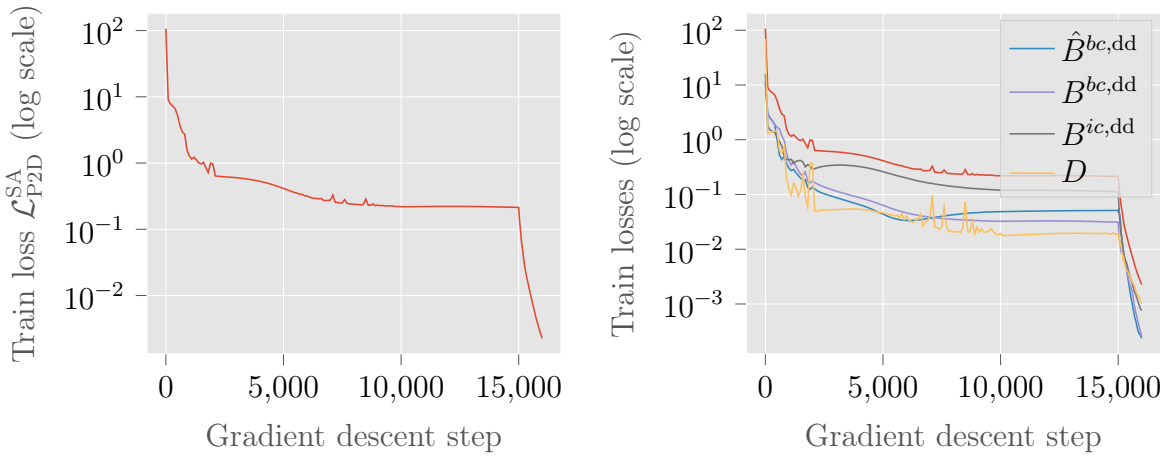



(e) MSE of lithium ions surface concentration in the solid-phase ($C_s^\star$)

Figure 32: Train and Test MSEs of each of the variables of the model for the data driven approach with weight correction, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ on 4-fold 4, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.

## F.3. Complementary results to Section 4.3

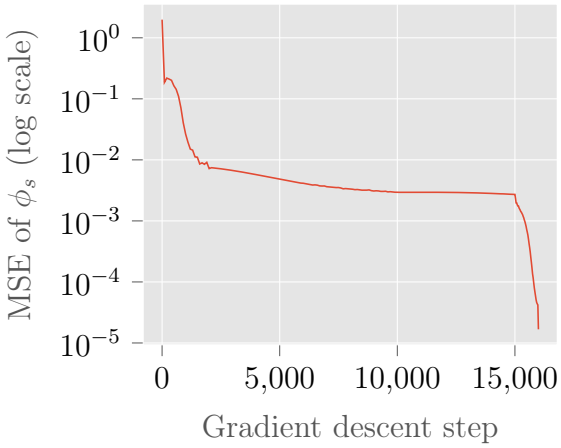
### F.3.1. Results for $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$



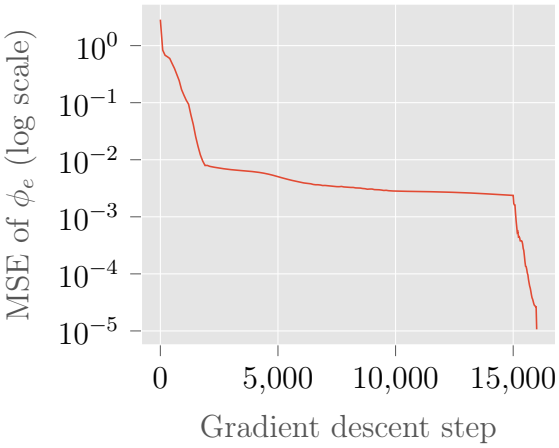

(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

(b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

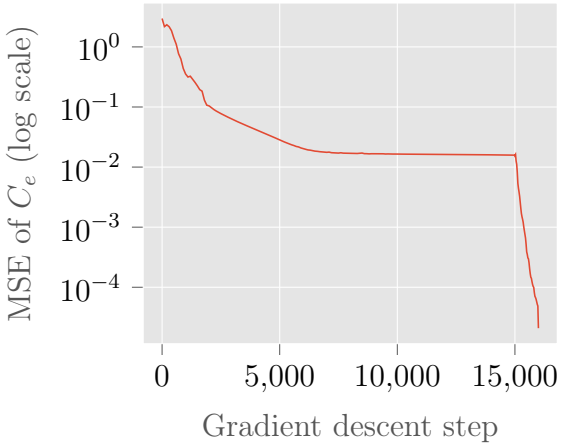
Figure 33: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the learning process succeeded, the loss losing more than 4 orders of magnitude during training, indicating that the neural network did learn both the boundary conditions data, and the losses from the P2D model PDEs. We also find that each loss component contributes exactly the same orders of magnitude to the total loss, meaning that each component has been learned equally.



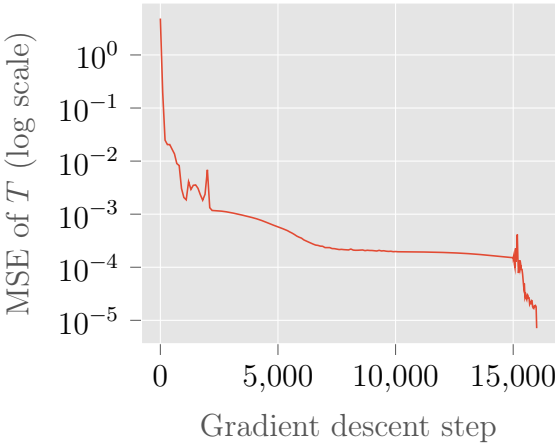

(a) MSE of solid potential ($\phi_s$)



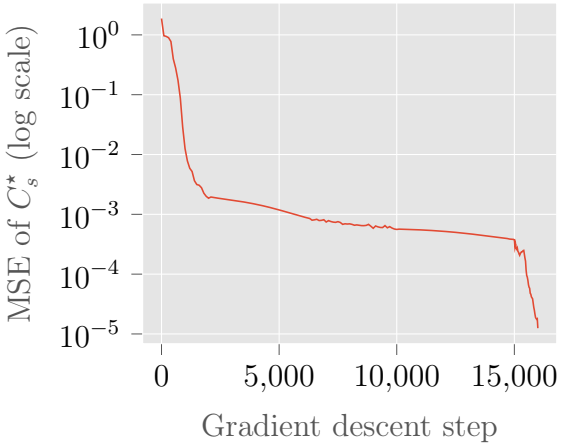

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)



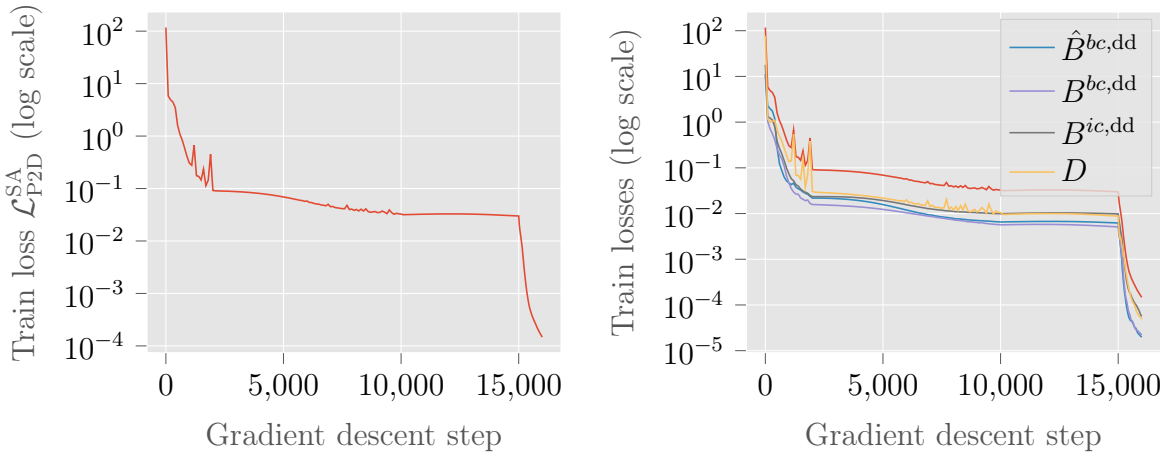

(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

Figure 34: MSEs of each of the variables of the model for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000. We observe that all variables are equally learned by the neural networks, each MSE losing 5 orders of magnitudes during the training process, which means that the neural network generalizes very well thanks to the equations enforced in the loss through operator $D$.

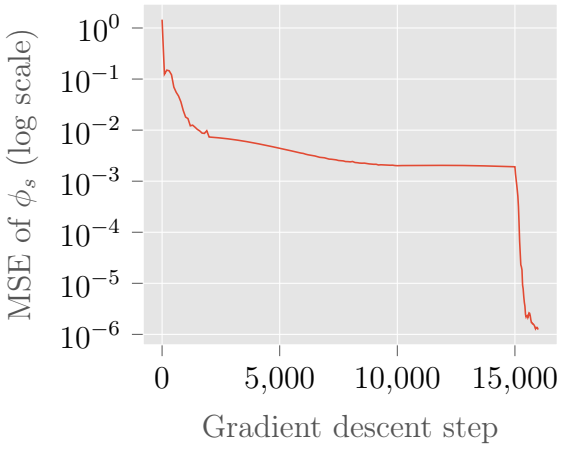
### F.3.2. Results for $I_{\text{app}} = -20\,\text{A·m}^{-2}$



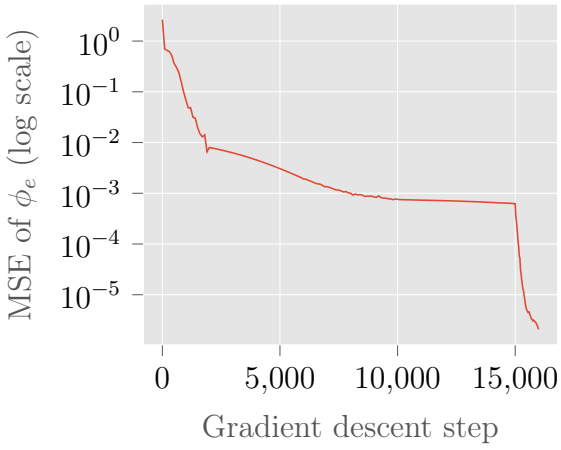

(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

(b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

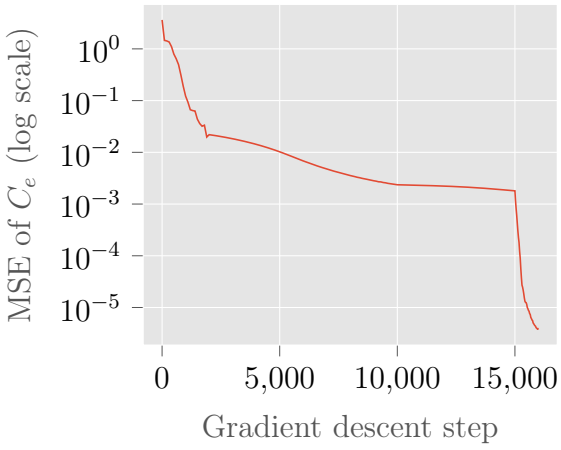
Figure 35: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -20\,\text{A·m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the learning process succeeded, the loss losing 6 orders of magnitude during training, indicating that the neural network did learn both the boundary conditions data, and the losses from the P2D model PDEs.
We also find that each loss component contributes exactly the same orders of magnitude to the total loss, meaning that each component has been learned equally.



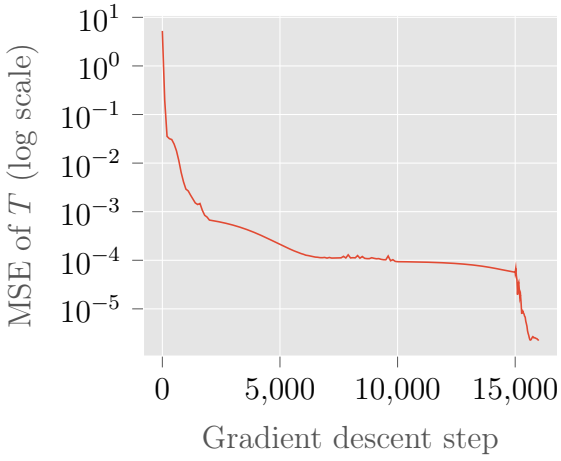

(a) MSE of solid potential ($\phi_s$)



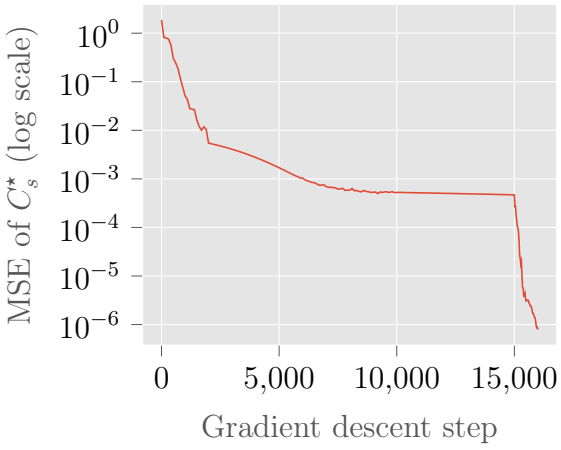

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)



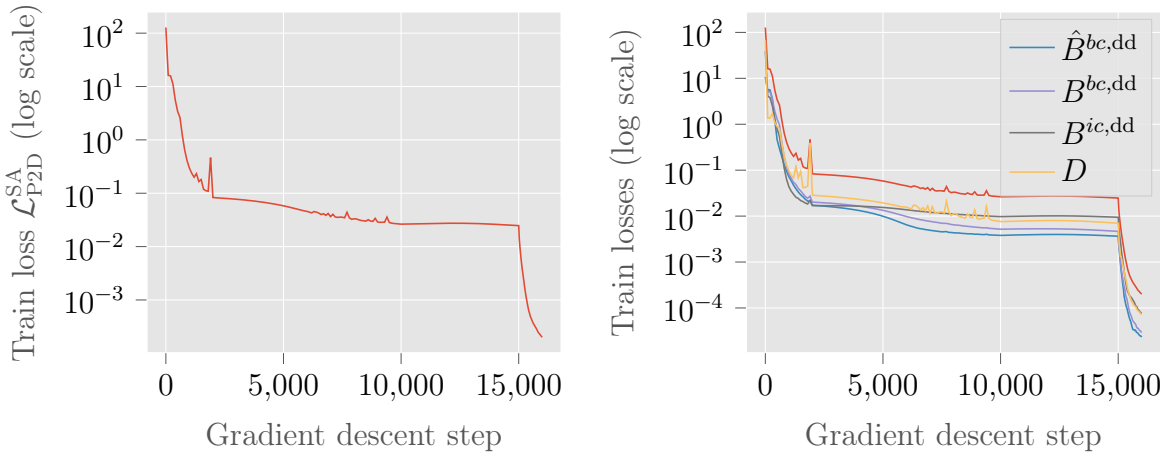

(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

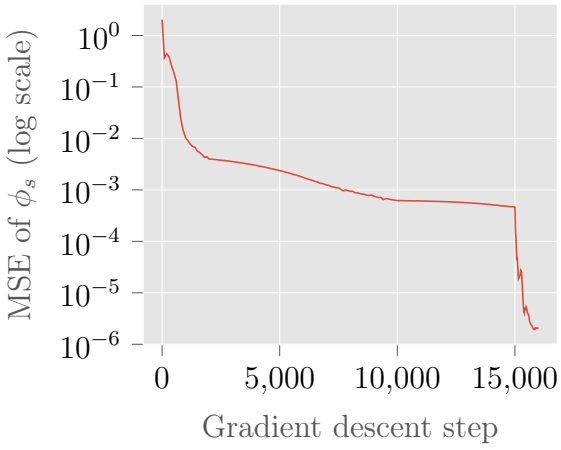
Figure 36: MSEs of each of the variables of the model for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = -20\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000. We observe that all variables are equally learned by the neural networks, each MSE losing betwenn 5 and 7 orders of magnitudes during the training process, which means that the neural network generalizes very well thanks to the equations enforced in the loss through operator $D$.

### F.3.3. Results for $I_{\text{app}} = 20\,\text{A}\cdot\text{m}^{-2}$

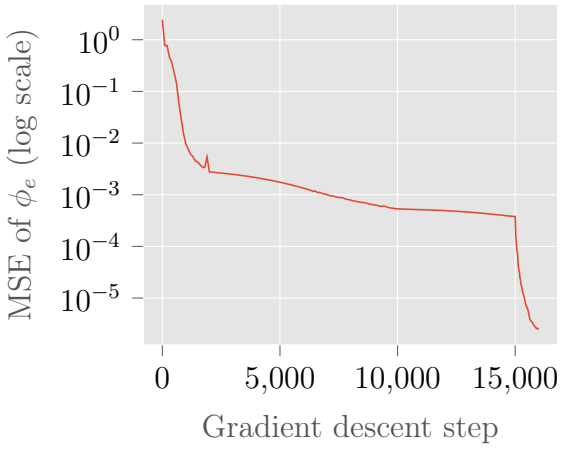



(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ (b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

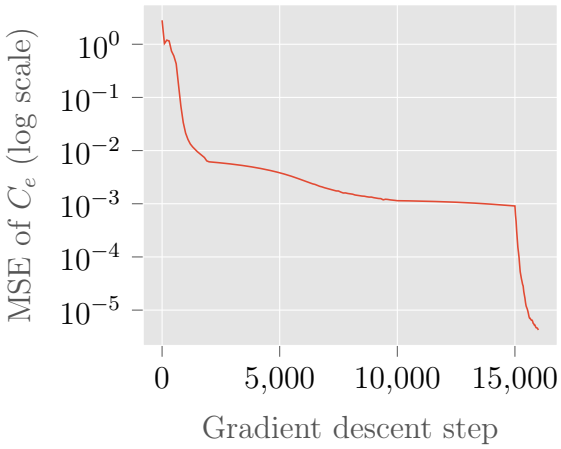
Figure 37: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 20\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the learning process succeeded, the loss losing more than 5 orders of magnitude during training, indicating that the neural network did learn both the boundary conditions data, and the losses from the P2D model PDEs. We also find that each loss component contributes exactly the same orders of magnitude to the total loss, meaning that each component has been learned equally.



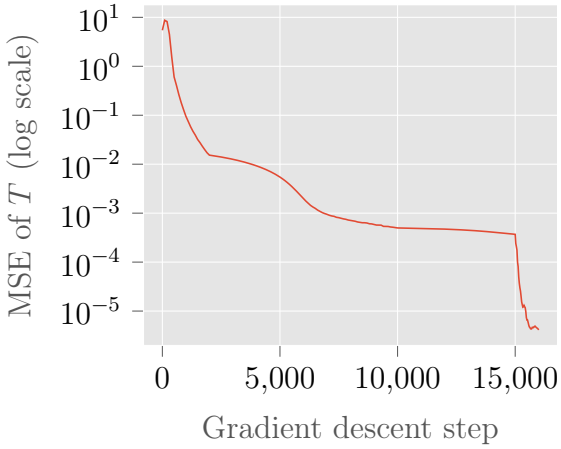

(a) MSE of solid potential ($\phi_s$)



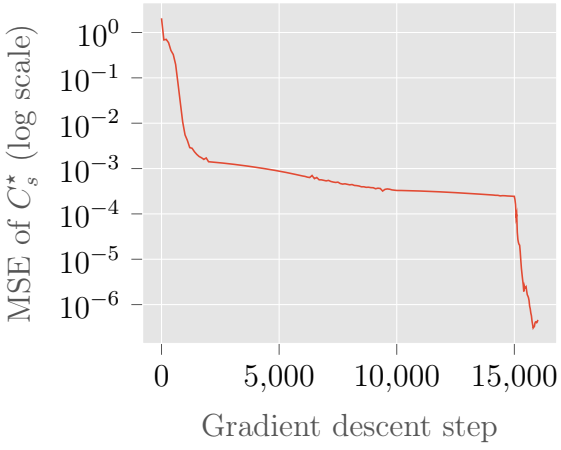

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)



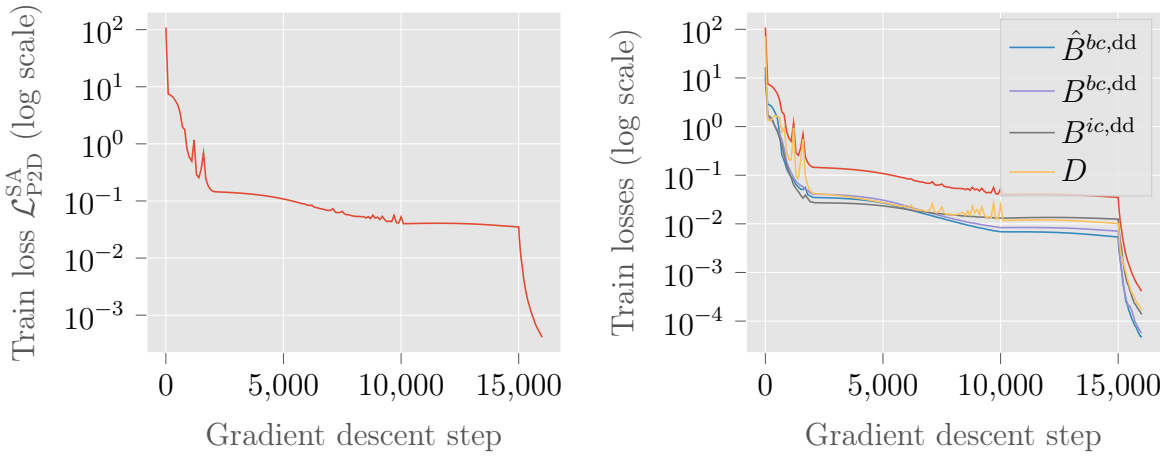

(d) MSE of temperature ($T$)




(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

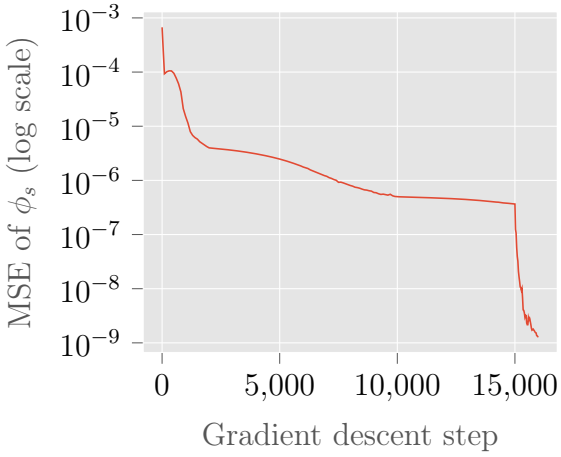
Figure 38: MSEs of each of the variables of the model for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 20\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000. We observe that all variables are equally learned by the neural networks, each MSE losing between 5 and 6 orders of magnitudes during the training process, which means that the neural network generalizes very well thanks to the equations enforced in the loss through operator $D$.

### F.3.4. Results for $I_{\text{app}} = 40\,\text{A·m}^{-2}$

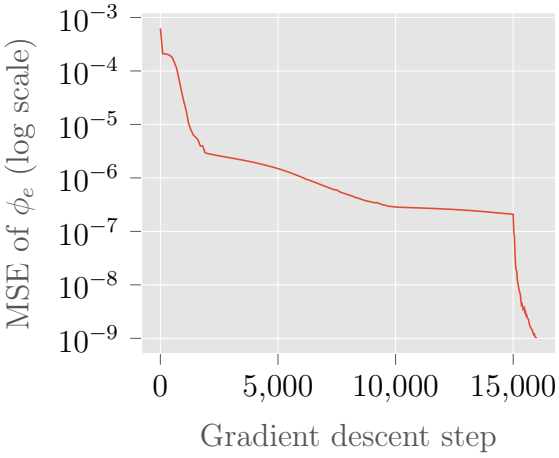



(a) Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ (b) Components contribution to train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$

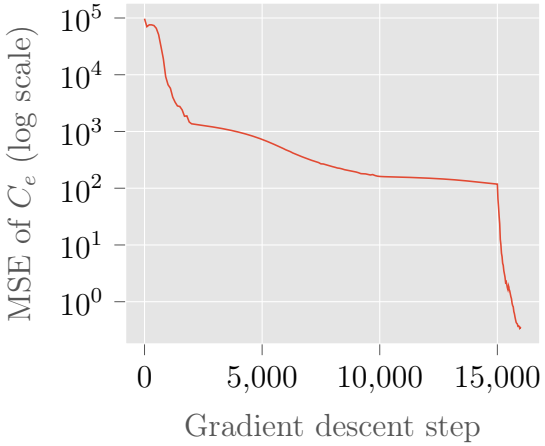
Figure 39: Train loss $\mathcal{L}^{\text{SA}}_{\text{P2D}}$ for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A·m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000.
We see that the learning process succeeded, the loss losing 5 orders of magnitude during training, indicating that the neural network did learn both the boundary conditions data, and the losses from the P2D model PDEs.
We also find that each loss component contributes exactly the same orders of magnitude to the total loss, meaning that each component has been learned equally.



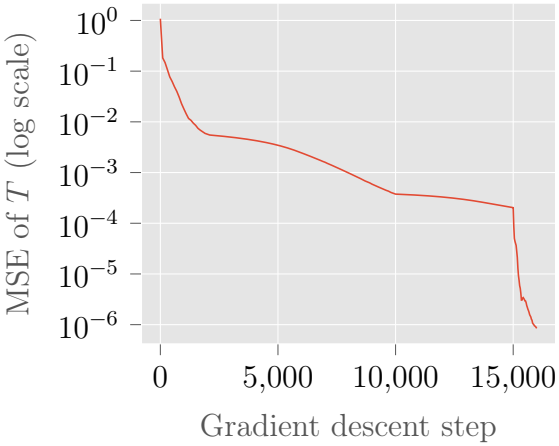

(a) MSE of solid potential ($\phi_s$)



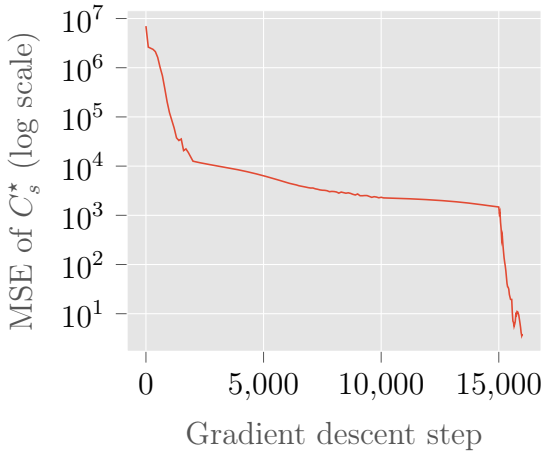

(b) MSE of electrolyte potential ($\phi_e$)




(c) MSE of lithium ions concentration in the electrolyte ($C_e$)



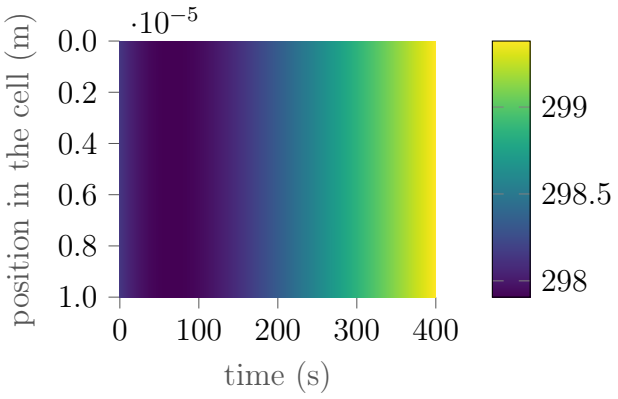

(d) MSE of temperature ($T$)



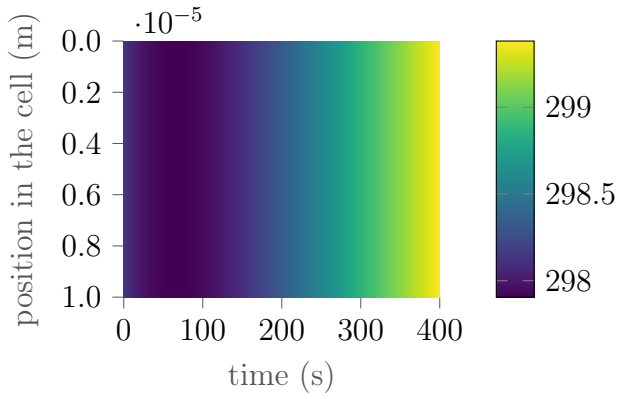

(e) MSE of lithium ions concentration in the solid-phase ($C_s^\star$)

Figure 40: MSEs of each of the variables of the model for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, with respect to the gradient descent steps with the Adam algorithm up to step 15000 and L-BFGS from step 15001 to step 16000. We observe that all variables are equally learned by the neural networks, each MSE losing 6 orders of magnitudes during the training process, which means that the neural network generalizes very well thanks to the equations enforced in the loss through operator $D$.

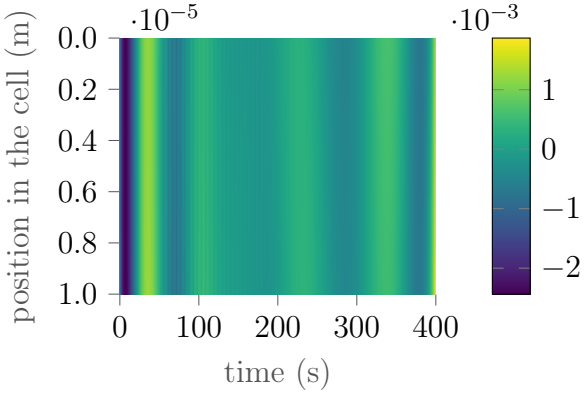
### F.3.5. Heatmaps in the positive current collector



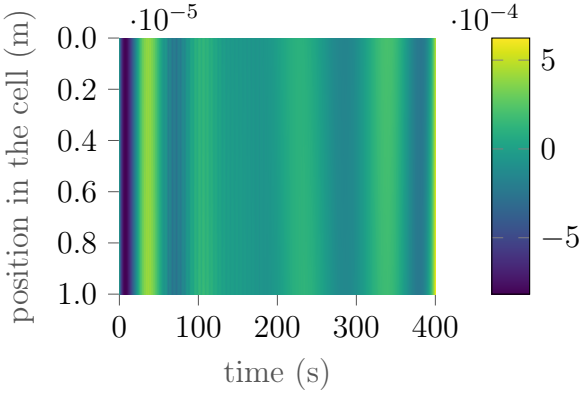

(a) Approximation of the temperature ($T$) in positive current collector with a neural network




(b) Approximation of the temperature ($T$) in positive current collector by LIONSIMBA



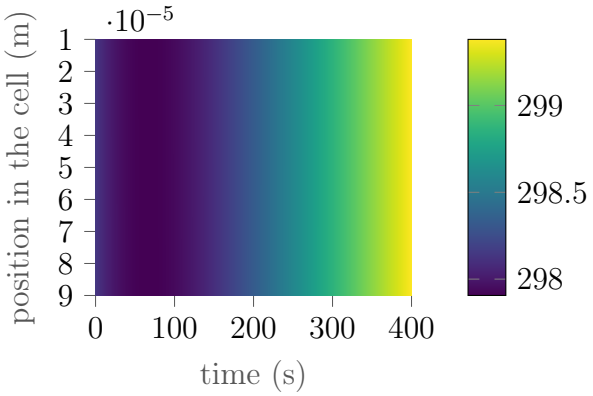

(c) Absolute error $T_{\text{LS}} - T_{\text{PINN}}$ of the temperature in positive current collector



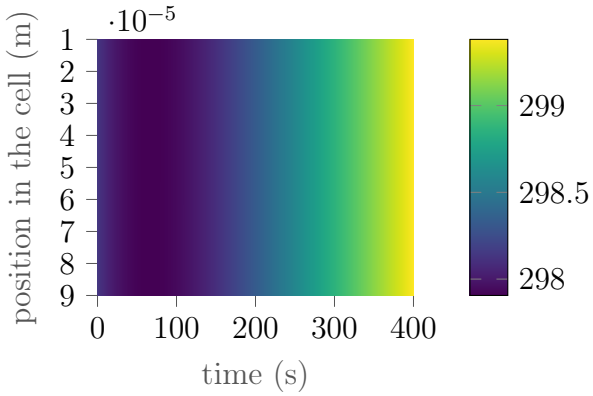

(d) Relative error $\frac{T_{\text{LS}} - T_{\text{PINN}}}{T_{\text{LS}}}$ in % of the temperature in positive current collector

Figure 41: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.0005%.

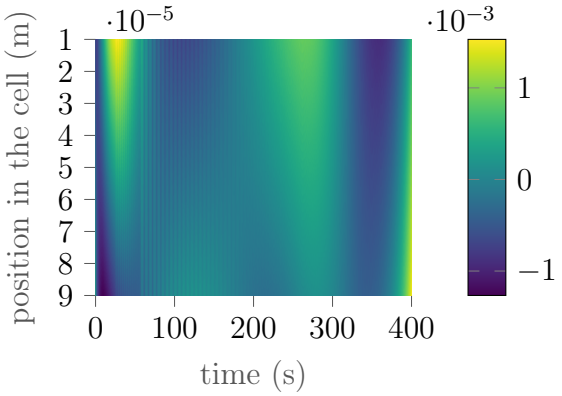
### F.3.6. Heatmaps in the cathode



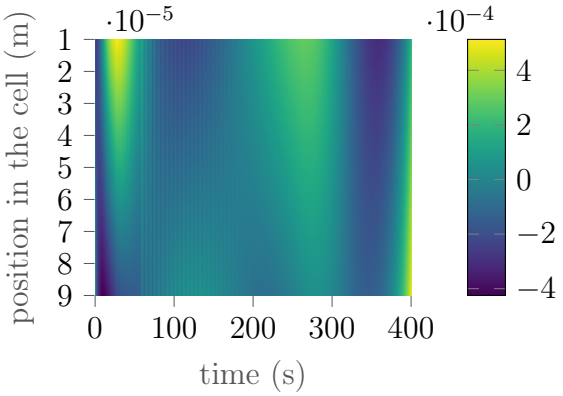

(a) Approximation of the temperature ($T$) in cathode with a neural network




(b) Approximation of the temperature ($T$) in cathode by LIONSIMBA

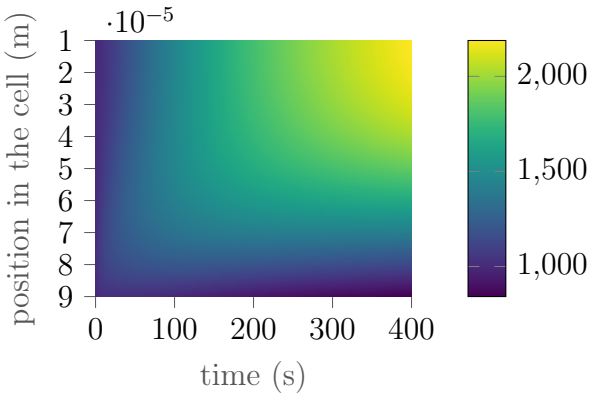



(c) Absolute error $T_{\text{LS}} - T_{\text{PINN}}$ of the temperature in cathode

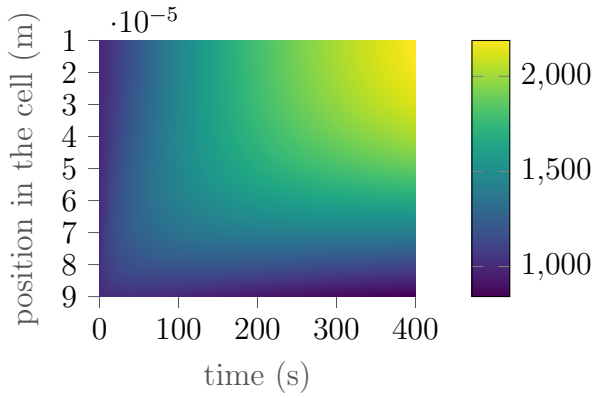



(d) Relative error $\frac{T_{\text{LS}} - T_{\text{PINN}}}{T_{\text{LS}}}$ in % of the temperature in cathode

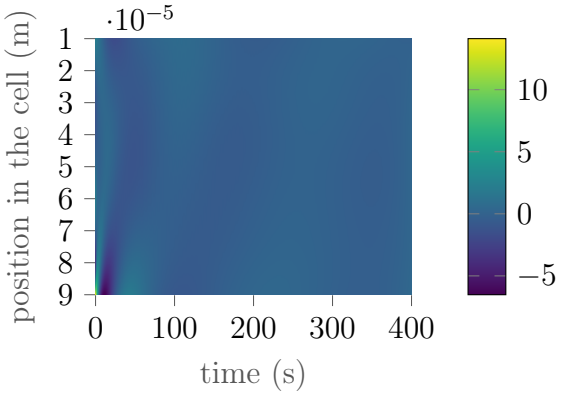
Figure 42: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.0005%.



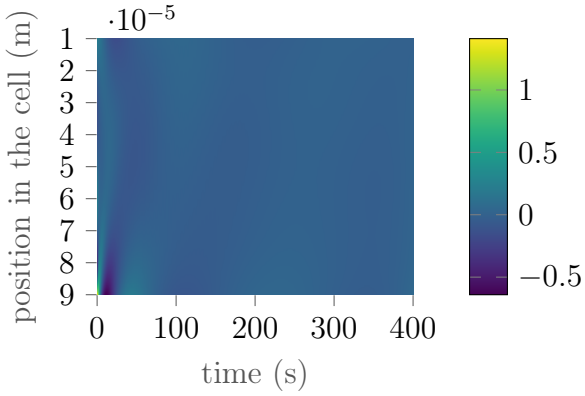

(a) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in cathode with a neural network




(b) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in cathode by LIONSIMBA

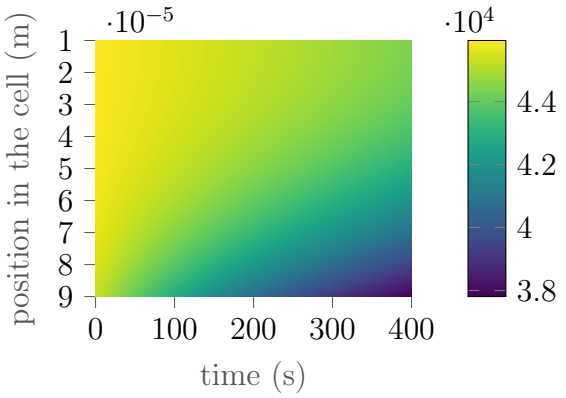



(c) Absolute error $C_{e\mathrm{LS}} - C_{e\mathrm{PINN}}$ of the lithium ions concentration in the electrolyte in cathode

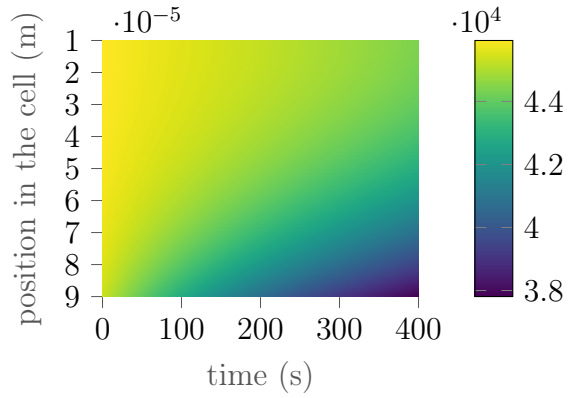



(d) Relative error $\frac{C_{e\mathrm{LS}} - C_{e\mathrm{PINN}}}{C_{e\mathrm{LS}}}$ in % of the lithium ions concentration in the electrolyte in cathode

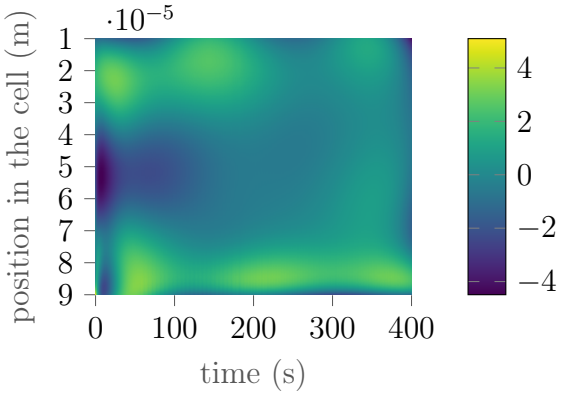
Figure 43: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\mathrm{app}} = 40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\mathrm{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of about 1%.



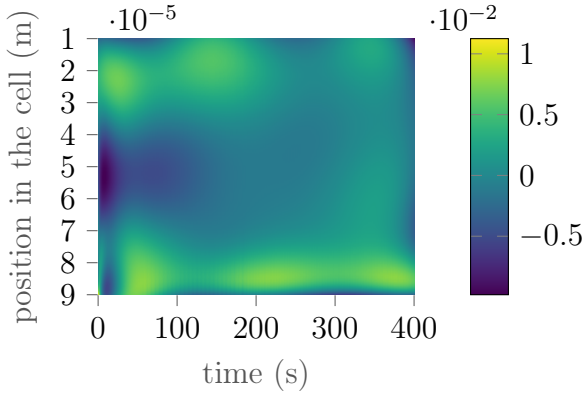

(a) Approximation of the lithium ions surface concentration in the solid-phase ($C_s^\star$) in cathode with a neural network




(b) Approximation of the lithium ions surface concentration in the solid-phase ($C_s^\star$) in cathode by LIONSIMBA

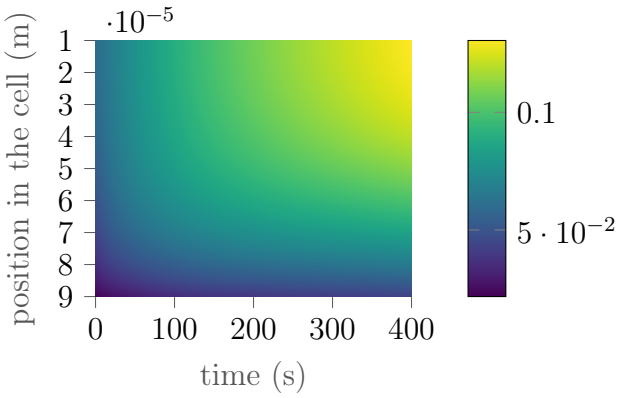



(c) Absolute error $C^\star_{s\,\mathrm{LS}} - C^\star_{s\,\mathrm{PINN}}$ of the lithium ions surface concentration in the solid-phase in cathode

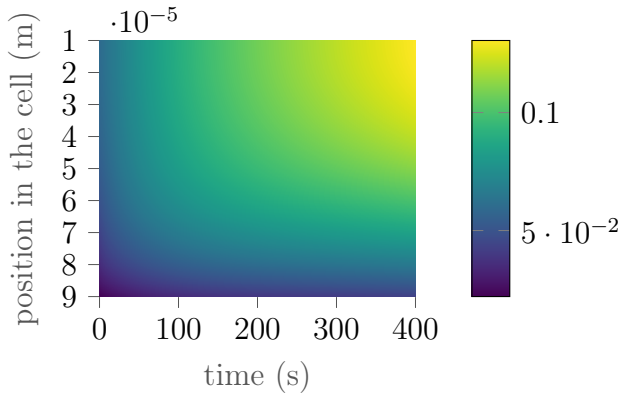



(d) Relative error $\frac{C^\star_{s\,\mathrm{LS}} - C^\star_{s\,\mathrm{PINN}}}{C^\star_{s\,\mathrm{LS}}}$ in % of the lithium ions surface concentration in the solid-phase in cathode

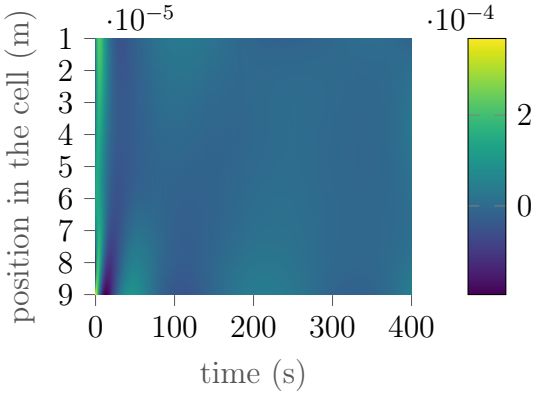
Figure 44: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\mathrm{app}} = 40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\mathrm{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of about 0.01%.



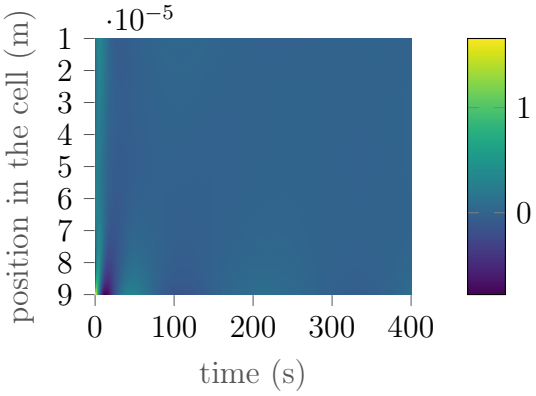

(a) Approximation of the electrolyte potential ($\phi_e$) in cathode with a neural network




(b) Approximation of the electrolyte potential ($\phi_e$) in cathode by LIONSIMBA

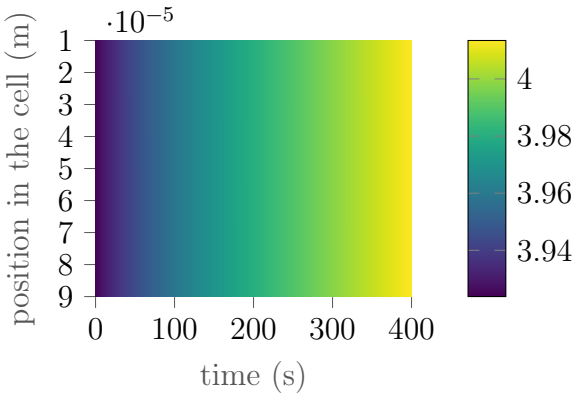



(c) Absolute error $\phi_{e\text{LS}} - \phi_{e\text{PINN}}$ of the electrolyte potential in cathode

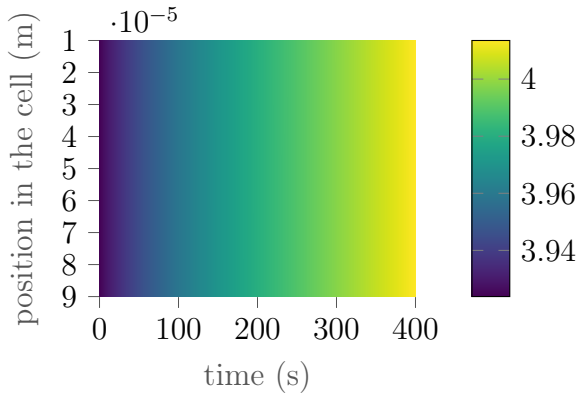



(d) Relative error $\frac{\phi_{e\text{LS}} - \phi_{e\text{PINN}}}{\phi_{e\text{LS}}}$ in % of the electrolyte potential in cathode

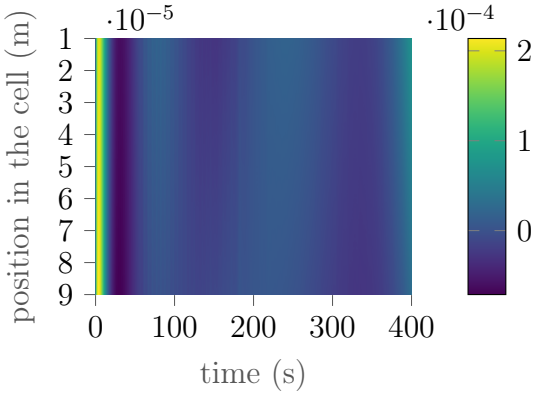
Figure 45: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of about 1%.



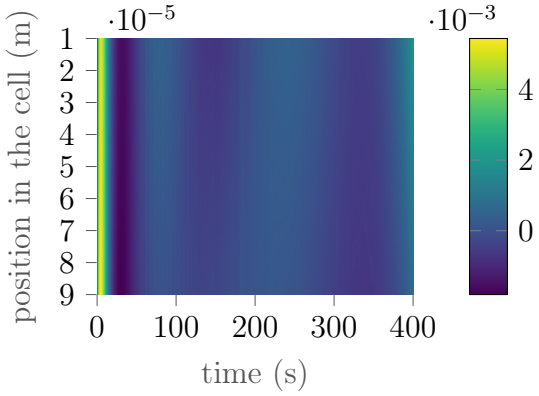

(a) Approximation of the solid potential ($\phi_s$) in cathode with a neural network




(b) Approximation of the solid potential ($\phi_s$) in cathode by LIONSIMBA



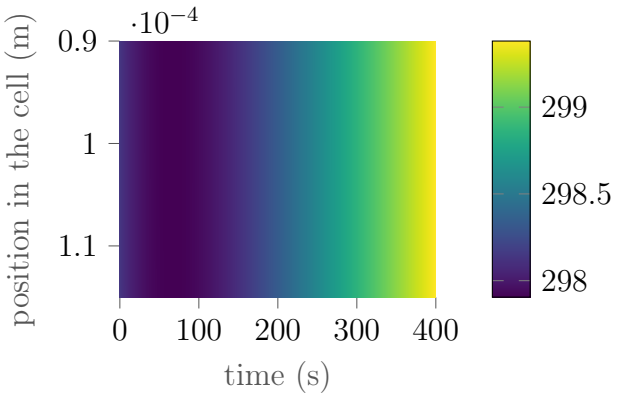

(c) Absolute error $\phi_{s\text{LS}} - \phi_{s\text{PINN}}$ of the solid potential in cathode



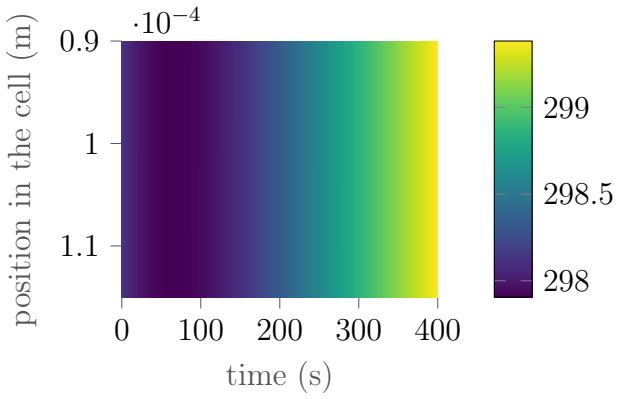

(d) Relative error $\frac{\phi_{s\text{LS}} - \phi_{s\text{PINN}}}{\phi_{s\text{LS}}}$ in % of the solid potential in cathode

Figure 46: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.005%.

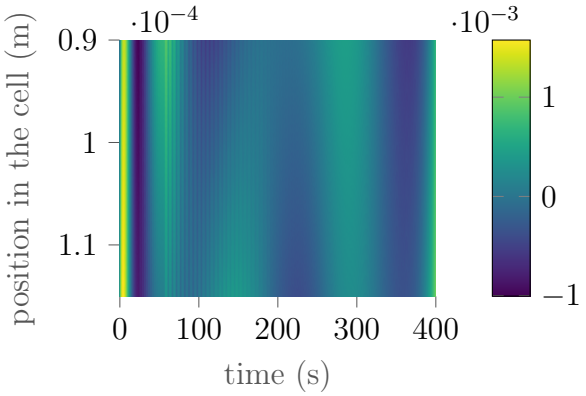
### F.3.7. Heatmaps in the separator



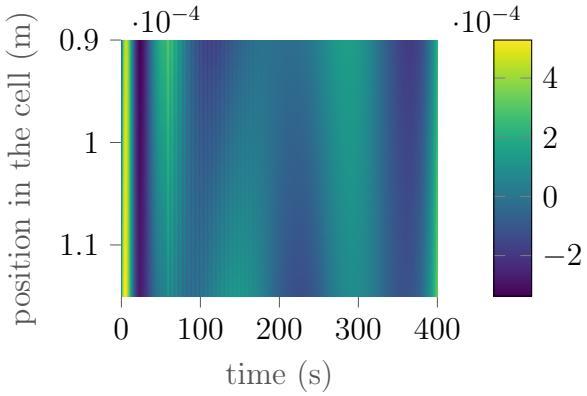

(a) Approximation of the temperature ($T$) in separator with a neural network




(b) Approximation of the temperature ($T$) in separator by LIONSIMBA

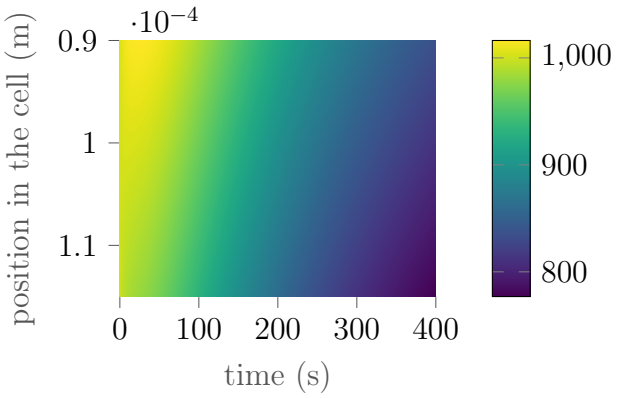



(c) Absolute error $T_{\text{LS}} - T_{\text{PINN}}$ of the temperature in separator

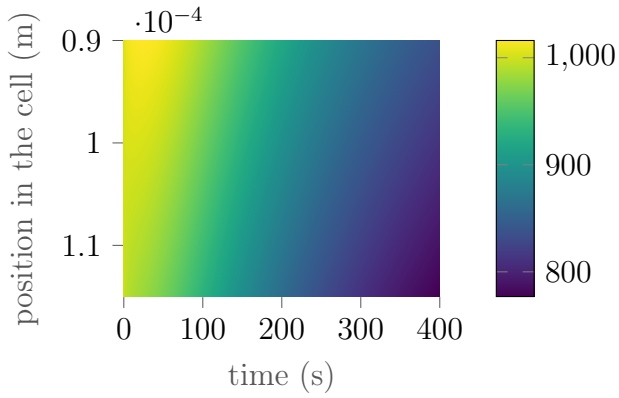



(d) Relative error $\frac{T_{\text{LS}} - T_{\text{PINN}}}{T_{\text{LS}}}$ in % of the temperature in separator

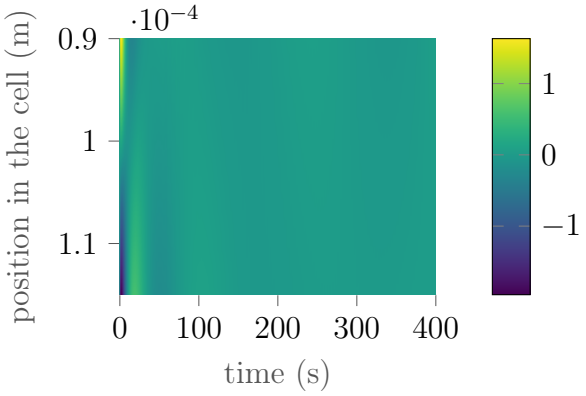
Figure 47: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.0005%.



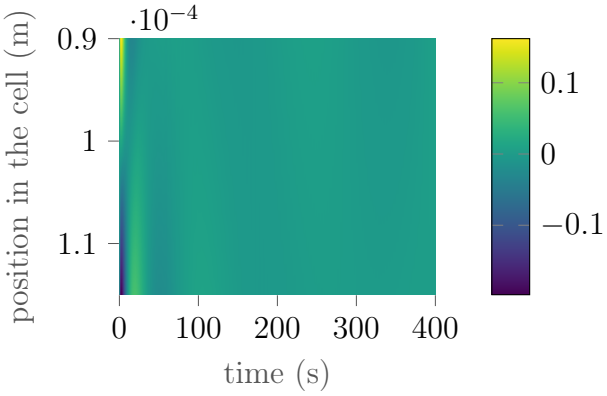

(a) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in separator with a neural network




(b) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in separator by LIONSIMBA

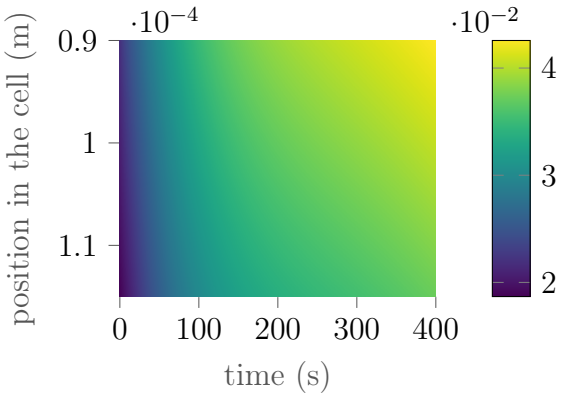



(c) Absolute error $C_{e\text{LS}} - C_{e\text{PINN}}$ of the lithium ions concentration in the electrolyte in separator

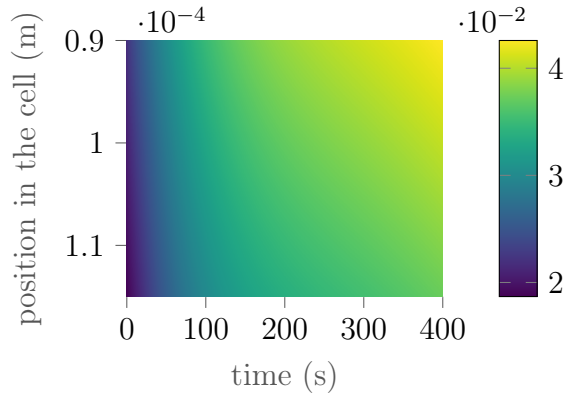



(d) Relative error $\frac{C_{e\text{LS}} - C_{e\text{PINN}}}{C_{e\text{LS}}}$ in % of the lithium ions concentration in the electrolyte in separator

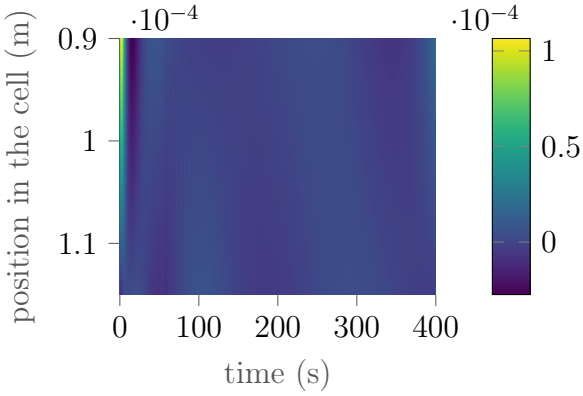
Figure 48: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of about 0.2%.



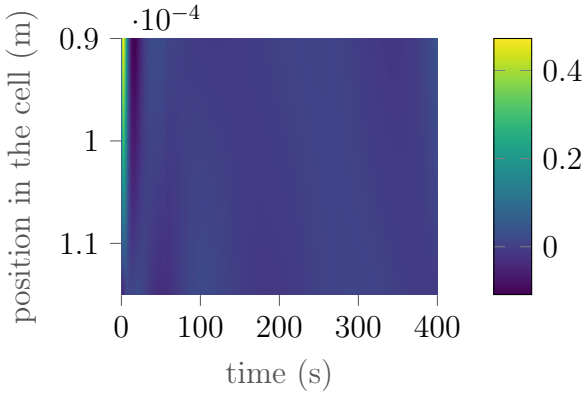

(a) Approximation of the electrolyte potential ($\phi_e$) in separator with a neural network




(b) Approximation of the electrolyte potential ($\phi_e$) in separator by LIONSIMBA



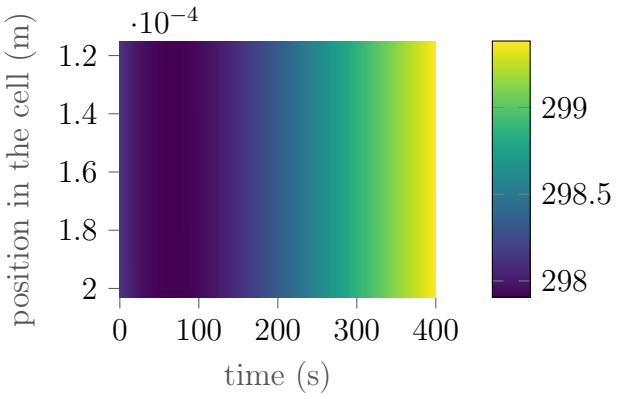

(c) Absolute error $\phi_{e\text{LS}} - \phi_{e\text{PINN}}$ of the electrolyte potential in separator



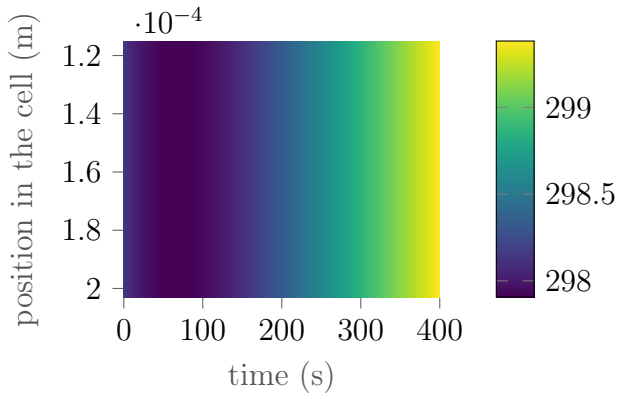

(d) Relative error $\frac{\phi_{e\text{LS}} - \phi_{e\text{PINN}}}{\phi_{e\text{LS}}}$ in % of the electrolyte potential in separator

Figure 49: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of about 0.5%.

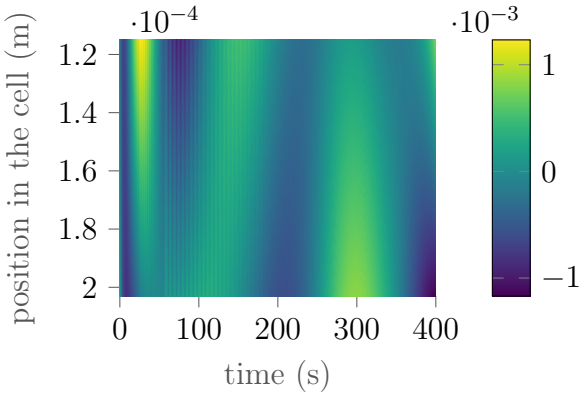
### F.3.8. Heatmaps in the anode



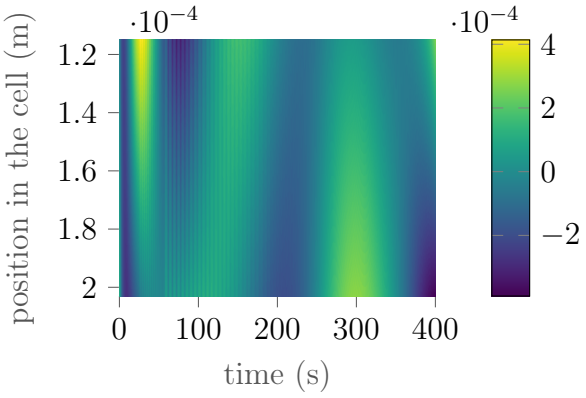

(a) Approximation of the temperature ($T$) in anode with a neural network




(b) Approximation of the temperature ($T$) in anode by LIONSIMBA

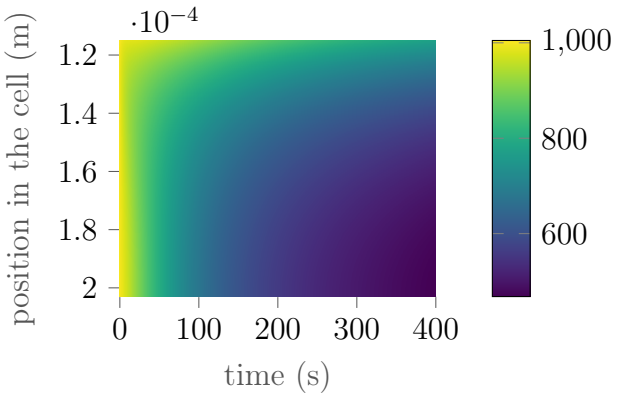



(c) Absolute error $T_{\text{LS}} - T_{\text{PINN}}$ of the temperature in anode

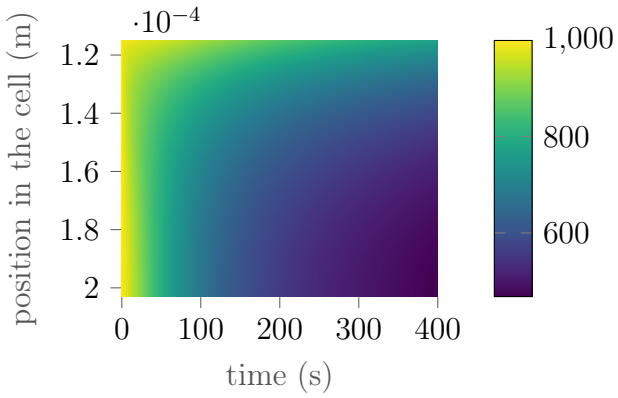



(d) Relative error $\frac{T_{\text{LS}} - T_{\text{PINN}}}{T_{\text{LS}}}$ in % of the temperature in anode

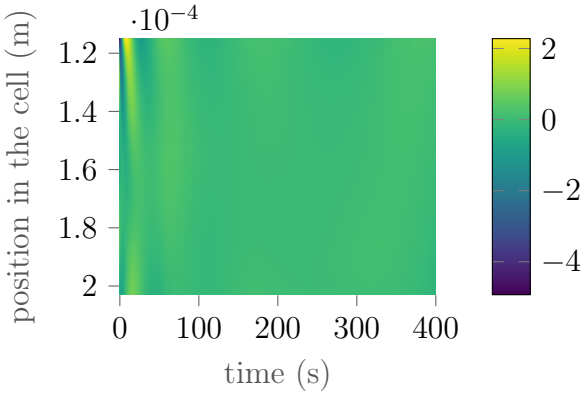
Figure 50: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.0005%.



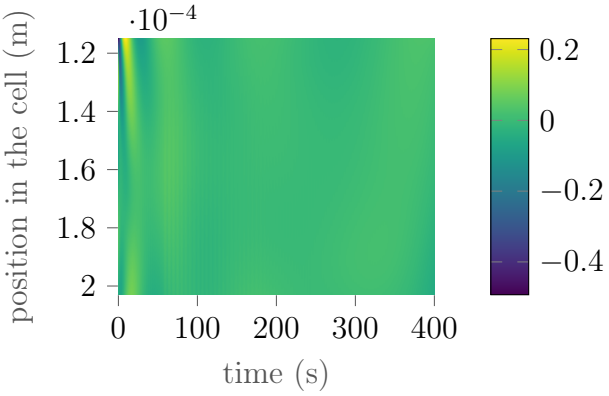

(a) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in anode with a neural network




(b) Approximation of the lithium ions concentration in the electrolyte ($C_e$) in anode by LIONSIMBA

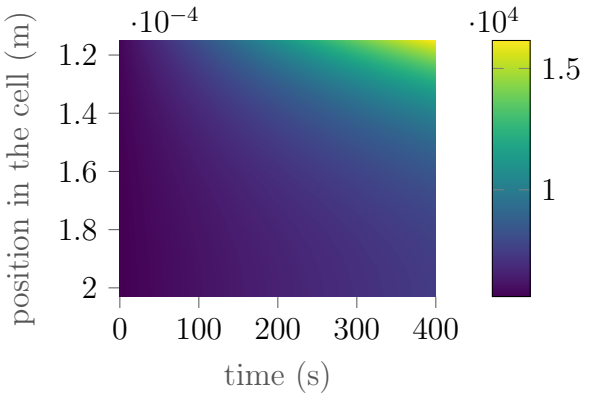



(c) Absolute error $C_{e\text{LS}} - C_{e\text{PINN}}$ of the lithium ions concentration in the electrolyte in anode

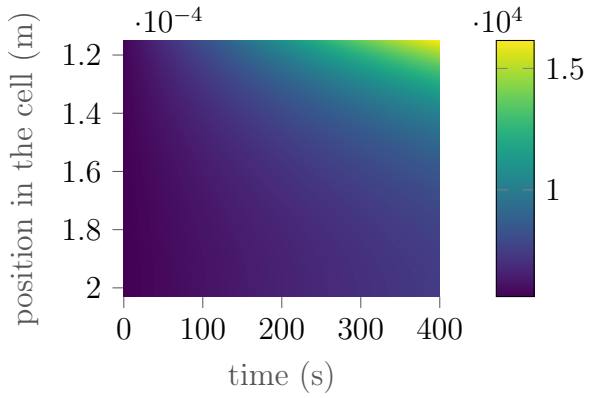



(d) Relative error $\frac{C_{e\text{LS}} - C_{e\text{PINN}}}{C_{e\text{LS}}}$ in % of the lithium ions concentration in the electrolyte in anode

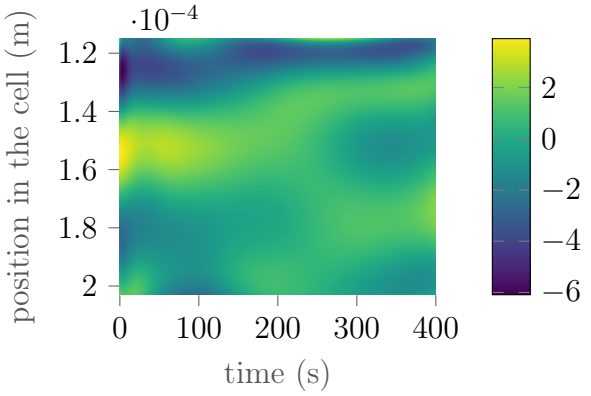
Figure 51: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.5%.



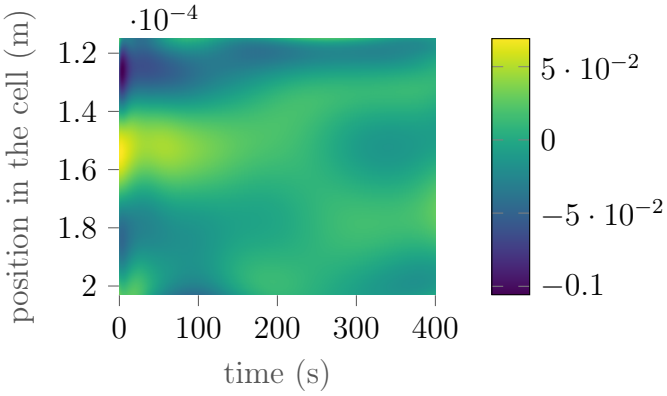

(a) Approximation of the lithium ions surface concentration in the solid-phase ($C_s^\star$) in anode with a neural network




(b) Approximation of the lithium ions surface concentration in the solid-phase ($C_s^\star$) in anode by LIONSIMBA

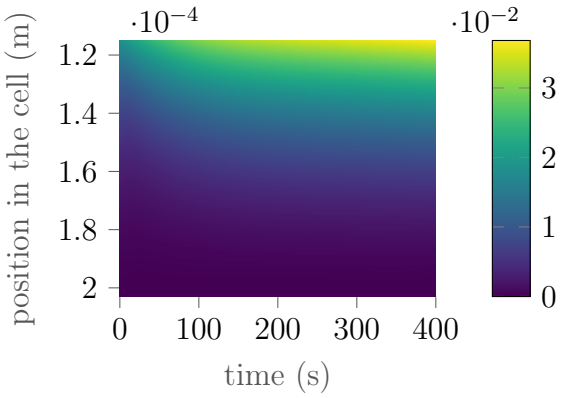



(c) Absolute error $C_{s\,\mathrm{LS}}^\star - C_{s\,\mathrm{PINN}}^\star$ of the lithium ions surface concentration in the solid-phase in anode

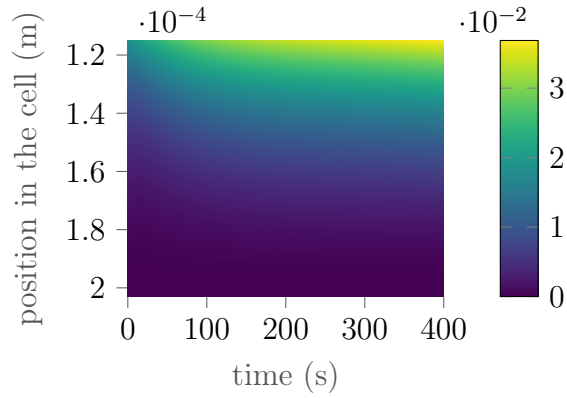



(d) Relative error $\frac{C_{s\,\mathrm{LS}}^\star - C_{s\,\mathrm{PINN}}^\star}{C_{s\,\mathrm{LS}}^\star}$ in % of the lithium ions surface concentration in the solid-phase in anode

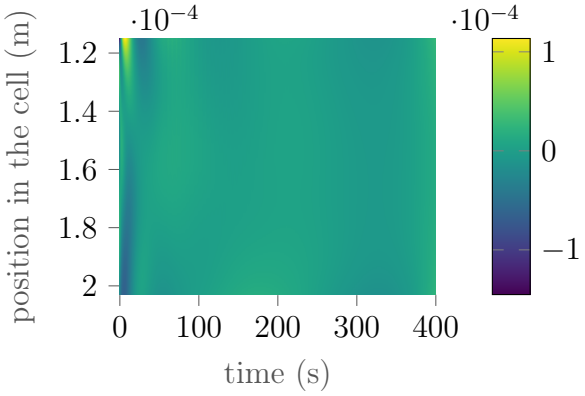
Figure 52: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\mathrm{app}} = 40\,\mathrm{A{\cdot}m^{-2}}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\mathrm{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of about 0.1%.



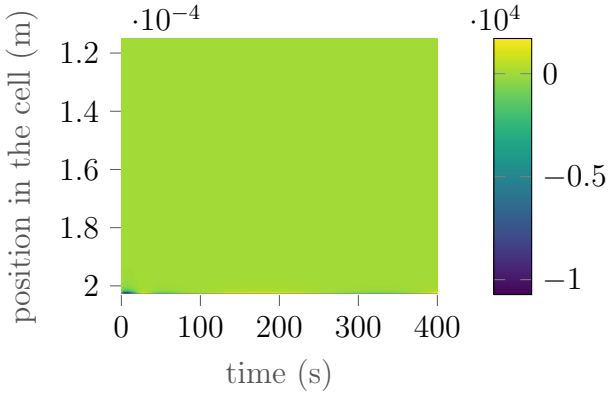

(a) Approximation of the electrolyte potential ($\phi_e$) in anode with a neural network




(b) Approximation of the electrolyte potential ($\phi_e$) in anode by LIONSIMBA

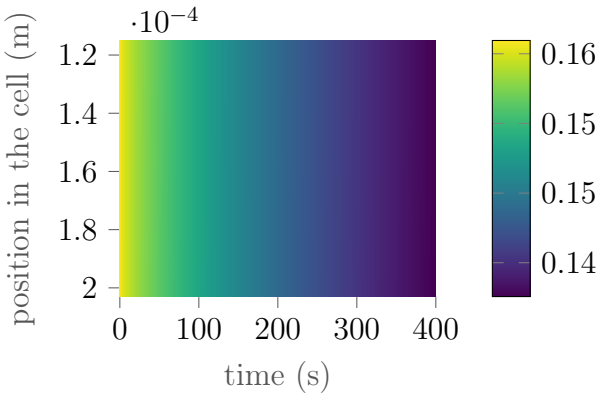



(c) Absolute error $\phi_{e\text{LS}} - \phi_{e\text{PINN}}$ of the electrolyte potential in anode

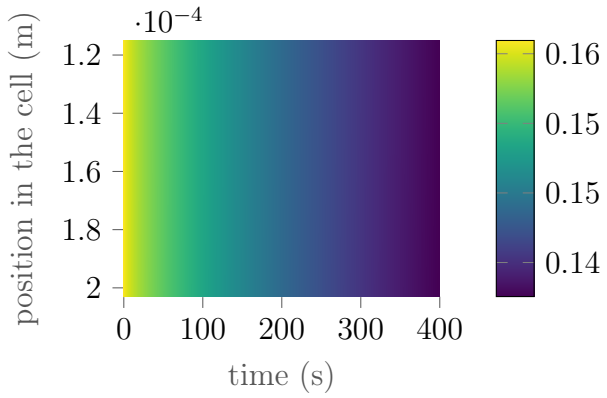



(d) Relative error $\frac{\phi_{e\text{LS}} - \phi_{e\text{PINN}}}{\phi_{e\text{LS}}}$ in % of the electrolyte potential in anode

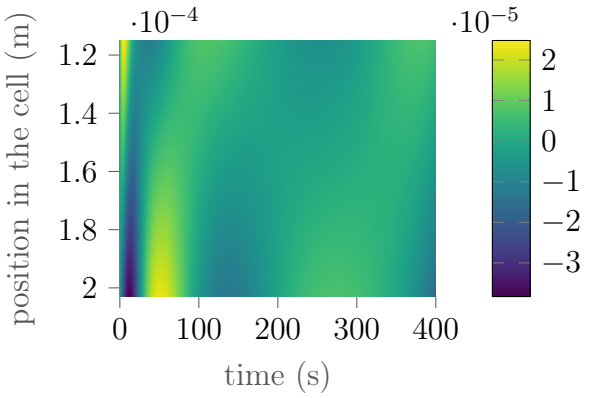
Figure 53: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
Here we see that the relative error with respect to LIONSIMBA seems very high, reaching almost $10^4$%. This is explained by the fact that $\phi_e$ is exactly equal to 0 on the end $x_n$ of the anode (here depicted in the lower part of the graph), as this is a boundary condition of the P2D model (*cf.* Equation (2.8c)). If, on the other hand, we focus on the absolute error, and compare it with the orders of magnitude of $\phi_s$ on the anode depicted in Figure 54b, we find that the effective relative error, outside a neighborhood close to $\{x_n\}[0, T_{\max}]$, is rather of the order of 1%.



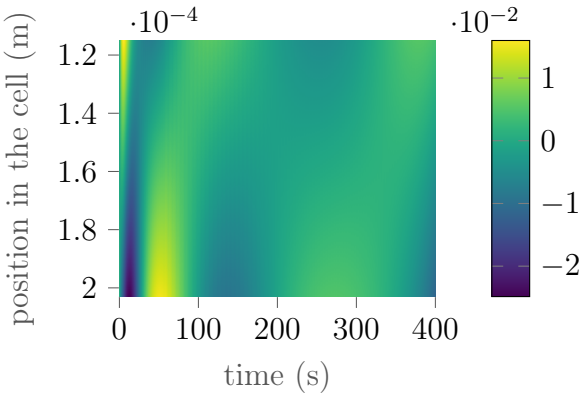

(a) Approximation of the solid potential ($\phi_s$) in anode with a neural network




(b) Approximation of the solid potential ($\phi_s$) in anode by LIONSIMBA



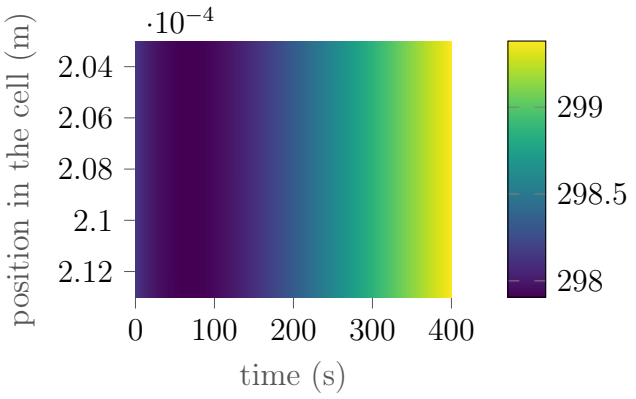

(c) Absolute error $\phi_{s\text{LS}} - \phi_{s\text{PINN}}$ of the solid potential in anode



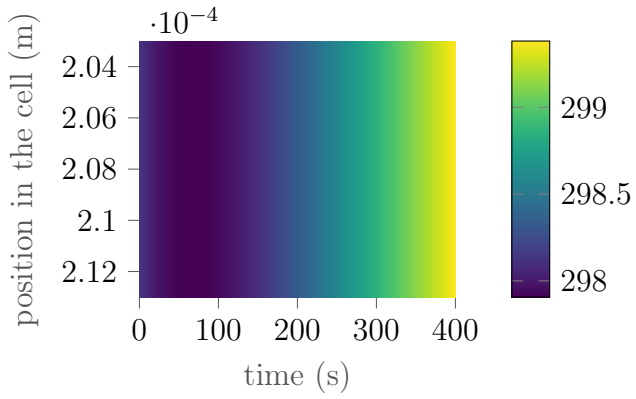

(d) Relative error $\frac{\phi_{s\text{LS}} - \phi_{s\text{PINN}}}{\phi_{s\text{LS}}}$ in % of the solid potential in anode

Figure 54: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\max} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.03%.

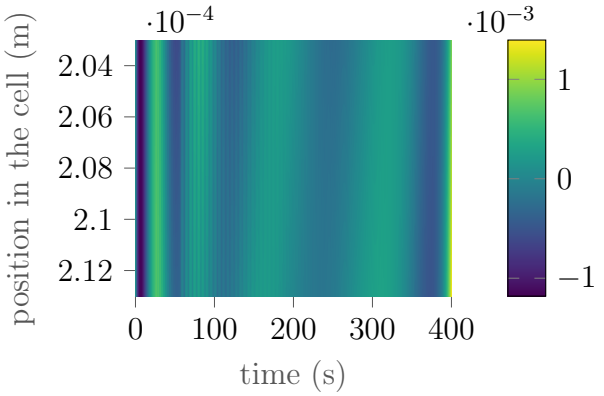
### F.3.9. Heatmaps in the negative current collector



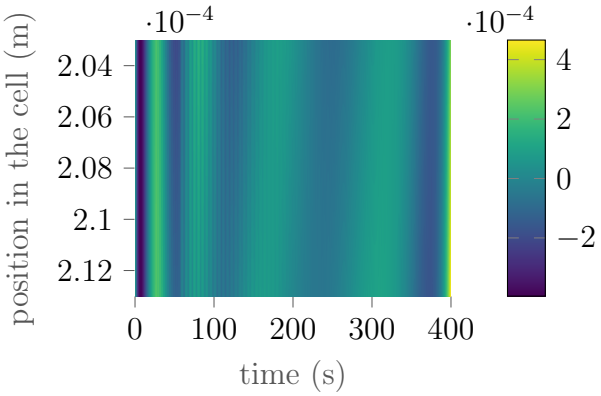

(a) Approximation of the temperature ($T$) in negative current collector with a neural network




(b) Approximation of the temperature ($T$) in negative current collector by LIONSIMBA




(c) Absolute error $T_{\text{LS}} - T_{\text{PINN}}$ of the temperature in negative current collector




(d) Relative error $\frac{T_{\text{LS}} - T_{\text{PINN}}}{T_{\text{LS}}}$ in % of the temperature in negative current collector

Figure 55: Heatmaps of the lithium ions concentration in the solid-phase in anode for the PINN approach with boundary conditions data from LIONSIMBA with self-adaptive mechanism, with a constant applied current density $I_{\text{app}} = 40\,\text{A}\cdot\text{m}^{-2}$ (*cf.* Table 7) and a simulation time $T_{\text{max}} = 400\,\text{s}$, evaluated after the final step of the gradient descent with 15000 Adam steps, followed by 1000 L-BFGS steps.
We see that the solution is very well approximated, achieving an error relative to LIONSIMBA of less than 0.0005%.

# Declaration

I certify that I have written this thesis on my own, that I have fully and accurately indicated all aids used, and that I have marked everything that has been taken from the work of others without modification or with modifications, and that I have observed the KIT Statutes for the Safeguarding of Good Scientific Practice as regularly updated.

Karlsruhe, September 8. 2023