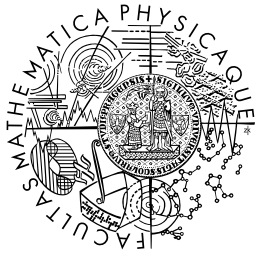


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Vojtěch Votruba

Recognition of Dissipative Systems Using Machine Learning

Mathematical Institute of Charles University

Supervisor of the bachelor thesis: doc. RNDr. Michal Pavelka, Ph.D.
Study programme: Physics

Prague 2025

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I thank my girlfriend Dorotka, my beloved family, and my friends for their undying support during my undergraduate studies. I am also indebted to my supervisor, Michal Pavelka, who graciously helped me whenever I asked.

Title: Recognition of Dissipative Systems Using Machine Learning

Author: Vojtěch Votruba

Institute: Mathematical Institute of Charles University

Supervisor: doc. RNDr. Michal Pavelka, Ph.D., Mathematical Institute of Charles University

Abstract: In this work, we design and implement neural networks specialized in modeling dissipative systems. For this purpose, we leverage a theory called generalized gradient dynamics that has the unique property of generating equations that automatically satisfy the second law of thermodynamics. Neural networks then serve as a powerful tool for reconstructing non-trivial physical models directly from experimental or simulated data. After introducing core concepts and taking inspiration from existing approaches, we introduce our method, which we successfully demonstrate by recognizing the dynamics of several well-known systems, e.g., chemical reactions. While the proposed approach can be used as an alternative to traditional numerical methods, the main future goal is to connect it with non-dissipative neural networks and apply it to the modeling of any physical, chemical, or biological systems where the exact evolution equations are unknown.

Keywords: machine learning, dissipative systems, pattern recognition

Název práce: Rozpoznávání disipativních systémů pomocí strojového učení

Autor: Vojtěch Votruba

Ústav: Matematický ústav UK

Vedoucí bakalářské práce: doc. RNDr. Michal Pavelka, Ph.D., Matematický ústav UK

Abstrakt: V této práci navrhujeme a implementujeme neuronové sítí zaměřené na modelování disipativních systémů. Za tímto účelem využíváme teorii zvanou zobecněná gradientní dynamika, jejíž unikátní vlastností je, že generuje rovnice tak, aby automaticky splňovaly druhý termodynamický zákon. Neuronové sítě pak představují mocný nástroj umožňující nalézat netriviální fyzikální modely přímo z experimentálních nebo nasimulovaných dat. Po uvedení do základních konceptů a inspiraci existujícími přístupy navrhujeme vlastní metodu, kterou úspěšně demonstrujeme na rozpoznání dynamiky několika známých systémů, například chemických reakcí. Navržený přístup lze použít jako možnou alternativu ke standardním numerickým metodám, hlavní ambicí však je ho v budoucnu propojit s nedisipativními neuronovými sítěmi a aplikovat pro modelování libovolných fyzikálních, chemických nebo biologických systému, kde přesné evoluční rovnice nejsou známy.

Klíčová slova: strojové učení, disipativní systémy, rozpoznávání vzorů

Contents

Introduction

In the year 2024, the Nobel Prize in Physics was awarded to a physicist, John J. Hopfield, and a computer scientist, Geoffrey Hinton, for *"foundational discoveries and inventions that enable machine learning with artificial neural networks"* [1]. In the same year, researchers from Google DeepMind released their machine learning models AlphaProof and AlphaGeometry2, which were able to solve problems from the International Mathematics Olympiad at a silver medalist level, a feat previously achieved only by the most promising young mathematicians [2].

It goes without saying that machine learning is spreading meteorically in today's world, becoming a ubiquitous phenomenon in science, culture, and society. However, in the public's eye, it might not be obvious that a non-trivial part of the theory behind machine learning is fundamentally connected to physics — specifically to thermodynamics and statistical mechanics, drawing upon the work of figures such as Jaynes [3] and Shannon [4].

Another area where thermodynamics is applied is the study of dissipative systems. One might even argue that studying dissipative systems is the trademark of thermodynamics: while many other fields of physics focus on conservative systems, describing them on a very high level, thermodynamics does not adopt this premise, aiming to develop a general theory of irreversibility and energy transformations.

It is then only natural that the study of dissipation and machine learning are two fields that can complement each other, with concepts like entropy being an essential component of both. In this thesis, we will attempt to very modestly deepen the connections between these two disciplines by trying to apply machine learning to modeling dissipative systems. In particular, our main goal is to create a deep learning scheme able to distill physical models from the given data, such that these models are consistent with thermodynamic laws.

Chapters 1 and 2 systematically introduce the reader to relevant theory from both non-equilibrium thermodynamics and machine learning, specifically focusing on the framework of generalized gradient dynamics and building deep neural networks.

We will then continue to review existing work in Chapter 3, covering a few state-of-the-art designs aimed towards training machine learning models to recognize and solve dissipative dynamics. At the end of this chapter, we conceptually describe our attempt at building such a machine learning model, introducing some novel elements.

In Chapter 4, we finally present the results based on applying this model to multiple known example systems, testing its capabilities, limits, and evaluating whether our design is able to learn the correct dynamics of purely dissipative systems.

1 Theory of Dissipative Systems

In this first chapter, we introduce the reader to the modern description of dissipative systems called generalized gradient dynamics. Our main reference is the book by Michal Pavelka, Václav Klika, and Miroslav Grmela [5], as it discusses many of the theoretical fundamentals relevant to this thesis.

We begin by addressing what a dissipative system is, connecting it to the concept of entropy. We will then explain to the reader how generalized gradient dynamics is specifically used as a framework in which we can write the evolution equations of these systems. Lastly, we provide a deductive argument for this theory, presenting its derivation based on stochastic processes.

1.1 Entropy & Dissipation

Let us start this chapter by defining the key concepts for our work. Firstly, we need to characterize entropy S as a function of the state of our system. Our main source [5] borrows (with a few alterations) the postulates defining entropy from the influential book by H.B. Callen [6]

1. *"There exists a function called entropy of the extensive parameters of a given system, defined for all equilibrium states and having the following property: the values assumed by the extensive parameters in the absence of an internal constraint are those that maximize the entropy over the manifold of constrained equilibrium states."*

2. *"The entropy of a composite system is additive over the constituent spatially separated subsystems. The entropy is continuous and differentiable and is a monotonically increasing function of the energy. This last property enables us to define thermodynamic temperature as* $T = \left(\frac{\partial S}{\partial E}\right)_X^{-1}$ *where* X *stands for all extensive parameters except energy* E*."*

3. *"The entropy of any system vanishes at the zero of temperature* T*."*

For our purposes, the most important postulate is the first one. Strictly speaking, we will not use the temperature or the dependence of entropy on energy at all in our thesis. We will treat entropy as a continuous function of the state variables, which attains its maximum in the equilibrium that our systems will be approaching in time.

Now that we have discussed what entropy is, we can define dissipation. Dissipative systems (or, more precisely, the dissipative evolutions of systems) will be such that they increase the total entropy in time. Typically, this system will grow its entropy until it reaches its maximum. The existence of such a maximum implies local conditions of stability — that is, at least local concavity of the entropy function. This fact will be useful to us when imposing hard constraints on our numerical models.

As a last note, we should mention that it can be further shown [5] that conservative systems satisfying Hamilton's equations of motion are non-dissipative. Thus, conservative systems can also be distinguished as those that do not change the total entropy.

1.2 Generalized Gradient Dynamics

In the previous section, we characterized dissipative systems as those driven to equilibrium by the tendency for entropy maximization.

Since the evolution equations of these systems need to "know", in a way, when exactly they should stop, it is only natural that derivatives of entropy (which equal zero when the system finally reaches its equilibrium state) are included in these equations. We will call the derivative of entropy with respect to a state variable its entropic conjugate. If we arrange all of the state variables of our system into a state vector $\boldsymbol{x}$, the vector of entropic conjugates can then be written as

$$\boldsymbol{x}^* = \frac{\delta S(\boldsymbol{x})}{\delta \boldsymbol{x}}.$$

Here, the symbol $\frac{\delta}{\delta(\cdot)}$ is used to denote the functional derivative[1]. It is worth noting that the choice of formulating the evolution equations in terms of the entropic conjugates is rooted in the long tradition of different irreversible thermodynamics theories, such as CIT (Classical Irreversible Thermodynamics) [7], EIT (Extended Irreversible Thermodynamics) [8, 7], or RIT (Rational Irreversible Thermodynamics) [9, 7]. Those theories work with entropic conjugates extensively, calling them thermodynamic forces [7].

The next step is constructing the evolution equations themselves. In generalized gradient dynamics, this is done by differentiating a quantity called the dissipation potential $\Xi(\boldsymbol{x}, \boldsymbol{x}^*)$ with respect to the entropic conjugate variables. Specifically in the form [10, 11]

$$\dot{\boldsymbol{x}} = \left.\frac{\delta \Xi(\boldsymbol{x}, \boldsymbol{x}^*)}{\delta \boldsymbol{x}^*}\right|_{\boldsymbol{x}^* = \frac{\delta S}{\delta \boldsymbol{x}}}. \tag{1.1}$$

By itself, Eq. (1.1) does not tell us much. The merits of this description arise after we impose a series of constraints on the dissipation potential. Taken from the book [5], the restrictions relevant to our work read

1. $\Xi(\boldsymbol{x}, \boldsymbol{x}^*) \geq 0$ and $\Xi(\boldsymbol{x}, \boldsymbol{0}) = 0$,
2. $\frac{\delta \Xi}{\delta \boldsymbol{x}^*} \cdot \boldsymbol{x}^* \geq 0, \quad \forall \boldsymbol{x}^*$,
3. near $\boldsymbol{x}^* = \boldsymbol{0}$ the potential Ξ must be convex.

The most important consequence of these imposed rules is the automatic fulfillment of the second law of thermodynamics. This law, in one of its formulations, states that the entropy of an isolated system is non-decreasing in time. Using the multivariable chain rule, we can write

$$\dot{S} = \frac{\delta S}{\delta \boldsymbol{x}} \cdot \dot{\boldsymbol{x}} = \boldsymbol{x}^* \cdot \frac{\delta \Xi}{\delta \boldsymbol{x}^*}$$

which will always be non-negative, according to our second constraint. Apart from this, the first constraint then effectively tells us that Ξ has a minimum at

[1] The functional derivative is an extension of the derivative to infinite functional spaces. Later, for most practical applications, we will replace this general object by a simple partial derivative $\frac{\partial}{\partial(\cdot)}$. For an introduction to functional derivatives, we refer the reader to the Appendix of [5].

$\boldsymbol{x}^* = \mathbf{0}$, this is another way of saying that evolution stops at $\delta_{\boldsymbol{x}} S = 0$ which we have touched upon earlier.

Eventually, we might ask why this framework is called generalized gradient dynamics and not just gradient dynamics. This is due to the historical development of this theory [12]. Early versions of it, which were called gradient dynamics, used only quadratic dissipation potentials with a dissipative matrix $M(\boldsymbol{x})$ in the form

$$\Xi(\boldsymbol{x}, \boldsymbol{x}^*) = \frac{1}{2}\boldsymbol{x}^* \cdot M(\boldsymbol{x}) \cdot \boldsymbol{x}^*, \tag{1.2}$$

instead of allowing for a more general structure like the one that we have described above [13, 14].

1.3 Stochastic Origins

Until now, we have spoken about gradient dynamics from a utilitarian standpoint. We have presented it as a theory that has sound properties that can be utilized to model dissipative systems. Nevertheless, we have yet to derive it from more fundamental principles. The strongest arguments for this theory, in our view, come from the study of stochastic processes, as they generally underpin many of the core concepts of thermodynamics.

Originally, the connection between stochastic processes and gradient dynamics (described with a quadratic dissipation potential as in Eq. (1.2)) was pioneered by Lars Onsager and Stefan Machlup in the 20th century [15, 16]. In 2013 and 2015, Mielke, Peletier, and Renger published two papers [17, 18], extending the original arguments to systems governed by generalized gradient dynamics. We will now define the key terms and sketch the logic of their derivation.

Path Integrals

One of the most popular ways to formulate a physics theory is through variational principles. For example, in classical mechanics, dealing with conservative systems, we can describe the entire deterministic evolution of a system through one simple equation

$$\delta\mathcal{S}[\boldsymbol{x}(t)] = 0,$$

where $\mathcal{S}$ (not to be confused with entropy S) is the so-called action functional calculated using the Lagrangian function $\mathcal{L}$ as

$$\mathcal{S}[x(t)] = \int_0^T \mathcal{L}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t))\,\mathrm{d}t.$$

This effectively tells us that with a given fixed origin and an endpoint, the system takes a trajectory between these points $\boldsymbol{x}(t)$ such that it extremizes this action $\mathcal{S}$ [19].

Drawing inspiration from classical mechanics, we can extend this description to non-deterministic systems. In these systems, extremizing action no longer identifies a unique path but rather the most probable one. Specifically, for any given path between the two fixed ends, we quantify the probability P of a system following this specific path as

$$P[\boldsymbol{x}(t)] \propto \exp(-\mathcal{S}[\boldsymbol{x}(t)]). \tag{1.3}$$

Taking the action to be dimensionless and positive.

As we can see, for the smallest action, the probability is the highest. The total probability of the system going from the point $\boldsymbol{x}_0$ to $\boldsymbol{x}$ can then be quantified by integrating all of the possible paths together (represented by the measure $\mathcal{D}[\boldsymbol{x}(t)]$), with the expression in Eq. (1.3), serving as a weight

$$P(\boldsymbol{x}_0 \to \boldsymbol{x}) \propto \int \exp(-\mathcal{S}[\boldsymbol{x}(t)])\mathcal{D}[\boldsymbol{x}(t)].$$

This type of integral is called a path integral. A similar formulation using the path integral has been employed with great success in quantum mechanics, where we refer to it as the Feynman formulation [20]. In the context of thermodynamics and stochastic processes, this description is also prevalent. For more details, we refer the reader to [15, 16, 21].

Boltzmann's Entropy

Previously, when introducing the concept of entropy in Section 1.1, we did not explain in great detail how it can be calculated. While in many cases, we can integrate entropy from the knowledge of phenomenological equations of state (e.g., for an ideal gas), generally, we have to delve into statistical mechanics to find its value. One of the most important formulae that expresses entropy in equilibrium as a function of the number of possible microstates was introduced by Boltzmann and reads

$$S = k_B \ln \Omega,$$

where Ω is the number of possible microstates of an isolated system [22, 6, 7, 5]. This number of microstates can then be further associated with the probability of a fluctuation. Realizing this link and inverting Boltzmann's equation, we then find that the equilibrium probability $P(\boldsymbol{x})$ of a system being in state $\boldsymbol{x}$ is

$$P(\boldsymbol{x}) \propto \exp(S(\boldsymbol{x})). \tag{1.4}$$

Note that one of the first uses of Eq. (1.4) came from Æ (A. Einstein) in 1902 [23, 7] — for further reading on this topic, see the last chapter of [7].

Deriving the Dissipation Potential Structure

Let us consider a system evolving stochastically in time with a reversible invariant measure — this essentially means that, for the equilibrium distribution, the probability of the system being in the first state and transitioning to the second state is equal to the probability of the system being in the second state and transitioning to the first state. We further assume that this equilibrium distribution satisfies Eq. (1.4).

Likewise, we require that the system can be described using the variational formulation similarly to Eq. (1.3); this means that there exists an action functional $\mathcal{S}$ that induces the most probable path by extremization. We next impose three additional conditions on the assigned Lagrangian function.

1. $\mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}) \geq 0$,
2. $\mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ is convex in $\boldsymbol{x}$,

3. For a trajectory $\boldsymbol{x}(t)$ such that $\mathcal{L}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t)) = 0$, it holds that $\dot{\boldsymbol{x}} = \mathcal{T}(\boldsymbol{x})$, where $\mathcal{T}$ is some function.

While the first two conditions are relatively standard [18], the third condition is non-trivial, postulating that the evolution induced by the Lagrangian corresponds to a unique first-order differential equation.

To derive generalized gradient dynamics from this, we first construct the dual Hamiltonian function H to our Lagrangian via the Legendre–Fenchel transform[2] as

$$H(\boldsymbol{x}, \boldsymbol{\xi}) = \sup_{\dot{\boldsymbol{x}}} \{\boldsymbol{\xi} \cdot \dot{\boldsymbol{x}} - \mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}})\}, \quad \text{and} \quad \mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \sup_{\boldsymbol{\xi}} \{\boldsymbol{\xi} \cdot \dot{\boldsymbol{x}} - H(\boldsymbol{x}, \boldsymbol{\xi}))\}.$$

Next, for an arbitrary vector field $\boldsymbol{\eta}(\boldsymbol{x})$, we define the potential $\Psi(\boldsymbol{x}, \boldsymbol{\xi}) := H(\boldsymbol{x}, \boldsymbol{\xi} + \boldsymbol{\eta}(\boldsymbol{x})) - H(\boldsymbol{x}, \boldsymbol{\eta}(\boldsymbol{x}))$ which we will later interpret as our dissipation potential. Taking the inverse Legendre–Fenchel transform of Ψ we arrive at

$$\Psi^*(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \sup_{\boldsymbol{\xi}} \{\boldsymbol{\xi} \cdot \dot{\boldsymbol{x}} - \Psi(\boldsymbol{x}, \boldsymbol{\xi})\} = \sup_{\boldsymbol{\xi}} \{\boldsymbol{\xi} \cdot \dot{\boldsymbol{x}} - H(\boldsymbol{x}, \boldsymbol{\xi} + \boldsymbol{\eta}(\boldsymbol{x})) + H(\boldsymbol{x}, \boldsymbol{\eta}(\boldsymbol{x}))\},$$

since we are calculating supremum, we can shift $\boldsymbol{\xi}$ by subtracting $\boldsymbol{\eta}(\boldsymbol{x})$. This transformation yields us

$$\begin{aligned}\Psi^*(\boldsymbol{x}, \dot{\boldsymbol{x}}) &= \sup_{\boldsymbol{\xi}} \{-\boldsymbol{\eta}(\boldsymbol{x}) \cdot \dot{\boldsymbol{x}} + \boldsymbol{\xi} \cdot \dot{\boldsymbol{x}} - H(\boldsymbol{x}, \boldsymbol{\xi}) + H(\boldsymbol{x}, \boldsymbol{\eta}(\boldsymbol{x}))\} = \\ &- \boldsymbol{\eta}(\boldsymbol{x}) \cdot \dot{\boldsymbol{x}} + \mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}) + H(\boldsymbol{x}, \boldsymbol{\eta}(\boldsymbol{x})) = -\boldsymbol{\eta}(\boldsymbol{x}) \cdot \dot{\boldsymbol{x}} + \mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}) - \Psi(\boldsymbol{x}, -\boldsymbol{\eta}(\boldsymbol{x})).\end{aligned}$$

In the last step we used the fact that $H(\boldsymbol{x}, \boldsymbol{0}) = 0$ — This can be easily seen from the definition of H: Since we assume a non-negative Lagrangian, for $\boldsymbol{\xi} = \boldsymbol{0}$, the supremum of $-\mathcal{L}$ must be zero. Rearranging, we finally get the expression for the Lagrangian as

$$\mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{\eta}(\boldsymbol{x}) \cdot \dot{\boldsymbol{x}} + \Psi(\boldsymbol{x}, -\boldsymbol{\eta}(\boldsymbol{x})) + \Psi^*(\boldsymbol{x}, \dot{\boldsymbol{x}}). \tag{1.5}$$

If we now consider that the most probable path extremizes the action and the conditions we imposed on the Lagrangian, it follows immediately from Eq. (1.5) that it satisfies $\mathcal{L} = 0$, which subsequently implies

$$0 = \frac{\delta}{\delta \boldsymbol{\eta}} (\boldsymbol{\eta}(\boldsymbol{x}) \cdot \dot{\boldsymbol{x}} + \Psi(\boldsymbol{x}, -\boldsymbol{\eta}(\boldsymbol{x})) + \Psi^*(\boldsymbol{x}, \dot{\boldsymbol{x}})) = \dot{\boldsymbol{x}} - \left. \frac{\delta \Psi}{\delta \boldsymbol{\eta}} \right|_{\boldsymbol{\eta} = -\boldsymbol{\eta}(\boldsymbol{x})}.$$

But if we set $\Psi \equiv \Xi$, we identify the equation for generalized gradient dynamics! It remains to justify why exactly it holds that $\boldsymbol{\eta}(\boldsymbol{x}) = -\frac{\delta S}{\delta \boldsymbol{x}}$.

Introducing Entropic Conjugates

We consider the systems in equilibrium at two points in time: $t = 0$ and $t = T$. From the reversibility condition, it then follows that the probability of the system starting at the first point $\boldsymbol{x}(0)$ and going forward to point $\boldsymbol{x}(T)$ traversing trajectory $\boldsymbol{x}_{\text{forw.}}(t)$, must be the same as when the system starts at the second

[2] Legendre–Fenchel transform is a generalization of the well-known Legendre transform, which is commonly used in classical mechanics and thermodynamics.

point and traverses the backward trajectory $\boldsymbol{x}_{\text{back.}}(t)$. We can write this equality as

$$P(\boldsymbol{x}(0))P[\boldsymbol{x}_{\text{forw.}}(t)] = P(\boldsymbol{x}(T))P[\boldsymbol{x}_{\text{back.}}(t)].$$

Substituting from Eqs. (1.3), and (1.4), we obtain

$$\begin{aligned}
\exp(S(\boldsymbol{x}(0))\exp(-\mathcal{S}[\boldsymbol{x}_{\text{forw.}}(t)]) &= \exp(S(\boldsymbol{x}(T))\exp(-\mathcal{S}[\boldsymbol{x}_{\text{back.}}(t)]),\\
S(\boldsymbol{x}(0)) - \int_0^T \mathcal{L}(\boldsymbol{x},\dot{\boldsymbol{x}})\,\mathrm{d}t &= S(\boldsymbol{x}(T)) - \int_0^T \mathcal{L}(\boldsymbol{x},-\dot{\boldsymbol{x}})\,\mathrm{d}t,\\
S(\boldsymbol{x}(0)) - S(\boldsymbol{x}(T)) &= \int_0^T \mathcal{L}(\boldsymbol{x},\dot{\boldsymbol{x}}) - \mathcal{L}(\boldsymbol{x},-\dot{\boldsymbol{x}})\,\mathrm{d}t.
\end{aligned}$$

Subsequently, in the limit $T \to 0$, it follows that

$$-\dot{S}(\boldsymbol{x}) = -\frac{\delta S}{\delta \boldsymbol{x}} \cdot \dot{\boldsymbol{x}} = \mathcal{L}(\boldsymbol{x},\dot{\boldsymbol{x}}) - \mathcal{L}(\boldsymbol{x},-\dot{\boldsymbol{x}}),$$

and using the Eq. (1.5) finally yields

$$-\frac{\delta S}{\delta \boldsymbol{x}} \cdot \dot{\boldsymbol{x}} = 2\boldsymbol{\eta}(\boldsymbol{x}) \cdot \dot{\boldsymbol{x}} + \cancel{\Psi^*(\boldsymbol{x},\dot{\boldsymbol{x}}) - \Psi^*(\boldsymbol{x},-\dot{\boldsymbol{x}})}.$$

Where the conjugate potentials cancel out due to their symmetry[3], producing the final result we were trying to derive. The factor 2 can then be eliminated by simply setting $\Xi(\boldsymbol{x},\boldsymbol{x}^*) = \Psi(\boldsymbol{x},\boldsymbol{x}^*/2)$.

[3]Technically, this is another assumption, not a proven fact. However, it is reasonable to adopt this premise, as it corresponds to a symmetric dissipation potential $\Xi(\boldsymbol{x},\cdot)$, which we are usually looking for.

2 Deep Learning Fundamentals

As we have mentioned in the Introduction, machine learning has experienced a meteoric rise in recent years. The subfield of machine learning that will be our focus is known as deep learning and is concerned with building and training so-called neural networks. Specifically, we employ *deep feedforward neural networks*, as is explained in Section 2.1.

After outlining how deep learning works in principle and briefly discussing neural networks' representational power, we shift our focus to their training techniques. In this context, we present an overview of first- and second-order optimization algorithms using the book [24] as a primary reference.

2.1 Terminology

For this thesis, we consider a *neural network* to be a composite function with many parameters, collectively denoted $\boldsymbol{\theta}$. These parameters should then be tuned to approximate a mapping between the given inputs and outputs. The inputs, hereafter referred to as *data*, are typically given as a collection of N-dimensional real vectors. Similarly, the outputs are a collection of M-dimensional vectors and will be called *targets*. In the literature, the composite function that we will talk about is customarily called a *feedforward* neural network [24]. However, we will mostly only use the expression neural network for brevity.

Equipped with these terms, we can define feedforward neural networks more rigorously. Specifically, it is to be a function $\boldsymbol{\Lambda} : \mathbb{R}^N \to \mathbb{R}^M$ such that

$$\boldsymbol{\Lambda}(\boldsymbol{x}; \boldsymbol{\theta}) := \boldsymbol{\varphi}_n(W_n \cdot (\boldsymbol{\varphi}_{n-1}(...\boldsymbol{\varphi}_1(W_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)) + \boldsymbol{b}_n), \tag{2.1}$$

where W_i are real-valued matrices (often called tensors in the deep learning nomenclature [24]) with dimensions chosen to match the input and output sizes. We refer to these matrices as *weight tensors*, with their elements called *weights*. The real-valued vectors $\boldsymbol{b}_i$ will be named *biases*, and together with the weights, they define the parameter vector $\boldsymbol{\theta}$.

Furthermore, the functions $\boldsymbol{\varphi}_i$ are called *activation functions*. Their primary role is to introduce non-linearity to the network. If we look at Eq. (2.1), we can immediately see that without these functions, $\boldsymbol{\Lambda}$ would merely be an affine map, as it would have been a composition of simple affine transforms. Common choices for these functions include the element-wise ReLU (*rectified linear unit*) function or its smooth alternative, the element-wise softplus function [24]

$$\mathrm{ReLU}(z) = \begin{cases} z, & z \geq 0, \\ 0, & z < 0, \end{cases} \qquad \mathrm{softplus}(z) = \log(1 + e^z).$$

Lastly, we define *depth* of a network as the number n corresponding to Eq. (2.1) above. Indexing by $i = 1, \ldots, n$, we can partition the network into different layers with the outermost layer commonly called the *output layer*. The layers between the output layer and the input vector $\boldsymbol{x}$ are referred to as *hidden layers*.

Neural networks are customarily visualized by drawing schemes of them, such as the one shown in Fig. 2.1. Additionally, for each of the layers of a neural

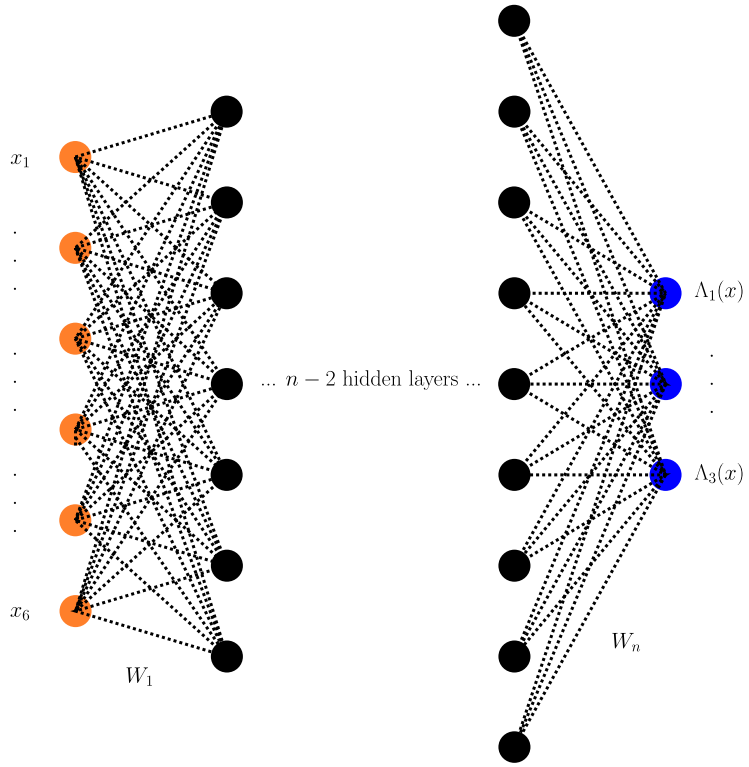


Figure 2.1 An example diagram of a neural network

network, we define its width as the number of output units (or neurons) it contains, corresponding to the output dimension of the associated weight tensor.

Although some of the architectures used in the next chapters are not, strictly speaking, feedforward, and therefore would not fit our definition of a neural network, we find it useful to think about neural networks in this way, as it makes most of the concepts discussed in this chapter more approachable.

2.2 Representational power

Now that we have established the basic terminology, we will comment on the motivation behind using neural networks to approximate the mappings between data and targets in the first place. To this end, we formulate (without proof) one of the *universal approximation theorems*, as proven in [25].

Theorem 1. *Let $\Phi \in L^{\infty}_{\text{loc.}}(\mathbb{R})$ be a function that is continuous almost everywhere with respect to the Lebesgue measure. Set*

$$\Sigma_n = \text{span}\{\Phi(\boldsymbol{w} \cdot \boldsymbol{x} + b) : \boldsymbol{w} \in \mathbb{R}^N, b \in \mathbb{R}\}.$$

Then Σ_n is dense in $C(\mathbb{R})$ if and only if Φ is not an algebraic polynomial.

Informally, this theorem suggests that a given neural network with a single hidden layer, using a non-polynomial activation function Φ that is continuous almost everywhere, can approximate any real continuous function with any desired

precision, provided that this hidden layer can have arbitrary width. The activation function on the output layer, in this case, is taken as the identity.

Since Theorem 1 uses an arbitrary width of the hidden layer, it does not directly guarantee any practical results. Nevertheless, it serves as an indication that neural networks can, at least theoretically, be useful function approximators. Moreover, other versions of it have been proven using an arbitrary depth of the network rather than the arbitrary width [24].

2.3 Training Neural Networks

Training a neural network typically refers to the process of tuning its parameters to approximate the mapping between the provided data and targets as accurately as possible.

Mathematically, it corresponds to an optimization problem. Firstly, we choose a *loss function* $L : \mathbb{R}^M \to [0, +\infty)$, which represents the distance between the target, denoted by $\boldsymbol{y}$, and the network's prediction based on the provided input data. The training then amounts to finding $\boldsymbol{\theta}_{\text{optim.}}$ such that

$$\boldsymbol{\theta}_{\text{optim.}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{y} - \boldsymbol{\Lambda}(\boldsymbol{x}; \boldsymbol{\theta})).$$

Common choices for these loss functions include L^1 and L^2 norms [24]. These norms are often divided by the input's dimension. For example, for the L^2 norm, this takes the form

$$L(\boldsymbol{u}) = \frac{1}{M}||\boldsymbol{u}||^2, \quad \boldsymbol{u} \in \mathbb{R}^M. \tag{2.2}$$

Assuming $\boldsymbol{u}$ to represent a difference between some prediction and the true output, Eq. (2.2) produces the mean squared error (MSE), as it is known in statistics.

Now that we have formulated our optimization problem, we turn to different practical methods to solve it numerically. In the context of neural networks, the algorithms implementing such methods are referred to as *optimizers*. We will briefly explore the two most prevalent categories, first-order methods and second-order methods, as they will be used in the specific implementation of our design in later chapters.

2.3.1 First-Order Methods

The fundamental idea behind both the first- and second-order methods is to solve the given optimization problem iteratively. If we denote the dimension of the parameter vector $\boldsymbol{\theta}$ as d, we aim to construct such a sequence in $\mathbb{R}^d$ that converges to $\boldsymbol{\theta}_{\text{optim.}}$. Most first-order methods used in deep learning are based on the algorithm called gradient descent, which we will now derive.

Let us start our sequence at a randomly selected point $\boldsymbol{\theta}_0$. From elementary calculus, we know that for the function L in d-dimensional space, the directional derivative $\nabla_{\boldsymbol{n}}$ (which quantifies the steepness along the direction $\boldsymbol{n}$) can be calculated as

$$\nabla_{\boldsymbol{n}} L = \nabla L \cdot \boldsymbol{n}, \tag{2.3}$$

where we consider gradient ∇ with respect to $\boldsymbol{\theta}$. We now aim to find a direction of steepest descent to construct a new value $\boldsymbol{\theta}_1$. Here, by direction of steepest

descent, we mean a vector $\boldsymbol{n}$ such that the loss $L(\boldsymbol{\theta}_1)$ is as small as possible. Applying the well-known Cauchy-Schwarz inequality to the dot product on the right-hand side of Eq. (2.3) yields

$$\nabla L \cdot \boldsymbol{n} \leq ||\nabla L||||\boldsymbol{n}||,$$

Over the years, many different improvements to this gradient descent algorithm have been developed, e.g., SGD [26], Adagrad [27], and Adam [28]. All of these algorithms rely on numerical differentiation to compute the gradient, which is usually done using the so-called automatic differentiation techniques. Furthermore, these algorithms employ techniques such as using adaptive learning rates or introducing momentum to compute the gradients. For additional reading on these algorithms and their specifics, we recommend [24].

Finally, a topic that is closely connected to the implementation of first-order methods is *minibatching*. Typically, the computationally heaviest part of the first-order methods is numerically estimating the gradient of the given loss function. Such an operation, without the use of minibatching, would require performing a large number of calculations going over all of the data and targets — which, as previously noted, are collections of vectors rather than individual vectors. To mitigate this, in each iteration of our algorithms, we (usually randomly) sample only a smaller *batch* of the given data and targets and use them instead of the entire dataset. This reduces our computational costs and potentially eliminates redundancy, which could lead to a more desirable approximation of the real mapping [24].

2.3.2 Second-Order Methods

Second-order methods go further than first-order methods in that they use not only the first derivatives of L but also its second derivatives to take steps towards the minimum. Similarly to the previous subsection, where we derived gradient descent, we will derive the most fundamental second-order method called Newton's method for optimization.

Starting with the initial value $\boldsymbol{\theta}_0$, we want to iteratively take steps towards the minimum of the loss function L. Firstly, we write the Taylor expansion of $\nabla L(\boldsymbol{\theta}_{\text{optim.}})$ for a nearby point $\boldsymbol{\theta}$ up to the first order, producing

$$\nabla L(\boldsymbol{\theta}_{\text{optim.}}) \approx \nabla L(\boldsymbol{\theta}) + \nabla^2 L(\boldsymbol{\theta}) \cdot (\boldsymbol{\theta}_{\text{optim.}} - \boldsymbol{\theta}),$$

where $\nabla^2 L_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j}$ is the Hessian matrix of L. Since we also know that the gradient of L in the minima must be zero, we omit the term $\nabla L(\boldsymbol{\theta}_{\text{optim.}})$. By rearranging, we then gain

$$\boldsymbol{\theta}_{\text{optim.}} \approx \boldsymbol{\theta} - (\nabla^2 L)^{-1}(\boldsymbol{\theta}) \cdot \nabla L(\boldsymbol{\theta}),$$

which we will use as a blueprint to construct our sequence as

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - (\nabla^2 L)^{-1}(\boldsymbol{\theta}_i) \cdot \nabla L(\boldsymbol{\theta}_i).$$

By calculating the Hessian, we expect to preserve more information than with the first-order methods, which should lead to faster convergence to the minima. The

numerical analysis of Newton's method supports this; however, there are a few caveats.

Firstly, the Newton method, in this formulation, has problems with getting stuck at saddle points. Secondly, the computation of the inverse of $\nabla^2 L$ is extremely costly compared to just calculating the gradient [24]. While the former of these problems can be somewhat resolved by regularization, the second one is more complicated. Thus, in practice, we prefer to use more advanced second-order methods, implementing different ways to estimate $\nabla^2 L$. Examples of these methods include the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [29, 30, 31, 32], or its limited memory version, abbreviated as L-BFGS [33].

The last remark we should make is about batching in the context of second-order methods. According to [24], while first-order methods can be used with relatively small batch sizes successfully, second-order methods often require larger ones to eliminate fluctuations in the estimates of the Hessian matrix. For this reason, when using second-order methods in the next chapters, we will stick to full-batch optimization instead of utilizing minibatches.

3 Dissipative Neural Networks

This chapter covers the first of the final two parts that form the core of this thesis: the development and testing of a model connecting generalized gradient dynamics with deep neural networks.

As we have explained, generalized gradient dynamics is a framework that can (in theory) be used to describe all dissipative systems. Developing a general deep learning model using this approach opens the doors to many potential applications. For example, we could use it by itself, modeling purely dissipative systems or uncovering the constitutive relations that are essential in continuum mechanics. Another exciting ambition is that, in future work, we can merge it with another model, constructing a coupling that describes systems with both dissipative and non-dissipative dynamics.

We outline the content as follows: In Section 3.1, following the review article [12] by Cueto and Chinesta, we discuss multiple conceived deep learning designs that have attempted to model dissipative phenomena. In Section 3.2, we present our proposed strategy, which is inspired by these listed designs, tweaking their elements, and adding some novel modifications.

3.1 Existing Architectures

We begin our discussion about existing architectures by introducing the general idea of Physics-Informed Neural Networks [34, 12], which are used not only regarding dissipation but across many areas of physics. We will then continue by presenting two more specific architectures: Metriplectic [12] and Variational Onsager Neural Networks [35, 12], targeted specifically at modeling thermodynamic descriptions.

3.1.1 Physics-Informed Neural Networks

The term Physics-Informed Neural Networks (abbreviated as PINNs) was coined by M. Raissi, P. Perdikaris, and G.E. Karniadakis in 2018 [34]. The main idea behind them is this: Many physical systems are governed by some set of partial differential equations (PDEs). Since deep learning is fundamentally an optimization method, we can train a neural network to find a function satisfying these PDEs by setting the loss function to account for the difference between the left-hand side of the equation and the right-hand side of the equation. Typically, the loss function L is then further extended with a term accounting for the enforcement of boundary conditions [34, 12]; this can be formally written as

$$L = L_{\text{PDE}} + L_{\text{BC}}.$$

Such an approach is different from traditional neural networks in that it enforces the fulfillment of physical laws in a stronger way. Additionally, PINNs typically need much less data than standard neural networks to produce accurate results [34]. For example, to illustrate their principle, when studying Hamiltonian dynamics, we might not want to teach the neural network the mapping $(\boldsymbol{p}, \boldsymbol{q}) \to (\dot{\boldsymbol{p}}, \dot{\boldsymbol{q}})$ by itself, but instead we can leverage a physics-informed neural network to learn the

Hamiltonian function $H(\boldsymbol{p}, \boldsymbol{q})$. We could then use the already provided automatic differentiation engine (which is required due to the training phase) to compute the pair $(\dot{\boldsymbol{p}}, \dot{\boldsymbol{q}})$ using Hamilton's equations, thus enforcing the preservation of the symplectic structure and therefore energy. This specific design is called the Hamiltonian Neural Networks [36, 12].

In recent years, PINNs have become quite popular, with researchers actively applying them to various physics problems, as well as being a subject of investigations on their own [12].

3.1.2 Metriplectic Neural Networks

One of the modern attempts to unite the descriptions of dissipative and conservative systems is called GENERIC, which stands for *General Equation for Non-Equilibrium Reversible-Irreversible Coupling* [5, 10, 11]. This approach integrates Hamiltonian dynamics given by the Poisson bracket $\{\cdot, \cdot\}$ and Hamiltonian function H with generalized gradient dynamics, producing the central equation

$$\dot{\boldsymbol{x}} = \{\boldsymbol{x}, H\} + \left.\frac{\delta \Xi}{\delta \boldsymbol{x}^*}\right|_{\boldsymbol{x}^* = \frac{\delta S}{\delta \boldsymbol{x}}}. \tag{3.1}$$

As mentioned in Chapter 1, the original description of gradient dynamics only used quadratic dissipation potentials. If we apply this consideration, Eq. (3.1) transforms into

$$\dot{\boldsymbol{x}} = \{\boldsymbol{x}, H\} + M(\boldsymbol{x}) \cdot \frac{\delta S}{\delta \boldsymbol{x}}. \tag{3.2}$$

Furthermore, in differential geometry, Hamilton's equations of motion can be reformulated in a coordinate-free language on a topological space called the *symplectic* manifold. In contrast, gradient dynamics can be built on a manifold with a *Riemannian metric*. This is the reason why GENERIC is commonly referred to as a *metriplectic* description [5, 10, 11].

Metriplectic Neural Networks (abbreviated as MNNs) can then be defined by Eq. (3.2) where they try to learn the functions $H(\boldsymbol{x})$, $M(\boldsymbol{x})$, $S(\boldsymbol{x})$, and the corresponding Poisson bracket. This approach is similar to Hamiltonian Neural Networks discussed before. Instead of learning the direct mapping $\boldsymbol{x} \to \dot{\boldsymbol{x}}$, the model learns to calculate these thermodynamic quantities instead. The physics-informed loss can then be written as

$$L \propto ||\dot{\boldsymbol{x}}^{(\text{nn.})} - \dot{\boldsymbol{x}}^{(\text{true})}||^2 = \left|\left|\{\boldsymbol{x}, H^{(\text{nn.})}\}^{(\text{nn.})} + M^{(\text{nn.})}(\boldsymbol{x}) \frac{\partial S^{(\text{nn.})}}{\partial \boldsymbol{x}} - \dot{\boldsymbol{x}}^{(\text{true})}\right|\right|^2,$$

where the upper index distinguishes between the neural network's prediction and the ground truth data [12]. Typically, MNNs also add terms to the loss function that enforce the distinction between the two evolutions (one being dissipative and the other one being conservative, not producing entropy) [12]; however, we will not explore them in detail, as it is not relevant to the purpose of modeling solely dissipative phenomena.

3.1.3 Variational Onsager Neural Networks

Variational Onsager Neural Networks (abbreviated as VONNs) were introduced in a 2021 paper by Celia Reina et al. [35]. Their original design was based on

Onsager's Variational Principle [37, 38, 15], which we briefly derive from generalized gradient dynamics.

The conjugate dissipation potential $\Xi^*(\boldsymbol{x}, \dot{\boldsymbol{x}})$ constructed via Legendre's transform takes the form:

$$\Xi^*(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \dot{\boldsymbol{x}} \cdot \boldsymbol{x}^* - \Xi(\boldsymbol{x}, \boldsymbol{x}^*).$$

Differentiating both sides of this equation with respect to $\dot{\boldsymbol{x}}$ immediately yields

$$\frac{\delta \Xi^*}{\delta \dot{\boldsymbol{x}}} = \boldsymbol{x}^* = \frac{\delta S}{\delta \boldsymbol{x}} \implies \frac{\delta \Xi^*}{\delta \dot{\boldsymbol{x}}} - \frac{\delta S}{\delta \boldsymbol{x}} = 0. \tag{3.3}$$

This resulting Eq. (3.3) can be thought of as one of the formulations of Onsager's Principle [1]. The authors of VONNs have employed this idea in the energetic representation, leveraging the free energy functional $F(\boldsymbol{x})$ instead of our entropic description using $S(\boldsymbol{x})$ [35, 12]. Nonetheless, for isothermal systems, these two descriptions are equivalent up to a sign [5].

Equipped with Eq. (3.3), its fulfillment can be enforced by implementing the typical physics-informed loss function in the form:

$$L \propto \left|\left| \frac{\partial \Xi^{*(\text{nn.})}}{\partial \dot{\boldsymbol{x}}} - \frac{\partial S^{(\text{nn.})}}{\partial \boldsymbol{x}} \right|\right|^2.$$

Next, suppose we formulate the balance equations — e.g., the conservation of energy and momenta — in terms of S and Ξ^*. By adding the corresponding terms to our loss function, the model is pushed to learn both entropy and conjugate dissipation potential in such a way that the learned dynamic satisfies both the ground truth evolution of the system (due to the balance equations), as well as the second law of thermodynamics (due to Onsager's principle). This is the fundamental idea behind VONNs [35, 12]. In practice, this architecture is implemented by splitting the model into two neural networks: one learning S and the other Ξ^* [35].

However, for Onsager's principle to imply the second law of thermodynamics, three restrictions on Ξ, which we have discussed in Chapter 1, must hold for Ξ^* as well [35]. This is a crucial part of the entire design.

First, addressing the convexity: VONNs extend this requirement from the neighborhood of $\boldsymbol{x}^* = \boldsymbol{0}$ to the entire domain [35]. This is done by implementing the function Ξ^* as a Partially Input Convex Neural Network [39, 35, 12] — an architecture we will explain more thoroughly in the next section.

Second, the requirement of minima in $\boldsymbol{x}^* = \boldsymbol{0}$ is satisfied by the reparameterization of the output. Specifically, if we denote the output of the network corresponding to the dissipation potential as $\Psi^*(\boldsymbol{x}, \dot{\boldsymbol{x}})$, the "true" conjugate dissipation potential $\Xi^*(\boldsymbol{x}, \dot{\boldsymbol{x}})$ can then be recovered as [35]

$$\Xi^*(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \Psi^*(\boldsymbol{x}, \dot{\boldsymbol{x}}) - \Psi^*(\boldsymbol{x}, 0) - \dot{\boldsymbol{x}} \cdot \left.\frac{\partial \Psi^*}{\partial \dot{\boldsymbol{x}}}\right|_{\dot{\boldsymbol{x}}=\boldsymbol{0}} \tag{3.4}$$

which is essentially an affine transform. By plugging $\dot{\boldsymbol{x}} = 0$ into Eq. (3.4) one can easily verify that $\Xi^*(\boldsymbol{x}, \boldsymbol{0}) = 0$, and by differentiating it also becomes clear that $\frac{\partial \Xi^*}{\partial \dot{\boldsymbol{x}}}(\boldsymbol{x}, \boldsymbol{0}) = \boldsymbol{0}$.

[1] Typically, this equation arises from minimization of the Rayleighan functional, which is why the principle is of variational nature [37, 38, 15].

Finally, the condition $\frac{\partial \Xi^*}{\partial \dot{\boldsymbol{x}}} \cdot \dot{\boldsymbol{x}} \geq 0$ is just an implication of the two constraints described above. Using one of the formulations of convexity [40], it must hold that

$$\Xi^*(\boldsymbol{x}, \boldsymbol{y}) \geq \Xi^*(\boldsymbol{x}, \boldsymbol{z}) + (\boldsymbol{y} - \boldsymbol{z}) \cdot \left. \frac{\partial \Xi^*}{\partial \dot{\boldsymbol{x}}} \right|_{\dot{\boldsymbol{x}}=\boldsymbol{z}}.$$

Setting $\boldsymbol{y} = 0$ and using $\Xi^*(\boldsymbol{x}, \boldsymbol{0}) = 0$, we arrive at

$$\boldsymbol{z} \cdot \left. \frac{\partial \Xi^*}{\partial \dot{\boldsymbol{x}}} \right|_{\dot{\boldsymbol{x}}=\boldsymbol{z}} \geq \Xi^*(\boldsymbol{x}, \boldsymbol{z}),$$

and since $\Xi^*(\boldsymbol{x}, \boldsymbol{z}) \geq 0$, we immediately obtain the results we have been looking for.

3.2 Proposed Approach

Note on concurrent work: In January 2025, Weilun Qiu, Shenglin Huang, and Celia Reina released a preprint [41] expanding on the original architecture of VONNs [35]. Rather than directly using Onsager's Variational Principle, the authors construct evolution equations in a way equivalent to ours. This creates a more significant overlap between our work and [41]. Nonetheless, our approach relies on a different design of the entropic network, in contrast to their free energy network. Additionally, we primarily study different example systems; our training strategy varies from theirs, and by using the entropy functional, our work better generalizes the framework to non-isothermal GENERIC systems.

Let us start by declaring our goals: Given a discretized trajectory in the state space of our dissipative system parametrized by physical time t, we want to train our model to predict the next state of the system $\boldsymbol{x}_{j+1} := \boldsymbol{x}(t + \Delta t)$ solely from the current state $\boldsymbol{x}_j := \boldsymbol{x}(t)$. In other words, we want it to reconstruct the entire trajectory of the system as accurately as possible just from the initial conditions. Furthermore, we require our model to be thermodynamically valid, constructing this trajectory using the right quantities that satisfy the second law of thermodynamics.

As discussed above, generalized gradient dynamics provides us with a framework through which we can calculate $\dot{\boldsymbol{x}}(t)$ while maintaining the desired thermodynamic consistency. Since the differential equations produced are of the first order in time, we can then easily utilize numerical integration schemes to reconstruct the evolution of our system.

The elements needed for generalized gradient dynamics are dissipation potential functional $\Xi(\boldsymbol{x}, \boldsymbol{x}^*)$, and entropy functional $S(\boldsymbol{x})$. Similarly to VONNs [35], we split the model into two neural networks, one learning each function. Subsequently, we can construct the loss function similarly to MNNs [12] in the form

$$L \propto ||\dot{\boldsymbol{x}}^{(\text{nn.})} - \dot{\boldsymbol{x}}^{(\text{true})}||^2 = \left|\left| \left. \frac{\partial \Xi^{(\text{nn.})}}{\partial \boldsymbol{x}^*} \right|_{\boldsymbol{x}^* = \frac{\partial S^{(\text{nn.})}}{\partial \boldsymbol{x}}} - \dot{\boldsymbol{x}}^{(\text{true})} \right|\right|^2,$$

where $\dot{\boldsymbol{x}}^{(\text{true})}$ is estimated from the ground truth data by means of finite differences:

$$\dot{\boldsymbol{x}}_j^{(\text{true})} \approx \frac{\boldsymbol{x}_{j+1}^{(\text{true})} - \boldsymbol{x}_{j-1}^{(\text{true})}}{2\Delta t}.$$

Equipped with the loss function and formulation of our problem, we can present the architecture of our networks and their training.

3.2.1 Entropy Architecture

First, we establish what constraints we want to impose on entropy. As specified in Chapter 1, the existence of the maximum of entropy implies at least its local concavity. Entropies that are not concave globally, but only locally, lead to instabilities and give rise to complex phenomena known as phase transitions [5, 6]. Since we will not be dealing with systems undergoing phase transitions (in contrast to [35]), we can assume entropy to be a concave function of $\boldsymbol{x}$ everywhere. This can be done by using a slightly modified Fully Input Convex Neural Network [39] (FICNN) architecture, which we will now explain.

Fully Input Convex Neural Networks first appeared in the paper [39], and their idea is straightforward: Using multivariable calculus, we can try to find the right activation functions and parameters of a neural network such that it is fully convex in all its inputs. Referring to Eq. (2.1), we can immediately see that these requirements will require at least some of the weights to be non-negative. However, these restrictions raise another important question. Does enforcing convexity in this way somehow limit the representational power of our network?

To answer this question, Fully Input Convex Neural Networks introduce a more complex architecture than the standard feedforward networks [39]. In Chapter 2, we discussed how a neural network can be partitioned into layers. In the feedforward design, the next layer can always be computed by taking the previous layer, contracting it with the weight tensor, adding bias, and applying the activation function. In contrast, FICNNs work by each layer being computed not only from the previous layer but also directly from the initial vector that is put in. Mathematically, if we use the letter i for indexing, we can write the relationship between the subsequent layers used in FICNNs as

$$\boldsymbol{x}_{i+1} = \boldsymbol{\varphi}_i(W_i \cdot \boldsymbol{x}_i + W_i^{(0)} \cdot \boldsymbol{x}_0 + \boldsymbol{b}_i), \tag{3.5}$$

which can be further illustrated by the diagram in Fig. 3.1.

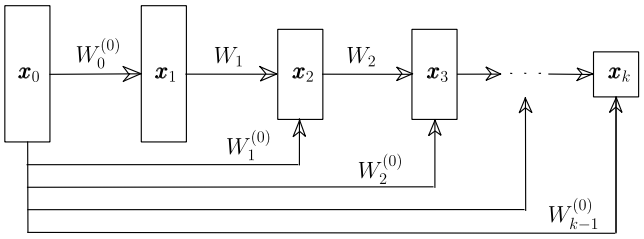


Figure 3.1 Fully Input Convex Neural Network [39]

Now, one of the main results of [39] regarding FICNNs is that given all of the elements of the corresponding weight tensors W_i are non-negative, and all of the activation functions $\boldsymbol{\varphi}_i$ are (element-wise) convex and non-decreasing, the entire network mapping is going to be convex [39]. As a small modification, if we then set the last layer not to be calculated as $x_n = \varphi_n(\ldots)$ but $x_n = -\varphi_n(\ldots)$,

we obtain an input concave neural network without any bigger changes to the architecture.

Evidently, one of the main goals of FICNNs is to mitigate the negative effect of weight enforcement by creating something like "other" connections in the network. In the foundational paper [39], the authors claim that despite the restriction of weights, the representational power of Fully Input Convex Neural Networks is still significant and support this claim empirically, applying them to multiple complicated classification problems.

Finally, we still have to comment on how to apply this architecture in practice. Since we need to use a (preferably smooth) convex non-decreasing activation function, we can opt to use softplus on all of our layers (see Chapter 2). Nonetheless, we still have not explained how to enforce the positivity of the given weights. Here, we will again take inspiration from VONNs [35] since they have elegantly solved this problem and tested their concept in practice. As they conceive it, instead of learning the actual weights used in the computation, the neural network is learning weight tensors $\tilde{W}_i$, which are further reparameterized as

$$(W_i)_{mn} = \begin{cases} (\tilde{W}_i)_{mn} + \exp(-\varepsilon), & (\tilde{W}_i)_{mn} \geq 0, \\ \exp((\tilde{W}_i)_{mn} - \varepsilon), & (\tilde{W}_i)_{mn} < 0, \end{cases} \tag{3.6}$$

where ε is an arbitrary positive parameter, which they choose to be 5 [35]. As we can notice, this reparameterization turns non-positive weights to positive, serving as something similar to a modulus function.

After this, we will impose no further constraints on entropy.

3.2.2 Dissipation Potential Architecture

For building the network learning dissipation potential, our approach will be identical to that of a Variational Onsager Neural Networks [35], as the conditions imposed by them on conjugate dissipation potential Ξ^* are the same as the ones we impose on our dissipation potential Ξ. This implies two things: using a reparameterization of the potential and implementing it via a Partially Input Convex Neural Network architecture.

Starting with reparameterization, we will proceed with Eq. (3.4), as described in Subsection 3.1.3. Therefore, if we denote the network's output as $\Psi(\boldsymbol{x}, \boldsymbol{x}^*)$, it stands that

$$\Xi(\boldsymbol{x}, \boldsymbol{x}^*) = \Psi(\boldsymbol{x}, \boldsymbol{x}^*) - \Psi(\boldsymbol{x}, \boldsymbol{0}) - \boldsymbol{x}^* \cdot \left.\frac{\partial \Psi}{\partial \boldsymbol{x}^*}\right|_{\boldsymbol{x}^*=\boldsymbol{0}}.$$

Next, we address partial input convexity. Previously, we discussed how Fully Input Convex Neural Networks enforce total convexity of the output in all inputs. In the same paper as FICNNs [39], the authors present the Partially Input Convex Neural Networks (PICNNs) architecture, which is a way to enforce convexity only in some of the inputs (in our case, only in $\boldsymbol{x}^*$, not in $\boldsymbol{x}$).

This leads to significantly more convoluted architecture than in the case of FICNNs. To begin with, we create what we could call two "branches" for the inputs to be parsed, one for $\boldsymbol{x}^*$, and another one for $\boldsymbol{x}$. The branches then are connected through different weight tensors; a rough sketch omitting some connections can be seen in Fig. 3.2.

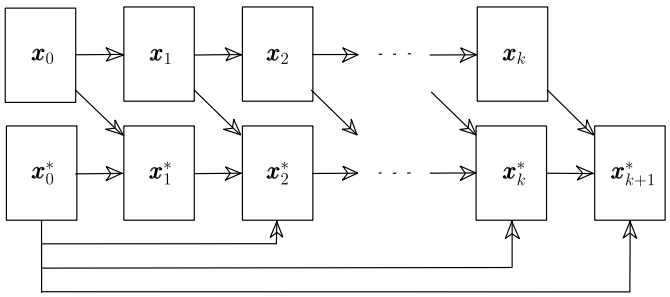


Figure 3.2 Partially Input Convex Neural Network [39]

That being said, these connections do not implement only matrix multiplication, as in the case of FICNNs, but also the element-wise product of vectors, also known as the Hadamard product [39, 35]. For two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ we can define it as

$$(\boldsymbol{u} \odot \boldsymbol{v})_m = u_m v_m, \qquad \boldsymbol{u} \odot \boldsymbol{v} \in \mathbb{R}^d. \tag{3.7}$$

Note that we do *not* employ Einstein's summation convention in Eq. (3.7). Equipped with this Hadamard product, and following the approach in [39], we can write the formulae analogous to Eq. (3.5) for PICNNs. The branch corresponding to $\boldsymbol{x}$ is just a simple feedforward network, as in

$$\boldsymbol{x}_{i+1} = \boldsymbol{\varphi}_i(W_i \cdot \boldsymbol{x}_i + \boldsymbol{b}_i),$$

and on the contrary, the branch propagating $\boldsymbol{x}^*$ consists of multiple weight tensors, biases, and Hadamard products in the form

$$\boldsymbol{x}_{i+1}^* = \boldsymbol{\psi}_i\Bigg(W_i^{(\boldsymbol{x}^*)} \cdot \Big(\boldsymbol{x}_i^* \odot [W^{(\boldsymbol{x}^*,\boldsymbol{x})} \cdot \boldsymbol{x}_i + \boldsymbol{b}_i^{(\boldsymbol{x}^*)}]_+\Big) + W_i^{(0)} \cdot \Big(\boldsymbol{x}_0^* \odot [W^{(0,\boldsymbol{x})} \cdot \boldsymbol{x}_i + \boldsymbol{b}_i^{(0)}]\Big) + W_i^{(\boldsymbol{x})} \cdot \boldsymbol{x}_i + \boldsymbol{b}_i^*\Bigg),$$

where $\{\boldsymbol{\psi}_i\}_{i=1}^n$ is another set of activation functions. The subscript + here denotes an arbitrary activation function that is always positive. According to the results [39, 35], the function computed using Partially Input Convex Neural Network is then convex in $\boldsymbol{x}^*$ provided that $\boldsymbol{\psi}_i$ are convex, non-decreasing, and that weights corresponding to $W_i^{(\boldsymbol{x}^*)}$ are all non-negative.

When applied, the non-negativity of the given weights can be enforced by using the same reparameterization as in Eq. (3.6). Furthermore, we can use the softplus activation for both $\boldsymbol{\varphi}_i$, $\boldsymbol{\psi}_i$, and the subscript +.

3.2.3 Training Strategy

Finally, we will briefly comment on the process of training our neural networks. Recall that in Chapter 2, we introduced two distinct classes of optimizers: first-order optimizers and second-order optimizers. If we were to strictly follow our main design inspiration [35], we would use the first-order optimizer Adam [28]. However, during our research, an article [42] from 2024 by Rathore et al. came to our attention.

In this paper, the authors compare the effectiveness of using the second-order optimizer L-BFGS alone, using the Adam optimizer alone, and using the combination of both optimizers, switching from one to the other during training. Empirically, they found that the last mixed approach outperformed the other two in nearly all of the examples they tested [42]. Furthermore, they tried to provide an interpretation of this result using optimization theory.

Building on these findings, our strategy will be as follows: We start with the Adam optimizer, setting its learning rate to `lr=1e-3`. After half of the pre-planned epochs (training loop iterations) are complete, we switch to L-BFGS with the learning rate `lr=1e-1`, using it until the entire training phase ends.

4 Results on Example Systems

Having introduced our model in the previous chapter, it is time to perform numerical experiments. All of the code used to generate and test our results was written in the language `Python` using the libraries `NumPy` [43] and `PyTorch` [44]. For visualization and plotting, we used the packages `Matplotlib` [45] and `SciencePlots` [46]. The code itself is available at a public GitHub repository.

The first system chosen for our tests is a deterministic overdamped particle in 1D and 2D. It was selected as a system that can be described with one of the simplest dissipation potentials and entropy, as we will see below.

The next example we modeled concerns the kinetics of a system of chemical reactions described by the law of mass action. In contrast to an overdamped particle, its dynamics are nonlinear and thus more complex.

Finally, we address the modeling of discretized continuous systems with many degrees of freedom. For this purpose, we specifically study the diffusion equation obtained from Fick's first and second laws, similarly to [35].

For all of these examples, we compare the predicted and analytical trajectories, and where it is possible, we show how the learned dissipation potential and entropy differ from the analytical solutions.

Technical Note

Before delving into the specifics, we will state that all of the networks we used were composed of a small number of hidden layers (3-5), with each layer using between 8 and 20 neurons.

Further, we initialized all the weights in our neural networks using the `kaiming_normal_` initialization in `PyTorch` with default settings. We trained them all on a graphics card, NVIDIA GeForce GTX 1660 Ti, using the CUDA computing platform.

4.1 Overdamped Particle

A deterministic overdamped particle is described by the equation

$$\dot{\boldsymbol{x}} = -\gamma \boldsymbol{x}, \tag{4.1}$$

where $\gamma > 0$ is the damping coefficient. This system is very well known, playing a key role, for example, in Langevin dynamics [7]. We can see immediately that the entropy and the dissipation potential that produce the overdamped evolution can be found in a quadratic form as

$$S(\boldsymbol{x}) = -\frac{1}{2}||\boldsymbol{x}||^2, \qquad \Xi(\boldsymbol{x}, \boldsymbol{x}^*) = \gamma \frac{1}{2}||\boldsymbol{x}^*||^2,$$

fulfilling all our imposed constraints. However, we have to be careful about uniqueness. Since we only use entropy to calculate the conjugate state variables $\boldsymbol{x}^*$, its formula can be clearly shifted by any constant value. Other considerations

should be about scaling by multiplication. If we use a differently scaled dissipation potential and entropy in the form

$$S(\boldsymbol{x}) = -\frac{1}{C}\frac{1}{2}||\boldsymbol{x}||^2, \qquad \Xi(\boldsymbol{x}, \boldsymbol{x}^*) = C\gamma\frac{1}{2}||\boldsymbol{x}^*||^2, \quad C > 0,$$

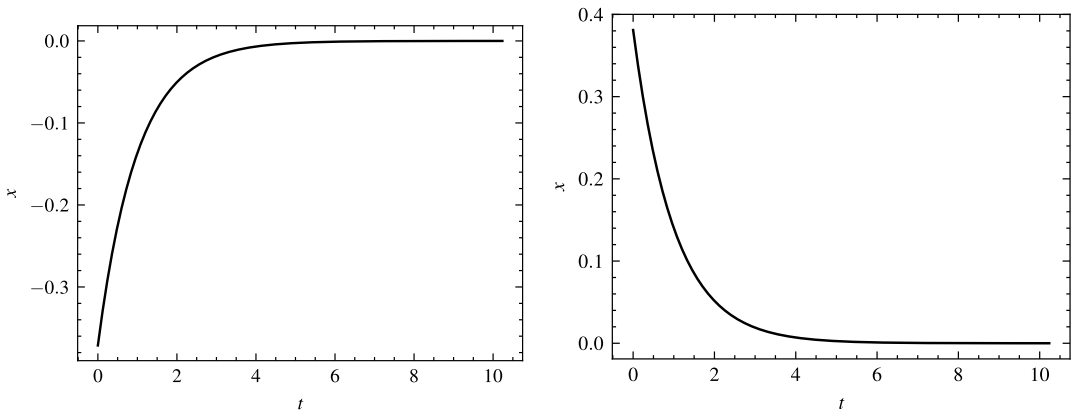
we can see that it yields the same dynamics as our original solution

$$\dot{\boldsymbol{x}} = \frac{\partial \Xi}{\partial \boldsymbol{x}^*} = C\gamma\boldsymbol{x}^* = C\gamma\frac{\partial S}{\partial \boldsymbol{x}} = -\frac{C}{C}\gamma\boldsymbol{x} = -\gamma\boldsymbol{x}.$$

4.1.1 Generating Data & Training

The trajectories for our numerical experiments were generated using Eq. (4.1) and the fourth-order Runge–Kutta integration scheme[1]. Since it is a common practice to train PINNs on normalized datasets [35], we generated the initial values for the state vector $\boldsymbol{x}$ from the uniform distribution over the interval $[-1, 1]$. The damping coefficient was set to $\gamma = 1.0$.

To further specify the dataset, we used a sample size of 128 generated trajectories with 2048 points each. The time step was chosen as `dt = 0.005`. Two example trajectories generated in 1D can be seen in Fig. 4.1; similarly, one example trajectory generated in 2D space can be seen in Fig. 4.2.



(a) **(b)**

Figure 4.1 Example trajectories in 1D; overdamped particle

For the training of our neural networks, we chose the strategy described in Subsection 3.2.3. The number of training epochs was selected to be 1000, as, after multiple experiments, we found the convergence satisfactory. One of the differences when switching between the two optimizers was the batch size. We set the trajectories batch size for the Adam optimizer as 64, while for L-BFGS, we used the full 128 trajectories (see Section 2.3.2). The dependence of the training log loss on the number of iterations can be seen in Fig. 4.3.

[1]To someone, this may seem strange: numerically integrating trajectories from a precise formula for velocity, only to later reconstruct the velocity through finite differences? This is done because we wanted our workflow to better emulate learning from experimental data, even generating some noise through the repeated use of these numerical methods.

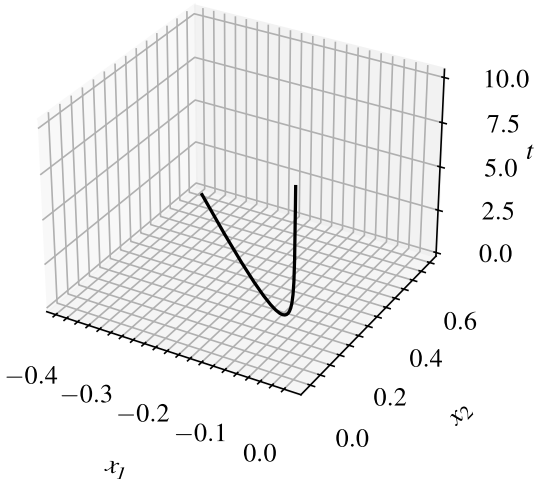
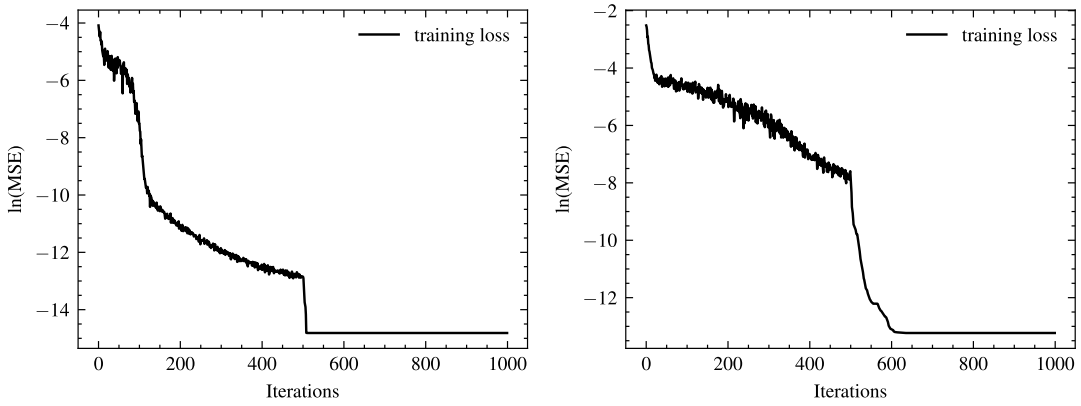


Figure 4.2 Example trajectory in 2D; overdamped particle



(a) 1D **(b)** 2D

Figure 4.3 Log loss decline during training; overdamped particle

In Fig. 4.3, both for the 1D and 2D cases, we can see a sharp decrease in the training log loss after switching to L-BFGS, which provides some support for our selected training strategy.

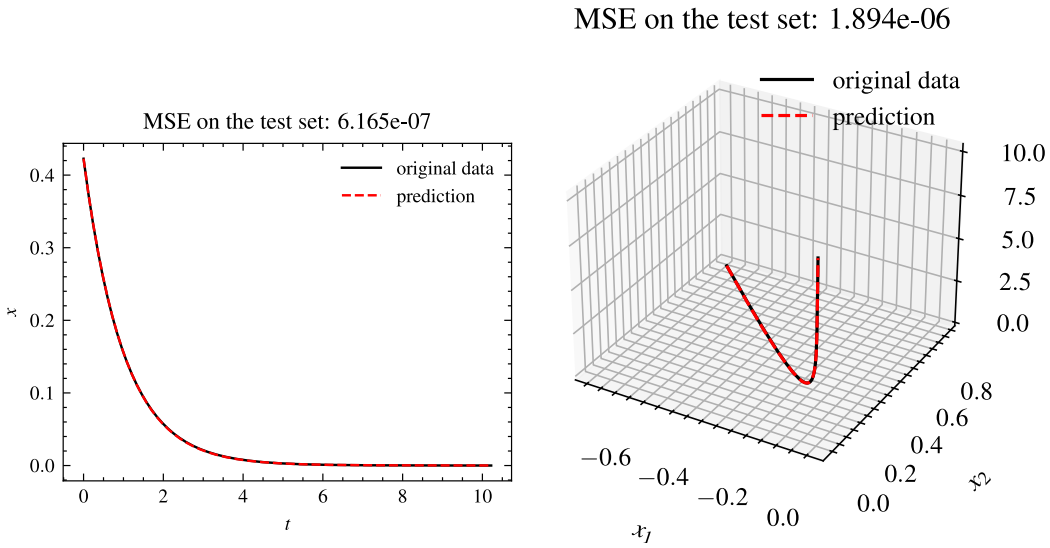
4.1.2 Results

As is customary when training neural networks [24], we split our dataset into two parts: the training set and the test set. The test set was chosen to consist of 20% of all trajectories.

Firstly, let us talk about the predictive power of our model. This can be quantified by taking the test set, applying our trained model to it, and calculating the mean squared error loss from the known and predicted velocities. We can further construct a histogram of the losses on each of the trajectories in the test set; see Fig. 4.5.

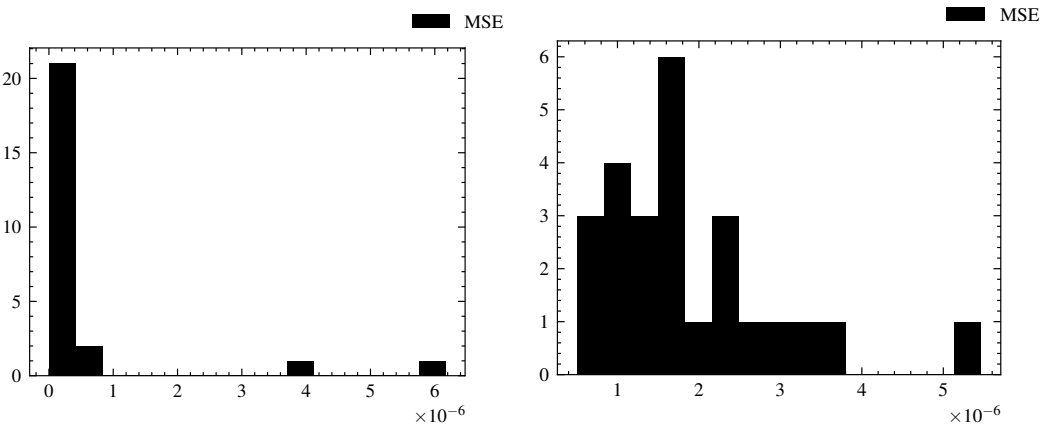
For illustration purposes, we also choose a random trajectory and plot it with its prediction. This prediction is calculated by taking the ground-truth initial conditions and using the RK4 scheme to numerically integrate the system's

trajectory in time, based on the output of our model. The total mean squared error along with the randomly chosen trajectory can be found in Fig. 4.4.



(a) 1D (b) 2D

Figure 4.4 Test trajectory along with the predicted trajectory using the trained model; overdamped particle



(a) 1D (b) 2D

Figure 4.5 Histogram of MSE on the test set; overdamped particle

We observe that the MSE is quite low, in the order of at most 10^{-6}. The predicted trajectories also almost perfectly match the correct ones.

Finally, to see how well our model uses the generalized gradient dynamics framework, we compare the learned entropy and dissipation potential functions with the analytical ones. To account for the shift and scaling mentioned at the beginning of this section, we implement the following mechanism:

1. We estimate the scaling constant C by comparing the learned dissipation potential with the analytical one. More specifically, we use a trivial least

squares estimate in the form

$$C = \frac{\sum_i \Xi_i^{(\text{anal.})} \Xi_i^{(\text{anal.})}}{\sum_i \Xi_i^{(\text{anal.})} \Xi_i^{(\text{pred.})}},$$

where $i = 1, ..., N$ indexes all elements in our discretized domain in state space.

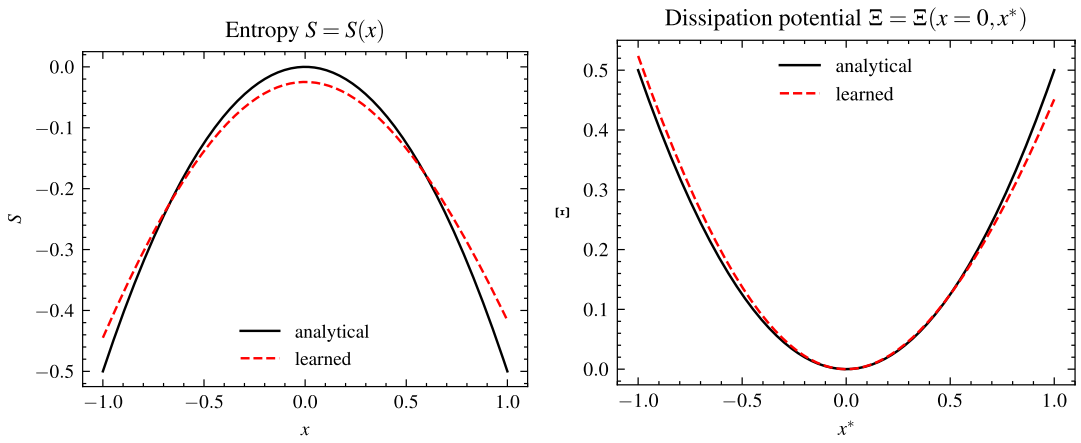
2. As we have found the scaling constant, we recalculate the learned entropy as $S^{(\text{pred.})} := S^{(\text{pred.})}/C$ and the dissipation potential as $\Xi^{(\text{pred.})} := C\Xi^{(\text{pred.})}$. We then compute the estimate of the additive constant K as

$$K = \frac{\sum_i S_i^{(\text{anal.})} - S_i^{(\text{pred.})}}{N}$$

3. Finally, we redefine our scaled and shifted entropy to be

$$S^{(\text{pred.})} := S^{(\text{pred.})} + K.$$

After performing these operations, we can compare the graphs of our functions. The dissipation potential will be plotted with the setting $\boldsymbol{x} = \mathbf{0}$, since what mainly interests us is the dependence on $\boldsymbol{x}^*$. Cases for both dimensions can be seen in Fig. 4.6 and Fig. 4.7.



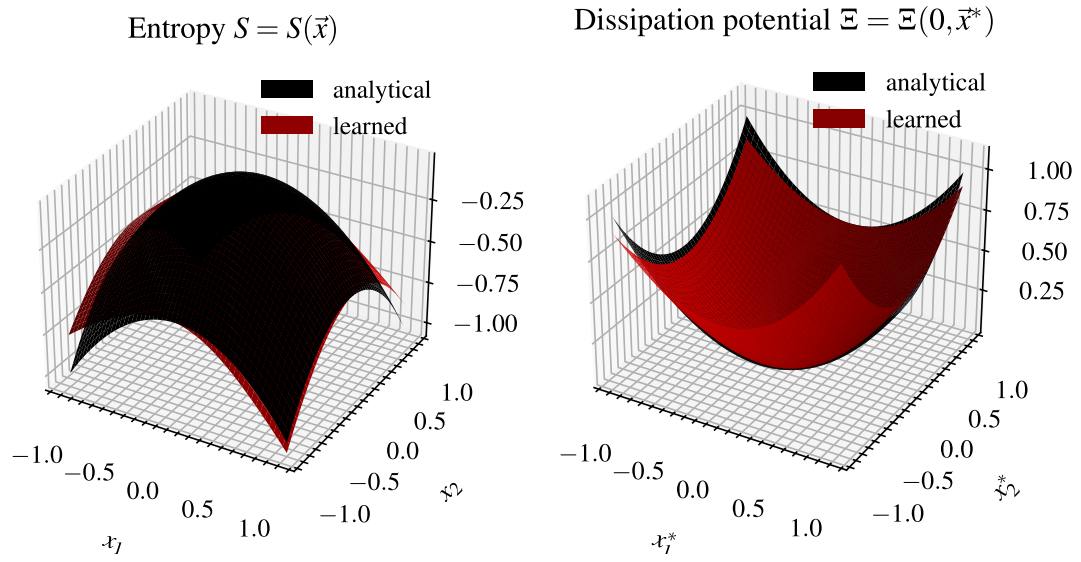
(a) Entropy **(b)** Dissipation Potential

Figure 4.6 Modeled thermodynamic quantities in comparison to their analytical solutions in 1D; overdamped particle

Evidently, while the match of entropy and dissipation potential is not as perfect as with the predicted trajectories, the shape of the functions after rescaling corresponds very well.

4.2 Chemical Reactions

Generally, a chemical reaction can be defined as a process during which matter transforms between different chemical species. For our examples, we will be



(a) Entropy **(b)** Dissipation Potential

Figure 4.7 Modeled thermodynamic quantities in comparison to their analytical solutions in 2D; overdamped particle

concerned with the kinetics of chemical reactions that are occurring at a given constant temperature and constant pressure. Our state vector, which we will denote by $\boldsymbol{c} \in \mathbb{R}_+^M$, will then naturally consist of the concentrations of the reactants and products interacting in these reactions.

One of the most employed models describing chemical kinetics is the classic law of mass action [47]. We borrow its formulation and modern description in terms of generalized gradient dynamics from the book [5].

Let us consider a system of N chemical reactions, each containing up to M chemical species $\mathbb{A}$, the n-th reaction can be then written as

$$\alpha_1^{(n)}\mathbb{A}_1^{(n)} + \ldots + \alpha_M^{(n)}\mathbb{A}_M^{(n)} \rightleftharpoons \beta_1^{(n)}\mathbb{A}_1^{(n)} + \ldots + \beta_M^{(n)}\mathbb{A}_M^{(n)}, \tag{4.2}$$

where for each reaction and each species the number $\nu_{mn} := \beta_m^{(n)} - \alpha_m^{(n)}$ is called the stoichiometric coefficient. As we can see, due to the two indices, the stoichiometric coefficients can be arranged into a stoichiometric matrix.

Next, we define the rate vector $\boldsymbol{\Gamma} \in \mathbb{R}^N$, which quantifies how fast individual reactions are proceeding. According to the law of mass action [47], the vector $\boldsymbol{\Gamma}$ for Eq. (4.2) can be computed as:

$$\Gamma_n = k_f^{(n)} \prod_{m=1}^{M} c_m^{\alpha_m^{(n)}} - k_b^{(n)} \prod_{m=1}^{M} c_m^{\beta_m^{(n)}},$$

where $k_f^{(n)}$, $k_b^{(n)}$ are the forward and backward rate coefficients for each reaction, respectively. Finally, after defining these quantities, the time derivative of the concentration vector can be written in the matrix form as

$$\frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}t} = \nu \cdot \boldsymbol{\Gamma}.$$

Now that we have explained the classical description of chemical kinetics through the mass action law, we show how the same equations can be obtained using generalized gradient dynamics.

As an axiom, let us start with the entropy that is traditionally associated with chemical mixtures:

$$S(\boldsymbol{c}) = -\sum_{m=1}^{M} c_m(\ln c_m + Q_m), \tag{4.3}$$

from the fundamental thermodynamic relation $\mathrm{d}S = \frac{1}{T}\,\mathrm{d}E + \frac{p}{T}\,\mathrm{d}V - \frac{\mu}{T}\,\mathrm{d}c$ we can immediately see that $Q_m + 1$ plays the role of some reference value for a chemical potential per unit of temperature μ/T.

We will further show that the dissipation potential compatible with this description is given by

$$\Xi(\boldsymbol{c}, \boldsymbol{c}^*) = \sum_{n=1}^{N} W_n(\boldsymbol{c})(\cosh(X_n/2) - 1), \tag{4.4}$$

where X_n is the so-called chemical affinity for the n-th reaction. Along with the expression W_n, this affinity can be calculated as

$$X_n = -\sum_{m=1}^{M} c_m^* \nu_{mn}, \qquad W_n = W_0 \sqrt{\prod_{m=1}^{M} c_m^{|\nu_{mn}|}}.$$

If we take the given dissipation potential and differentiate it, trying to derive the law of mass action, we obtain

$$\begin{aligned}\dot{c}_m = \frac{\partial \Xi}{\partial c_m^*} &= \frac{1}{2}\sum_{n=1}^{N} W_n(\boldsymbol{c}) \sinh(X_n/2) \frac{\partial X_n}{\partial c_m^*} = -\frac{1}{2}\sum_{n=1}^{N} W_n(\boldsymbol{c}) \sinh(X_n/2)\nu_{mn} = \\ &- \frac{1}{2}\sum_{n=1}^{N} W_0 \sqrt{\prod_{m=1}^{M} c_m^{|\nu_{mn}|}} \sinh\left(-\sum_{m=1}^{M} \frac{\partial S}{\partial c_m}\frac{\nu_{mn}}{2}\right)\nu_{mn} = \\ &- \frac{1}{2}\sum_{n=1}^{N} W_0 \sqrt{\prod_{m=1}^{M} c_m^{|\nu_{mn}|}} \sinh\left(\sum_{m=1}^{M} (\ln c_m + Q_m + 1)\frac{\nu_{mn}}{2}\right)\nu_{mn}.\end{aligned}$$

Focusing only on the hyperbolic sine in the expression further yields

$$\begin{aligned}&\sinh\left(\sum_{m=1}^{M} (\ln c_m + Q_m + 1)\frac{\nu_{mn}}{2}\right) = \\ &\qquad \sinh\left(\ln\left(\prod_{m=1}^{M} c_m^{\frac{\nu_{mn}}{2}}\right) + \sum_{m=1}^{M}(Q_m + 1)\frac{\nu_{mn}}{2}\right) = \\ &\qquad \frac{1}{2}\left(\prod_{m=1}^{M} c_m^{\frac{\nu_{mn}}{2}} \exp\left(\sum_{m=1}^{M} \frac{(Q_m+1)\nu_{mn}}{2}\right)\right. \\ &\qquad\qquad \left. - \prod_{m=1}^{M} c_m^{-\frac{\nu_{mn}}{2}} \exp\left(-\sum_{m=1}^{M} \frac{(Q_m+1)\nu_{mn}}{2}\right)\right),\end{aligned}$$

and by setting $k_f^{(n)} = \frac{W_0}{4}\exp\left(-\sum_m \frac{(Q_m+1)\nu_{mn}}{2}\right)$, $k_b^{(n)} = \frac{W_0}{4}\exp\left(\sum_m \frac{(Q_m+1)\nu_{mn}}{2}\right)$, we can finally write:

$$\begin{aligned}\dot{c}_m = -\sum_{n=1}^{N} \nu_{mn} \sqrt{\prod_{m=1}^{M} c_m^{|\nu_{mn}|}} \left(k_b^{(n)} \prod_{m=1}^{M} c_m^{\frac{\nu_{mn}}{2}} - k_f^{(n)} \prod_{m=1}^{M} c_m^{\frac{-\nu_{mn}}{2}}\right) = \\ \sum_{n=1}^{N} \nu_{mn} \left(k_f^{(n)} \prod_{m=1}^{M} c_m^{-\Theta(-\nu_{mn})\nu_{mn}} - k_b^{(n)} \prod_{m=1}^{M} c_m^{\Theta(\nu_{mn})\nu_{mn}}\right),\end{aligned}$$

where Θ is used as the Heaviside step function. As we can see, this structure almost resembles the formula for the mass action law. While it may not be completely clear at first sight, when using a balanced chemical reaction where either $\alpha_m^{(n)} = 0$ or $\beta_m^{(n)} = 0$, it is exactly this formula. If $\nu_{mn} < 0$, $\nu_{mn} = -\alpha_m^{(n)}$, and if $\nu_{mn} > 0$, $\nu_{mn} = \beta_m^{(n)}$.

With this in mind, we will conduct all numerical tests using the choice of natural units in which $W_0 = 4$, $Q_m = -1$, and $k_b = k_f = 1$, as it will simplify our expressions.

However, similarly to the previous example of the overdamped particle, we have to address the uniqueness of our solution. We can illustrate the non-uniqueness of the pair of entropy and dissipation potential on the simplest possible chemical reaction in the form

$$A \rightleftharpoons B. \tag{4.5}$$

This reaction evolves, according to the law of mass action, in such a way that

$$\dot{c}_A = c_B - c_A, \qquad \dot{c}_B = c_A - c_B.$$

Looking at this, we can easily construct a dissipation potential Ξ' and entropy S' which satisfy this evolution as quadratic forms

$$S'(\boldsymbol{c}) = -\frac{1}{2}||\boldsymbol{c}||^2, \qquad \Xi'(\boldsymbol{c}, \boldsymbol{c}^*) = \frac{1}{2}\boldsymbol{c}^* \cdot \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \cdot \boldsymbol{c}^*.$$

Therefore, we can see that our description is not unique. In the latter subsections, we will first try to restrict our neural networks by prescribing the dissipation potential and the entropy, and only after these experiments, will we attempt to use the full model, recognizing both the dissipation potential and the entropy from the given data.

4.2.1 Generating Data & Training

For our specific study, we select the system of two chemical reactions, which can be written as:

$$\begin{aligned} A + 2B &\rightleftharpoons C, \\ C + D &\rightleftharpoons A. \end{aligned} \tag{4.6}$$

We once again generate the dataset by using the fourth-order Runge–Kutta scheme utilizing the law of mass action. An example trajectory can be seen in Fig. 4.8

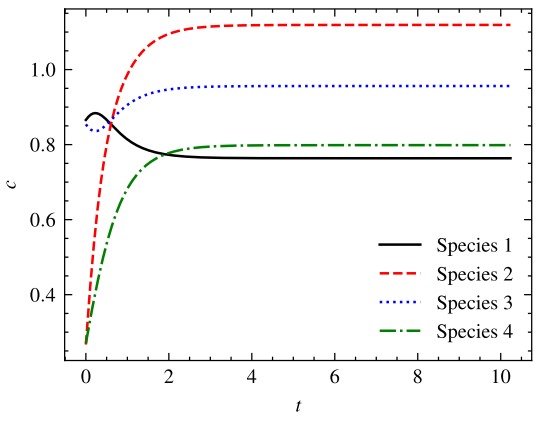


Figure 4.8 Example trajectory; reaction system

We will then use the same number of trajectories and the same time step as is described in Subsection 4.1.1. An important fact for chemical reactions is that the state vector $\boldsymbol{c}$ has to be composed of non-negative numbers; thus, we sample the initial conditions uniformly only from the interval $[0.001, 1]$.

Moreover, the training will proceed similarly to what was detailed in Section 4.1. The exception is the number of training epochs, which we increased from 1000 to 2500, since empirically, we did not find the convergence with 1000 epochs as adequate as before. We present the graphs of training losses for each of our experiments in Subsection 4.2.2.

4.2.2 Results

Prescribed Entropy

As we have said, due to the non-uniqueness of the pair entropy, dissipation potential, we start by prescribing one and seeing how our model learns the other. First, we begin by prescribing entropy using Eq. (4.3) with $Q_m = -1$.

The training loss curve for our model can be seen in Fig. 4.9 below.

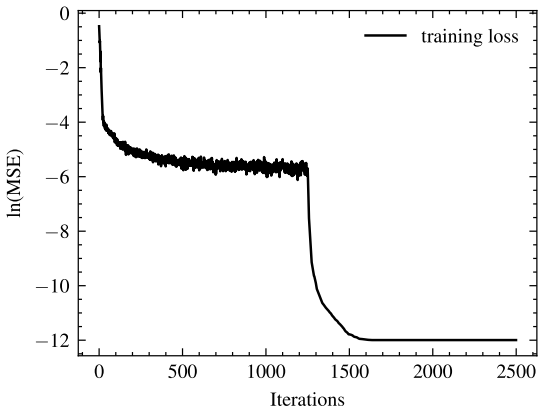
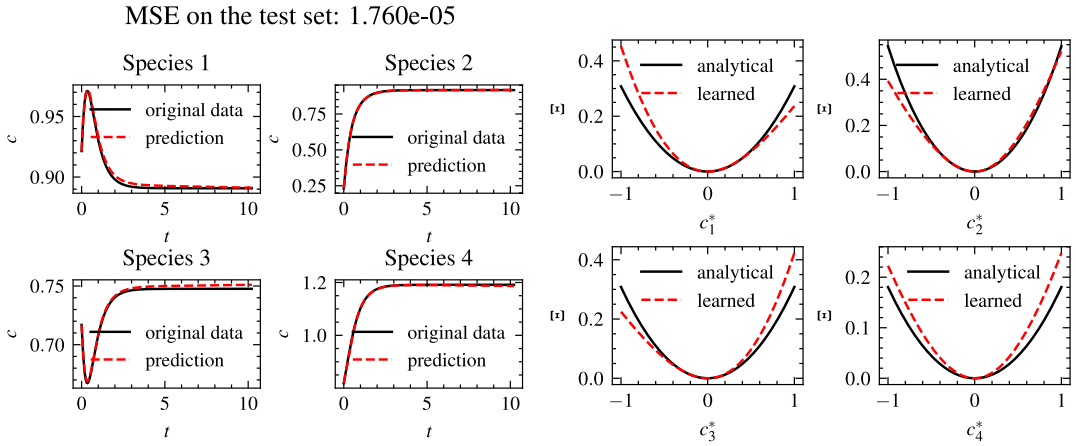


Figure 4.9 Log loss decline during training; reaction system with prescribed entropy

The comparison of real and predicted trajectories for each of the species, along with the computed mean squared error for the entire reaction system, is then included in Fig. 4.10a, from it we observe that while the mean squared error is higher than with the overdamped particle, the trajectories still match quite well.

Finally, we compare the learned dissipation potential with its analytical solution. Since in our particular case the dissipation potential is eight-dimensional, we cannot visualize it directly. We therefore fix the concentrations vector $\boldsymbol{c}$ by setting $c_m = 0.5$, and for each of the four conjugate concentrations, we plot the dissipation potential independently, setting the others to zero.

This produces the graphs in Fig. 4.10b. Once again, while the match is not flawless, it clearly resembles the analytical solution, and therefore, we conclude that our model was able to learn the dynamics at a satisfactory level.



(a) Evolution for each species **(b)** Dissipation potential

Figure 4.10 Main comparison using prescribed entropy; reaction system

Prescribed Dissipation Potential

In the same fashion as we prescribed entropy in the paragraphs above, we will now prescribe the dissipation potential using Eq. (4.4). The difference from before is that even now if we impose the dissipation potential, the entropy remains non-unique, which is due to the parameter $\boldsymbol{Q}$ from Eq. (4.3). Suppose we add such a vector $\boldsymbol{q}$ to $\boldsymbol{Q}$ that $\boldsymbol{q} \cdot \nu = \boldsymbol{0}$, then the dynamics do not change, which can be seen from our derivation of $\dot{\boldsymbol{c}}$. Moreover, the entropy can obviously be shifted by any arbitrary constant.

To mitigate this non-uniqueness, we will perform a series of transformations as follows:

1. We find the estimate of $\boldsymbol{q}$ by calculating it as the difference

$$\boldsymbol{q} = \frac{1}{N} \sum_i \left(\frac{\partial S_i^{(\text{pred.})}}{\partial \boldsymbol{c}} - \frac{\partial S_i^{(\text{anal.})}}{\partial \boldsymbol{c}} \right),$$

   with $i = 1, ..., N$ again indexing all of the elements in our domain.

2. After estimating $\boldsymbol{q}$ we can recalculate the learned entropy as $S^{(\text{pred.})} := S^{(\text{pred.})} - \boldsymbol{c} \cdot \boldsymbol{q}$. After that, the additive constant can be approximated as the average distance between $S^{(\text{pred.})}$ and $S^{(\text{anal.})}$, in the same way as in Subsection 4.1.2.

Now that we have resolved the non-uniqueness problem, we can proceed to presenting the results. The graph of training loss is in Fig. 4.11, and the comparison between trajectories is in Fig. 4.12a.

As for predicted trajectories, their mean squared error and visual match are roughly the same as when we prescribed entropy.

What is novel is the comparison between learned and analytical entropy dependence as seen in Fig. 4.12b. Same as before, the plots were done by letting one component of the concentration vector free while fixing the other to $c_m = 0.5$. The match in the functions seems even better than with prescribing entropy, so we deem the model with prescribed dissipation potential successful.

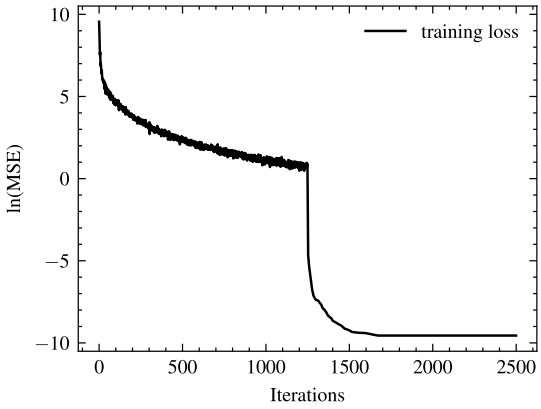
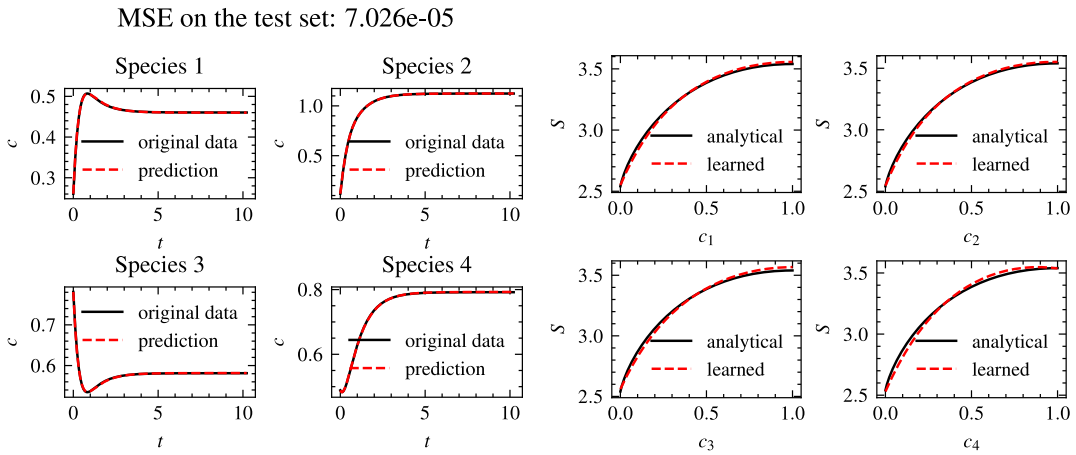


Figure 4.11 Log loss decline during training; reaction system with prescribed dissipation potential



(a) Evolution for each species **(b)** Entropy

Figure 4.12 Main comparison using prescribed dissipation potential; reaction system

General Description

Coming to the end of this section, we try to model the evolution of our reaction network without the prior knowledge of either dissipation potential or entropy. We will go about it in the same way as in both previous attempts, including the reparameterization of entropy.

As we have reflected on before, we do not expect to obtain the correct matching to the analytical solutions. However, we would still like the model to preserve its predictive power and integrate the correct trajectories in the state space.

The training loss curve looks similar to examples with prescribed thermodynamic quantities, as can be seen in Fig. 4.13. Due to convergence issues, which we will address below, we once again increased the number of iterations from 2500 to 3000.

For the comparison of the trajectory prediction, see Fig. 4.14. The mean squared error is on a similar level to the two approaches that we tried before. However, during our initial attempts, with the number of epochs being 2500, we consistently obtained it about two orders of magnitude higher. Before increasing

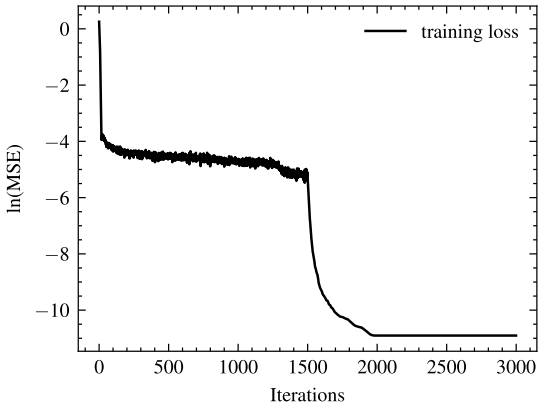


Figure 4.13 Log loss decline during training; reaction system

the number of epochs, we also tried increasing the number of neurons for each network, but that idea was unsuccessful.

This seems to point out an interesting fact: better convergence can, in our case, be achieved by increasing the number of epochs for our network configuration. While it is non-trivial to provide a clear interpretation of this notion, it intuitively seems that although L-BFGS improves the speed of convergence tremendously, it sometimes chooses the wrong minimum, and thus, the pretraining phase with Adam is essential for finding the right solution.

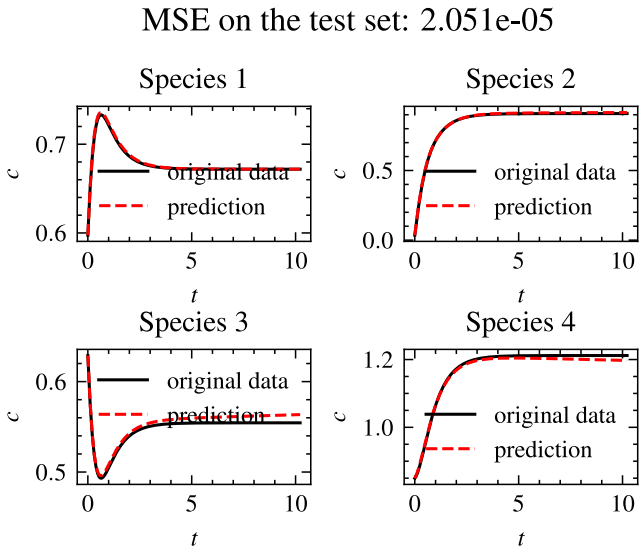
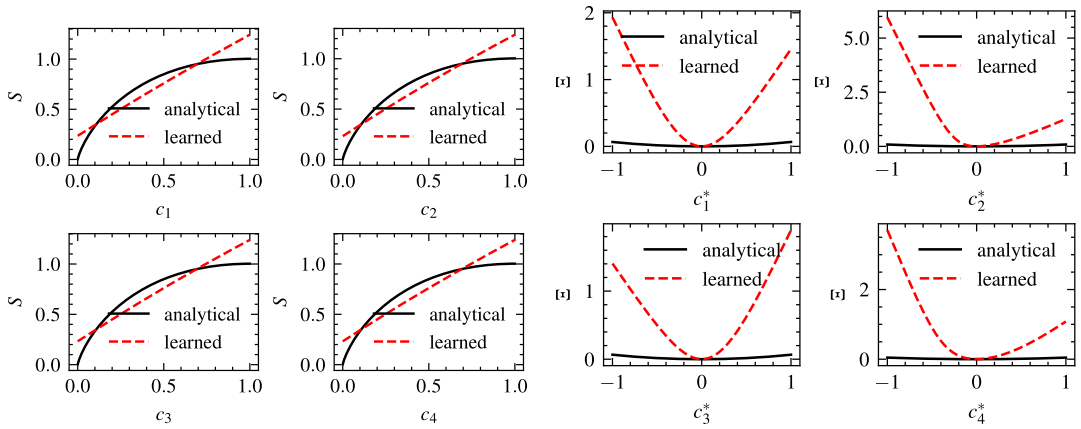


Figure 4.14 Test trajectory along with the predicted trajectory using the trained model; reaction system

Finally, Fig. 4.15 compares the learned thermodynamic functions. As we have suspected, they differ quite radically from our analytical solution. At first glance, what might seem a bit suspicious is that the learned entropy looks almost linear instead of being curved, as we would expect. However, plotting the partial derivatives of S in Fig. 4.16, we can clearly observe that they are *not* constant, and therefore, entropy is not linear.



(a) Entropy **(b)** Dissipation potential

Figure 4.15 Modeled thermodynamic quantities in comparison to their analytical solutions; reaction system

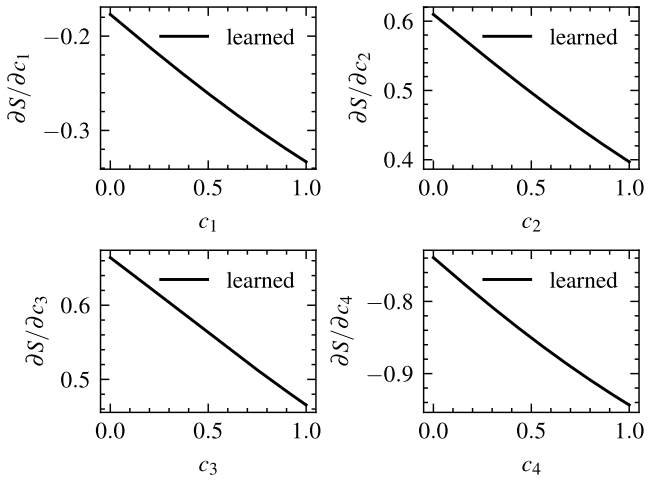


Figure 4.16 Derivatives of the learned entropy function; reaction system

4.3 Fickian Diffusion

Lastly, let us consider, for example, a large number of randomly scattered particles in a fluid spanning the spatial domain Ω. At each point in space and time, we can then define the local concentration of these particles, producing the field function $c = c(\boldsymbol{r}, t)$. Working in arbitrary units, where c is dimensionless, we might now ask: How do these concentrations evolve in time? The answer to this query is given by Fick's first and second laws [5, 7], which (in the case of homogeneous, isotropic diffusivity) tell us:

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) = D \Delta c, \tag{4.7}$$

where D is the diffusivity constant and Δ the Laplace operator. In the context of our work, we wish to describe this evolution with the generalized gradient dynamics.

We begin by writing down the entropy. Axiomatically, we define that

$$S = -\int_\Omega c \ln(c)\, \mathrm{d}^3 r\,.$$

For more details, see [5, 7]. Since we need to involve the gradient of c in some way in our equation, we have to do it through dissipation potential. The simplest possible choice for it is a quadratic potential, producing

$$\Xi = \frac{1}{2}\int_\Omega \tilde{D}(c)||\nabla c^*||^2\, \mathrm{d}^3 r$$

where $\tilde{D}$ is some function linked to diffusivity. Taking the functional derivative of such a dissipation potential, we obtain

$$\dot{c} = \frac{\delta \Xi}{\delta c^*} = -\nabla\cdot(\tilde{D}(c)\nabla c^*) = -\nabla\cdot(\tilde{D}(c)\nabla[-\ln(c)-1]) = \nabla\cdot\left(\frac{\tilde{D}(c)}{c}\nabla c\right).$$

Thus, choosing $\tilde{D} = Dc$ yields the standard Eq. (4.7). Nonetheless, there exist other solutions, such as

$$S = -\frac{1}{2}\int_\Omega c^2\, \mathrm{d}^3 r, \qquad \Xi = \frac{1}{2}\int_\Omega D||\nabla c^*||^2\, \mathrm{d}^3 r\,.$$

In other words, the pair dissipation potential–entropy is once again non-unique.

4.3.1 Generating Data & Training

For our investigation of the diffusion equation, we will be working in 1D, and thus Ω will be a uniform chain, specifically implemented on the interval [0,1]. The discretization of this interval is then done with 64 points — as our work is more of a proof of concept than an actual in-practice simulation, we do not need to perform large-scale, performance-heavy computations.

Since we are calculating a numerical solution, we need to discretize Eq. (4.7). Using finite differences, this means approximating the Laplace operator as:

$$\Delta c_i = \frac{\partial^2 c_i}{\partial x^2} \approx \frac{c_{i+1} - 2c_i + c_{i-1}}{h^2},$$

where h is the spatial resolution of our grid, which we will set naturally to `h=1/64`. For all our simulations, diffusivity will be set to `D=3e-3`.

As for the size of our dataset, to introduce a bigger variety of initial conditions and avoid overfitting, we will use 1024 trajectories; however, to avoid our training being too computationally taxing, we use 64 points in time with the time step `dt = 3e-2` as opposed to `dt = 5e-3` used before. Note that one trajectory now consists of the evolution of our entire computational grid. For numerical integration, we leverage the RK4 scheme, and for the training, 1500 epochs.

The initial and boundary conditions are essential for generating data. For simplicity of the implementation, the boundary conditions of our choice will be periodic[2]. As for initialization, we aim to use a wide variety of functions so that

[2]To give due credit, we remark that a similar choice of basic parameters and boundary conditions was made in [35], where the authors have explored diffusion as well; however, from initial conditions onward, our approaches diverge.

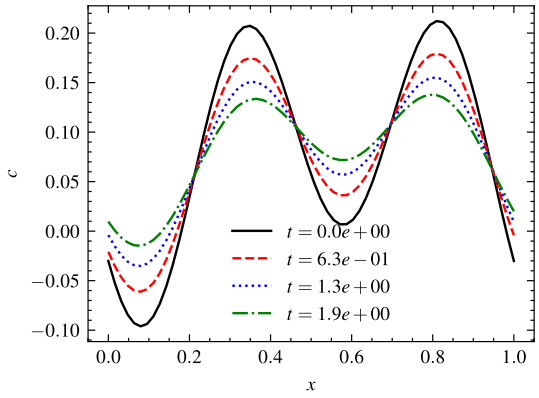


Figure 4.17 Example trajectory; Fickian diffusion

our network generalizes well. While negative concentrations have no physical basis behind them, including them for the training might be useful, as it will add to the richness of our training set without contradicting Eq. (4.7).

Due to the reasons above (better generalization, primarily), we choose to train our network on initializations based on the Fourier modes with frequencies $2\pi n$ (where n is an integer) that form a complete basis of functions on the interval [0,1] [48]. For each trajectory, we will randomly choose a natural number $N \in [0, 15]$, and we will construct a linear combination of sines and cosines of frequencies $n \in 0, 2\pi, ..., 2\pi N$. The coefficients of these linear combinations are sampled from the normal distribution around zero. Since we usually prefer to use smooth functions, we can recall a theorem from real analysis stating that the Fourier coefficients decay with a degree corresponding to the smoothness of the expanded function [48]. Due to this theorem, we implement the normal distribution in a way that its variance is selected to be $1/n^p$ for each of the Fourier modes, specifically setting `p=1.5`.

After we construct the initial conditions in this way, we normalize them using the standard Euclidean norm over all discrete points in Ω. An integrated example trajectory with the described initialization can be seen in Fig. 4.17.

When we finish generating trajectories, we can proceed to training. The graph of the training log loss can be seen in Fig. 4.18.

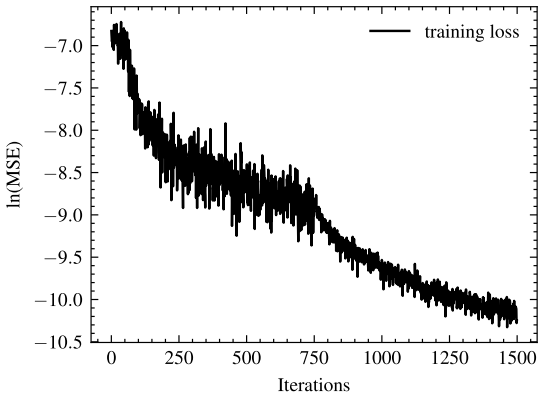


Figure 4.18 Log loss decline during training; Fickian diffusion

4.3.2 Results

Since comparing the dissipation potential and entropy functionals (as opposed to simple functions) to their analytical solutions would be difficult for a many-dimensional system (even more so if the pair is not unique), we are going to focus on the evolutions themselves.

To observe how well our model generalizes after being trained on the Fourier dataset, we evaluate it on three test sets consisting only of trajectories with initial conditions that the model has not seen prior. We dedicate one subsection to each initialization. Moreover, this time for evaluation, we will select strictly positive initial conditions to represent real physical concentrations.

Polynomial Initialization

We start with polynomial initial conditions, as polynomials are one of the most commonly used functions in mathematics and physics. However, we want these polynomials to be non-negative and to have the same value at 0 and 1. The easiest way to accomplish this is by constructing the initial condition in the form

$$c(x, 0) = x(x - 1)p(x) + A,$$

where the constant A will be chosen as the modulus of the minimum of the x-dependent term to shift the entire function above zero. We then construct $p(x)$ in a similar fashion to the previously used Fourier modes, sampling a random number from 0 to 4 as the degree of $p(x)$ and choosing coefficients of each of the terms from [-1,1]. Finally, we apply normalization.

By plotting four graphs, we are going to compare the original and learned trajectory at the beginning, 1/3, 2/3, and the end of the evolution (similarly to Fig. 4.17). We are also including the total mean squared error on the test set. This comparison can be seen in Fig. 4.19a.

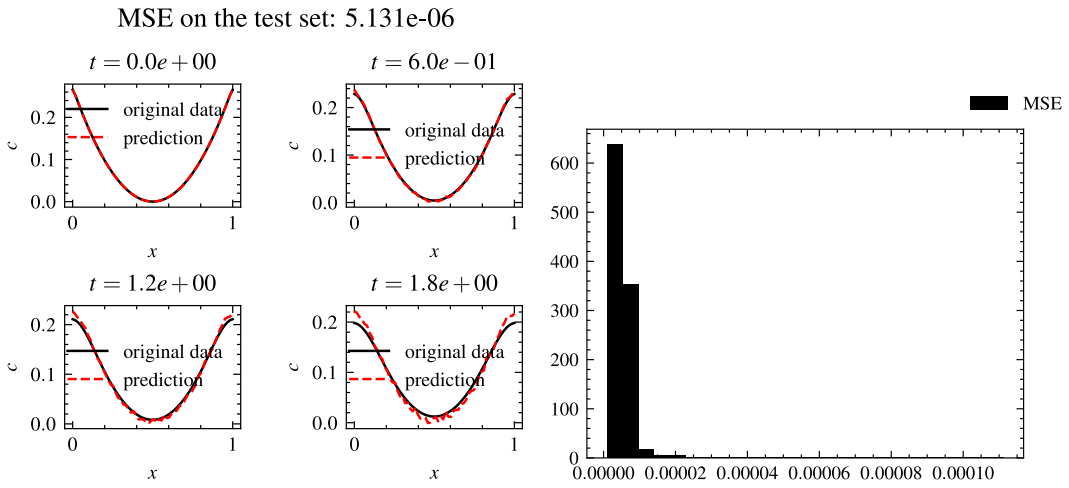
Additionally, we plot a histogram of the mean squared errors on all of the trajectories in the test set in Fig. 4.19.

As we can see, the comparison looks satisfactory, with the MSE being consistently small, in the order of 10^{-6}. This can be, however, heavily influenced by the overall slow evolution of polynomials, as they do not reach equilibrium as fast as the other examples described below.

Discontinuous Initialization

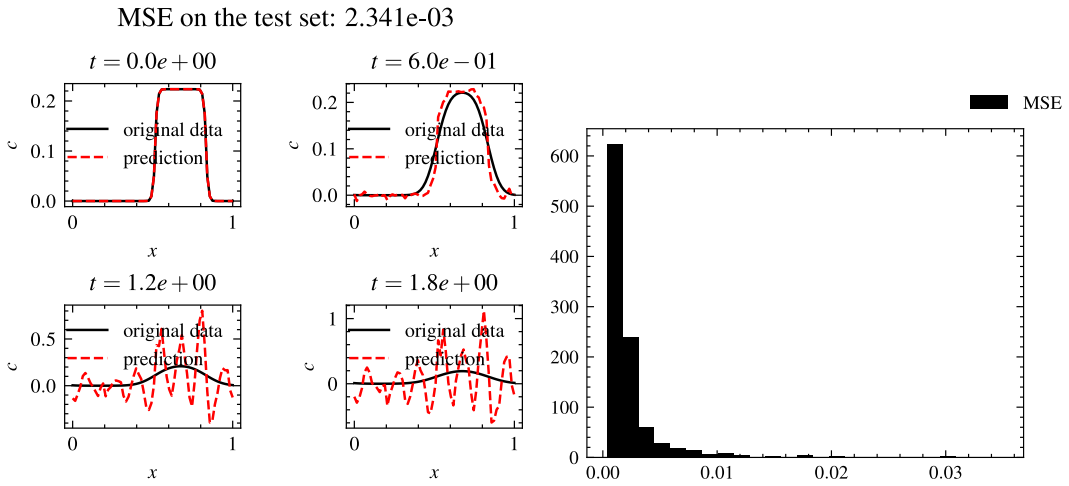
Now, we turn from continuous polynomials to discontinuous initializations, which we define as a linear combination of step functions. We construct such initial conditions by randomly choosing the number of steps N from 1 to 5, then uniformly choosing their height from 0.1 to 1. We position these steps by selecting the width and the center for each one. To avoid influencing the periodic boundary conditions, we limit the centers to be from the interval $[1/N,\ 1 - 1/N]$ with the widths of these steps being from $1/4N$ to $1/2N$.

For analysis, we will follow the same procedure as with polynomial initialization. Therefore, both of the graphs, equivalent to the ones before, are in Fig. 4.20.



(a) Learned evolution **(b)** Histogram of MSE on the test set

Figure 4.19 Main comparison for polynomial initialization; Fickian diffusion



(a) Learned evolution **(b)** Histogram of MSE on the test set

Figure 4.20 Main comparison for discontinuous initialization; Fickian diffusion

Looking at the results, we observe that the MSE on the test set is much higher than in the case of the polynomial initial conditions. From Fig. 4.20, we can also infer that neural networks learned on sine and cosine initialization struggle with the final steps of the evolution approaching equilibrium.

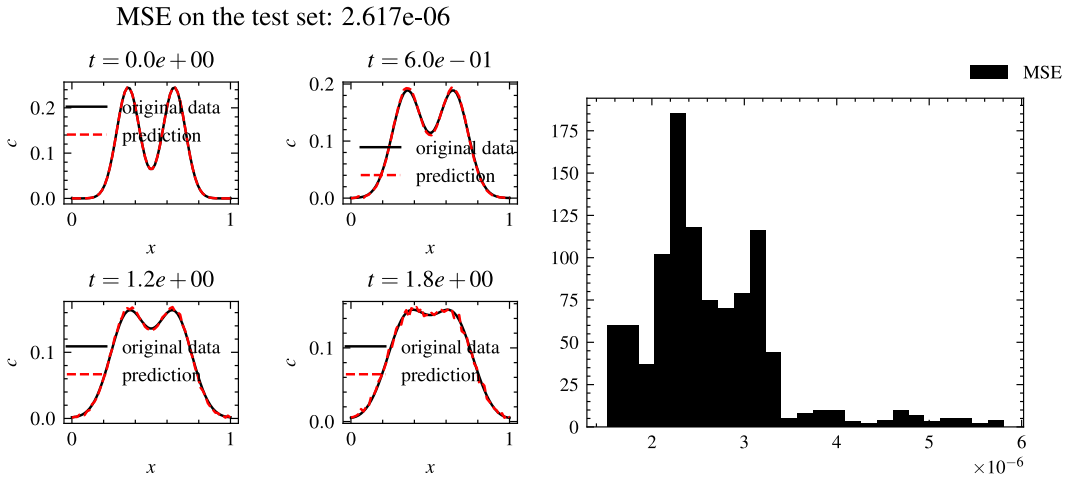
While we are unable to provide a concrete analysis of this phenomenon, we suspect that both the discontinuity and the relatively small capacity of our networks may be causing it. One factor that might influence the training is that we are using the smooth softplus activation, and therefore, our entire networks act as a smooth mapping. Additionally, expansion into the Fourier series sometimes struggles with what we call the Gibbs phenomenon [48], which could also partly explain the higher error.

Exponential Spikes Initialization

Lastly, for something different, we test our model on an initialization composed of two exponential spikes in the form:

$$c(x,0) = \exp\left(-\left(\frac{x+(0.5-x_0)}{0.1}\right)^2\right) + \exp\left(-\left(\frac{x+(0.5+x_0)}{0.1}\right)^2\right),$$

where $x_0 \in [0.1, 0.3]$. We further normalize this function in the same way as previous examples. The main comparison analogous to previous initializations can be observed in Fig. 4.21.



(a) Learned evolution **(b)** Histogram of MSE on the test set

Figure 4.21 Main comparison for exponential spikes initialization; Fickian diffusion

Notice that the MSE is very low, similar to the polynomial initialization. This seems to empirically imply that the model can effectively learn evolutions even with relatively sharp initial conditions; however, we also suspect that such a success can still be a result of the similarity of two peaks to some of the sines and cosines on which we trained our networks.

4.4 Summary

Finally, we briefly summarize the findings of Chapter 4. Using our deep learning model, we have conducted numerical experiments on the three examples above: the overdamped particle, chemical reactions, and Fickian diffusion.

The overdamped particle system was the simplest and most straightforward of the three studied. It has a quadratic dissipation potential and quadratic entropy, which our neural networks learned convincingly. The matching between the ground truth trajectories and the predicted ones was also reliable, and therefore, the entire experiment on this system was a success.

For chemical reactions, our deep learning model was able to capture the kinetics of chemical reactions correctly. This was, as is the case with all our experiments, measured using independent test sets to which the model did not have access

until the end of its training. Furthermore, it was shown that if the model is conditioned by prescribing the analytical dissipation potential or entropy, it will successfully reproduce the second quantity. In the absence of such preconditioning, the model learns some other compatible form of these quantities (we suspect quadratic functions, as they are the simplest), which, although they are not a standard choice, will satisfy the second law of thermodynamics.

Finally, Fickian diffusion was meant to show how neural networks based on generalized gradient dynamics extend to many-dimensional systems. Focusing only on the comparison of predicted trajectories, we have demonstrated how training the neural networks on a diverse initialization dataset comprised of Fourier modes (using periodic boundary conditions) can help them generalize and solve the diffusion equation with multiple different initial conditions. Our model has encountered struggles only with discontinuous initializations, where its mean-squared error is multiple orders of magnitude higher than with other tests. While the interpretation of this result is inconclusive, we suspect a few potential explanations, such as the Gibbs phenomenon or the smaller capacity of our model. Nevertheless, the other tested continuous initial conditions have yielded accurate results.

Conclusion

To finish off our work, we briefly summarize what we have accomplished in a broader context and outline the potential extensions in this research.

We have successfully implemented a machine learning scheme that recognizes the structure of generalized gradient dynamics in a time series of a given dissipative system, fulfilling the main goal of this thesis. This was done by forcing our deep learning model to learn the convex dissipation potential and concave entropy, optimizing a loss function given by the difference between the ground truth data and its prediction, as is customary in deep learning. Our model can learn these thermodynamic quantities and reconstruct the evolution of the state of the system from them without assuming any specific form of balance equations prior to training.

We further believe that our thesis adds meaningful and novel contributions beyond the articles we reviewed in Chapter 3. Specifically, the application of such a deep learning framework to the kinetics of chemical reactions was not shown before, and we are not aware of any machine learning scheme that uses the entropy functional in the same form as we do.

Finally, one other contribution of this thesis is that we have provided a comprehensive review of the underlying theory from both thermodynamics and machine learning, going through a wide range of literature containing state-of-the-art findings in these fields.

As we have mentioned several times throughout this work, we aim to, in the future, extend our scheme to include both dissipative and non-dissipative terms, modeling systems through GENERIC. Furthermore, we would also like to explore how the newly developed machine learning techniques of the so-called neural operators — which allow constructing discretization-agnostic deep learning models — could be connected to our present method.

Bibliography

1. Nobel Prize Outreach AB. *The Nobel Prize in Physics 2024* [online]. 2024. [visited on 2025-04-21]. Available from: https://www.nobelprize.org/prizes/physics/2024/summary/.
2. DeepMind. *AI achieves silver-medal standard solving International Mathematical Olympiad problems*. 2024. Available also from: https://www.deepmind.com/blog/ai-achieves-silver-medal-standard-solving-international-mathematical-olympiad-problems. Accessed: 2025-04-24.
3. Jaynes, E. T. Information Theory and Statistical Mechanics. *Phys. Rev.* 1957, vol. 106, pp. 620–630. Available from DOI: 10.1103/PhysRev.106.620.
4. Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal*. 1948, vol. 27, no. 3, pp. 379–423. Available from DOI: 10.1002/j.1538-7305.1948.tb01338.x.
5. Pavelka, Michal; Klika, Václav; Grmela, Miroslav. *Multiscale Thermo-Dynamics: Introduction to GENERIC*. Berlin, Boston: De Gruyter, 2018. ISBN 9783110350951. Available from DOI: 10.1515/9783110350951.
6. Callen, H.B. *Thermodynamics: an introduction to the physical theories of equilibrium thermostatics and irreversible thermodynamics*. Wiley, 1960. Available also from: http://books.google.cz/books?id=mf5QAAAAMAAJ.
7. Lebon, G.; Jou, D. *Understanding Non-equilibrium Thermodynamics: Foundations, Applications, Frontiers*. Springer Berlin Heidelberg, 2008. SpringerLink: Springer e-Books. ISBN 9783540742524. Available also from: https://books.google.cz/books?id=oN9PprnxKncC.
8. Jou, D.; Casas-Vázquez, J.; Lebon, G. *Extended Irreversible Thermodynamics*. New York: Springer Berlin, Heidelberg, 2001. ISBN 9783642625053. Available from DOI: 10.1007/978-3-642-56565-6.
9. I. Müller, T. Ruggeri. *Rational extended thermodynamics*. Springer New York, NY, 1998. ISBN 9781461274605. Available from DOI: 10.1007/978-1-4612-2210-1.
10. Grmela, Miroslav; Öttinger, Hans Christian. Dynamics and thermodynamics of complex fluids. I. Development of a general formalism. *Phys. Rev. E*. 1997, vol. 56, pp. 6620–6632. Available from DOI: 10.1103/PhysRevE.56.6620.
11. Öttinger, Hans Christian; Grmela, Miroslav. Dynamics and thermodynamics of complex fluids. II. Illustrations of a general formalism. *Phys. Rev. E*. 1997, vol. 56, pp. 6633–6655. Available from DOI: 10.1103/PhysRevE.56.6633.
12. Cueto, Elias; Chinesta, Francisco. Thermodynamics of Learning Physical Phenomena. *Archives of Computational Methods in Engineering*. 2023, vol. 30, no. 8, pp. 4653–4666. ISSN 1886-1784. Available from DOI: 10.1007/s11831-023-09954-5.

13. Otto, Felix. The geometry of dissipative evolution equations: the porous medium equation. *Communications in Partial Differential Equations*. 2001, vol. 26, no. 1-2, pp. 101–174. Available from DOI: 10.1081/PDE-100002243.

14. Ginzburg, V.L.; Landau, L.D. On the theory of superconductivity. *Zhur. Eksp. Theor. Fiz.* 1950, vol. 20, pp. 1064–1082.

15. Onsager, L.; Machlup, S. Fluctuations and Irreversible Processes. *Phys. Rev.* 1953, vol. 91, pp. 1505–1512. Available from DOI: 10.1103/PhysRev.91.1505.

16. Machlup, S.; Onsager, L. Fluctuations and Irreversible Process. II. Systems with Kinetic Energy. *Phys. Rev.* 1953, vol. 91, pp. 1512–1515. Available from DOI: 10.1103/PhysRev.91.1512.

17. Mielke, Alexander; Renger, D.; Peletier, Mark. On the Relation between Gradient Flows and the Large-Deviation Principle, with Applications to Markov Chains and Diffusion. *Potential Analysis*. 2013, vol. 41, pp. 1293–1327. Available from DOI: 10.1007/s11118-014-9418-5.

18. Mielke, Alexander; Renger, D. R. Michiel; Peletier, Mark A. A Generalization of Onsager's Reciprocity Relations to Gradient Flows with Nonlinear Mobility. *Journal of Non-Equilibrium Thermodynamics*. 2016, vol. 41, no. 2, pp. 141–149. Available from DOI: 10.1515/jnet-2015-0073.

19. Landau, L.D.; Lifshitz, E.M. *Mechanics*. Butterworth-Heinemann, 1976. Butterworth Heinemann. ISBN 9780750628969. Available also from: http://books.google.cz/books?id=e-xASAehg1sC.

20. Feynman, R. P. Space-Time Approach to Non-Relativistic Quantum Mechanics. *Rev. Mod. Phys.* 1948, vol. 20, pp. 367–387. Available from DOI: 10.1103/RevModPhys.20.367.

21. Hunt, Katharine L. C.; Ross, John. Path integral solutions of stochastic equations for nonlinear irreversible processes: The uniqueness of the thermodynamic Lagrangian. *The Journal of Chemical Physics*. 1981, vol. 75, no. 2, pp. 976–984. ISSN 0021-9606. Available from DOI: 10.1063/1.442098.

22. Gesamtausgabe, L.B. *Ludwig Boltzmann Gesamtausgabe - Collected Works*. 1983. ISBN 9789904000071.

23. Peliti, Luca; Rechtman, Raúl. Einstein's Approach to Statistical Mechanics: The 1902-04 Papers. *Journal of Statistical Physics*. 2017, vol. 167. Available from DOI: 10.1007/s10955-016-1615-8.

24. Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron. *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016. Adaptive computation and machine learning. ISBN 9780262035613.

25. Leshno, Moshe; Lin, Vladimir Ya.; Pinkus, Allan; Schocken, Shimon. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*. 1993, vol. 6, no. 6, pp. 861–867. ISSN 0893-6080. Available from DOI: 10.1016/S0893-6080(05)80131-5.

26. Robbins, Herbert; Monro, Sutton. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*. 1951, vol. 22, no. 3, pp. 400–407. Available from DOI: 10.1214/aoms/1177729586.

27. Duchi, John; Hazan, Elad; Singer, Yoram. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 2011, vol. 12, no. null, pp. 2121–2159. ISSN 1532-4435.

28. Kingma, Diederik P.; Ba, Jimmy. Adam: A Method for Stochastic Optimization. In: Bengio, Yoshua; LeCun, Yann (eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. Available from DOI: 10.48550/arXiv.1412.6980.

29. BROYDEN, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*. 1970, vol. 6, no. 1, pp. 76–90. ISSN 0272-4960. Available from DOI: 10.1093/imamat/6.1.76.

30. Fletcher, R. A new approach to variable metric algorithms. *The Computer Journal*. 1970, vol. 13, no. 3, pp. 317–322. ISSN 0010-4620. Available from DOI: 10.1093/comjnl/13.3.317.

31. Goldfarb, Donald. A Family of Variable-Metric Methods Derived by Variational Means. *Mathematics of Computation* [online]. 1970, vol. 24, no. 109, pp. 23–26 [visited on 2025-04-12]. ISSN 00255718, ISSN 10886842. Available from DOI: 10.2307/2004873.

32. Shanno, D. F. Conditioning of Quasi-Newton Methods for Function Minimization. *Mathematics of Computation* [online]. 1970, vol. 24, no. 111, pp. 647–656 [visited on 2025-04-12]. ISSN 00255718, ISSN 10886842. Available from DOI: 10.2307/2004840.

33. Liu, Dong C.; Nocedal, Jorge. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*. 1989, vol. 45, no. 1, pp. 503–528. ISSN 1436-4646. Available from DOI: 10.1007/BF01589116.

34. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*. 2019, vol. 378, pp. 686–707. ISSN 0021-9991. Available from DOI: 10.1016/j.jcp.2018.10.045.

35. Huang, Shenglin; He, Zequn; Chem, Bryan; Reina, Celia. Variational Onsager Neural Networks (VONNs): A thermodynamics-based variational learning strategy for non-equilibrium PDEs. *Journal of the Mechanics and Physics of Solids*. 2022, vol. 163, p. 104856. ISSN 0022-5096. Available from DOI: 10.1016/j.jmps.2022.104856.

36. Greydanus, Samuel; Dzamba, Misko; Yosinski, Jason. Hamiltonian Neural Networks. In: Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; Garnett, R. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019, vol. 32. Available also from: https://proceedings.neurips.cc/paper_files/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

37. Onsager, Lars. Reciprocal Relations in Irreversible Processes. I. *Phys. Rev.* 1931, vol. 37, pp. 405–426. Available from DOI: 10.1103/PhysRev.37.405.

38. Onsager, Lars. Reciprocal Relations in Irreversible Processes. II. *Phys. Rev.* 1931, vol. 38, pp. 2265–2279. Available from doi: 10.1103/PhysRev.38.2265.

39. Amos, Brandon; Xu, Lei; Kolter, J. Zico. Input Convex Neural Networks. In: Precup, Doina; Teh, Yee Whye (eds.). *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017, vol. 70, pp. 146–155. Proceedings of Machine Learning Research. Available from doi: 10.48550/arXiv.1609.07152.

40. Rudin, W. *Principles of Mathematical Analysis*. McGraw-Hill, 1953. International series in pure and applied mathematics. Available also from: https://books.google.cz/books?id=im8GAQAAIAAJ.

41. Qiu, Weilun; Huang, Shenglin; Reina, Celia. Bridging statistical mechanics and thermodynamics away from equilibrium: a data-driven approach for learning internal variables and their dynamics. 2025. Available from arXiv: 2501.17993 [cond-mat.stat-mech].

42. Rathore, Pratik; Lei, Weimu; Frangella, Zachary; Lu, Lu; Udell, Madeleine. Challenges in Training PINNs: A Loss Landscape Perspective. *CoRR*. 2024, vol. abs/2402.01868. Available from doi: 10.48550/ARXIV.2402.01868.

43. Harris, Charles R.; Millman, K. Jarrod; Walt, Stéfan J. van der; Gommers, Ralf; Virtanen, Pauli; Cournapeau, David; Wieser, Eric; Taylor, Julian; Berg, Sebastian; Smith, Nathaniel J.; Kern, Robert; Picus, Matti; Hoyer, Stephan; Kerkwijk, Marten H. van; Brett, Matthew; Haldane, Allan; Río, Jaime Fernández del; Wiebe, Mark; Peterson, Pearu; Gérard-Marchant, Pierre; Sheppard, Kevin; Reddy, Tyler; Weckesser, Warren; Abbasi, Hameer; Gohlke, Christoph; Oliphant, Travis E. Array programming with NumPy. *Nature*. 2020, vol. 585, no. 7825, pp. 357–362. Available from doi: 10.1038/s41586-020-2649-2.

44. Ansel, Jason; Yang, Edward; He, Horace; Gimelshein, Natalia; Jain, Animesh; Voznesensky, Michael; Bao, Bin; Bell, Peter; Berard, David; Burovski, Evgeni; Chauhan, Geeta; Chourdia, Anjali; Constable, Will; Desmaison, Alban; DeVito, Zachary; Ellison, Elias; Feng, Will; Gong, Jiong; Gschwind, Michael; Hirsh, Brian; Huang, Sherlock; Kalambarkar, Kshiteej; Kirsch, Laurent; Lazos, Michael; Lezcano, Mario; Liang, Yanbo; Liang, Jason; Lu, Yinghai; Luk, CK; Maher, Bert; Pan, Yunjie; Puhrsch, Christian; Reso, Matthias; Saroufim, Mark; Siraichi, Marcos Yukio; Suk, Helen; Suo, Michael; Tillet, Phil; Wang, Eikan; Wang, Xiaodong; Wen, William; Zhang, Shunting; Zhao, Xu; Zhou, Keren; Zou, Richard; Mathews, Ajit; Chanan, Gregory; Wu, Peng; Chintala, Soumith. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 2024. Available from doi: 10.1145/3620665.3640366.

45. Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, vol. 9, no. 3, pp. 90–95. Available from doi: 10.1109/MCSE.2007.55.

46. Garrett, John D. garrettj403/SciencePlots. 2021. Available from doi: 10.5281/zenodo.4106649.

47. Guldberg, C.M.; Waage, P. Concerning Chemical Affinity. *Erdmann's Journal für Practische Chemie*. 1879, vol. 127, no. 69-114.

48. Stein, E.M.; Shakarchi, R. *Fourier Analysis: An Introduction*. Princeton University Press, 2011. Princeton lectures in analysis. isbn 9781400831234. Available also from: https://books.google.cz/books?id=FAOc24bTfGkC.

List of Figures