UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Thesis

Solution of Schrödinger Equation for Quantum Systems via Physics-Informed Neural Networks

Internal supervisor, Università di Padova

Prof. Marco Baiesi

Candidate

Paolo Zinesi

External supervisors, Fraunhofer IISB

Philipp Brendel

Dr. Simon Mundinar

Academic Year 2022/2023

Abstract

The numerous successes achieved by machine learning techniques in many technical areas have sparked interest in the scientific community for their application in science. By merging the knowledge of machine learning experts and computational scientists, the field of scientific machine learning has shown its ability to greatly improve the performance of existing computational methods. One possible approach to developing physics-aware machine learning is the inclusion of physical constraints in the training of a machine learning model. Physics-Informed Neural Networks are an example of such an approach, as they can incorporate prior physical knowledge into their architecture, enabling them to learn and simulate complex phenomena while respecting the underlying physics principles. Possible constraints are physical laws, symmetries, and conservation laws. Compared to other machine learning models, Physics-Informed Neural Networks do not require substantial input data, with the exception of initial and boundary conditions to correctly formalize the problem.

In this Thesis, we exploit the advantages of Physics-Informed Neural Networks to efficiently simulate one-electron quantum systems. The simulations rely on the direct solution of the eigenvalue equation represented by the Schrödinger equation. Traditional methods for solving the Schrödinger equation often rely on approximations and can become computationally expensive for nontrivial systems. The mesh-free Physics-Informed Neural Networks approach avoids the need for discretization, as the residuals computed with respect to the physical constraints are minimized during training for a given set of points within the domain.

The solution of the Schrödinger equation allows one to calculate important physical quantities of the physical system under study, such as the ground state energy, the electronic wavefunction, and the associated electron density. These quantities are compared with the estimations present in the quantum chemistry literature to assess the performance of the Physics-Informed Machine Learning approach.

Contents

Introduction

Since the first formulation of quantum mechanics, our understanding of the fundamental nature of the physical universe has changed completely. The detailed study of the laws underlying our physical universe showed that the classical laws, formulated until the end of the 19th century, were not descriptive enough of the true behavior of matter at small scales. At microscopic scales, matter behaves according to laws that seem to go beyond common intuition. Superposition of states and entanglement are examples of such nonintuitive phenomena that are related to physical systems at small scales [1].

Despite the seemingly unusual nature of this new theory of quantum mechanics, scientists proposed different formalisms capable of describing the behavior of quantum systems. For example, Heisenberg matrix mechanics interpreted the physical properties of particles as matrices that evolve in time [2]. However, a crucial moment in the development of quantum mechanics came with the formulation of the Schrödinger equation by Erwin Schrödinger in 1925 [3]. This groundbreaking theory introduced a wave-based description of particles, replacing the classical deterministic view with one that embraced uncertainty and wave-particle duality. This description was no longer based on deterministic trajectories in the space-time domain, but rather on a field-like quantity known as the wavefunction of the system. Today, the Schrödinger equation is still a fundamental tool for analyzing microscopic phenomena and for predicting the evolution of quantum systems. The solution of the equation yields exactly the wavefunction of the physical system, from which prediction of physical observables can be performed straightforwardly. The theory has revolutionized fields such as chemistry, physics, and materials science, allowing for unprecedented accuracy in the prediction of the properties of the physical systems.

Theoretical physicists have spent a great deal of effort in finding analytical solutions to the Schrödinger equation for the most common quantum systems. However, closed-form analytical solutions cannot be found in systems of many interacting particles. This problem is called the *many-body problem* in quantum physics [4, 5] and is analogue to the *n-body problem* that arises in classical mechanics. In the absence of an analytical solution, scientists have designed computational methods to solve the Schrödinger equation for more complex quantum systems. Those computational methods require a large amount of computation time and memory given the difficulty of simulating interacting systems [6], which is also true beyond the physical sciences. In fact, the simulation of an interacting system cannot be simplified into the simulations of its components, since the correlations between the different components might have a strong impact on the overall solution. In quantum systems, in particular, by not considering the entanglement between different components of the system, the performances of the simulations are compromised. At the same time, the simulation of entangled systems is expensive. For these reasons, in the decades following the formulation of the Schrödinger equation, many optimized algorithms have been proposed to accurately and efficiently simulate these intricate systems. These algorithms span from simple mean-field approaches to more complicated approaches such as quantum Monte-Carlo (QMC) algorithms [7], Density Functional Theory (DFT) methods [8, 9], and Tensor Network methods [10, 11].

One novel approach to simulate quantum systems relies on the use of Neural Networks (NNs) to approximate the wavefunction of the physical system under study [12]. By transferring knowledge and optimized algorithms from the advanced field of Machine Learning (ML) [13], under some specific assumptions, the problem of finding the ground-state solution of the Schrödinger equation can be formulated as an unsupervised ML problem. The most common approach is to use the NN as a wavefunction ansatz and to compute the energy of the wavefunction as a function of the network parameters. This approach takes the name of Neural Network Quantum States (NNQS) [14] and is a particular example of *variational approach.* Therefore, gradient-based optimization routines can be used to find the best set of network parameters that minimize any given loss function. When the goal is to retrieve the ground-state wavefunction of the system, the energy represents a valid unsupervised loss, which minimization leads to the ground-state solution of the Schrödinger equation.

This thesis is devoted to the solution of the Schrödinger equation for quantum systems using Physics-Informed Neural Networks (PINNs), a class of NN function approximators that incorporate prior physical knowledge into their learning scheme [15, 16]. The wavefunction is parametrized using an artificial neural network, as in the NNQS approaches, but the training is performed following a physics-informed loss that depends on the Schrödinger equation. This loss encodes all the physical constraints of the system and is conceptually different from the energy loss of NNQS, because the PINN formulation does not depend on a problem-specific quantity as the ground-state energy of the system. Therefore, the PINN approach has the potential to generalize the current simulation methods beyond ground-state simulations, since the physics-informed loss does not make any assumption on the ground-state nature of the computed eigenvalue. Moreover, the physics-informed framework allows one to easily integrate prior knowledge or experimental data into the ML model, thus improving the performance of NNQS models.

The field of physics-informed learning promises to revolutionize and improve scientific computing thanks to ML techniques. On the one hand, PINNs improve traditional scientific computing approaches, which do not require input data, by avoiding the creation of expensive point meshes, typically used to discretize a function. PINNs compute derivatives using Automatic Differentiation (AD) [17], and therefore do not need the particular structure of point meshes to compute the desired derivatives. On the other hand, PINNs can improve data-driven ML solutions by taking into account physical constraints in the training process. The communication between the fields of scientific computing and machine learning generates a set of algorithms that improve the methods of both.

The thesis is structured as follows. In Chapter 1, ab initio methods to simulate quantum systems are briefly reviewed. A particular focus is given to the Variational Monte Carlo algorithms and their successful combination with machine learning variational classes. In Chapter 2, the framework of Physics-Informed Machine Learning is presented, describing the possible strategies to integrate specific physics-informed biases into a traditional machine learning problem. The specific PINN architecture and the loss components used to solve a Partial Differential Equation are then introduced. In Chapter 3, PINNs are applied to simulations of quantum systems. At first, previous physics-informed approaches to simulate simple systems are described and reviewed. Then, the architecture used in this thesis is explained in detail, and the different loss components are defined and motivated theoretically. Chapter 4 presents the results of physics-informed simulations of the hydrogen atom H and of the hydrogen molecule ion H_2^+. For the hydrogen atom case, in one and three dimensions, the simulations are first performed by fixing the eigenvalue variable to the analytical value. Then, this condition is relaxed, and no knowledge of the true eigenvalue is assumed. The simulations of the hydrogen molecule ion are, instead, performed only without fixing the eigenvalue variable, since no analytical solution for that system is available. In addition, some experiments are performed to assess the importance of the size of the neural network, the number of domain points, and the distribution of training points. Finally, in Chapter 5 the results are summarized and future developments discussed.

Chapter 1

Ab initio simulations of quantum systems

The physical properties of a quantum system are completely characterized by its wavefunction, which contains all the information about the state of a system and allows probabilistic predictions of the physical observables of interest [3, 18]. The wavefunction depends on the Hamiltonian of the system and is obtained by solving the Schrödinger equation. Theoretical researches conducted on the solution of the Schrödinger equation show that analytical solutions are available only for simple systems and simple Hamiltonians. Therefore, the solution of the Schrödinger equation for more complex systems requires numerical methods.

The focus of this thesis is on the study of quantum systems in their ground-state. In particular, simple atoms and molecules are the main systems of interest. At first, this chapter reviews the well-established quantum chemistry methods that are commonly employed to simulate such systems. The emergent ML approaches are then introduced. In this context, the joint use of the Variational Monte Carlo (VMC) algorithm and a NN wavefunction ansatz allows electronic wavefunction calculations that are significantly more accurate than VMC calculations using other approaches [14, 19].

1.1 Schrödinger equation

A system composed of N electrons and M ions is described by the Hamiltonian

$$\begin{aligned}\hat{H}_{\text{tot}} = &-\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \frac{1}{2}\sum_{I=1}^{M}\nabla_I^2 \\ &+\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{1}{|\mathbf{r}_i-\mathbf{r}_j|} - \sum_{I=1}^{M}\sum_{i=1}^{N}\frac{Z_I}{|\mathbf{r}_i-\mathbf{R}_I|} + \sum_{I=1}^{M}\sum_{J=I+1}^{M}\frac{Z_I Z_J}{|\mathbf{R}_I-\mathbf{R}_J|},\end{aligned} \tag{1.1}$$

where $\{\mathbf{r}_i\}_{i=1\ldots N}$ are the spatial coordinates of the electrons, $\{\mathbf{R}_I\}_{I=1\ldots M}$ are the spatial coordinates of the ions, and $\{Z_I\}_{I=1\ldots M}$ denotes the nuclear charges of the ions. Calculations are performed in the Born-Oppenheimer approximation [20], where nuclear positions are fixed input parameters, and Hartree atomic units are used. Therefore, the Hamiltonian $\hat{H}_{\text{tot}}$ of the whole system is simplified to the electronic Hamiltonian

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 + \sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{1}{|\mathbf{r}_i-\mathbf{r}_j|} - \sum_{I=1}^{M}\sum_{i=1}^{N}\frac{Z_I}{|\mathbf{r}_i-\mathbf{R}_I|}, \tag{1.2}$$

in which the coordinates $\{\mathbf{R}_I\}_{I=1\ldots M}$ are fixed and the ion-ion interaction terms have been traced out. The solution of the time-independent Schrödinger equation,

$$\hat{H}\,\Psi(\mathbf{x}_1,\ldots,\mathbf{x}_N) = E\,\Psi(\mathbf{x}_1,\ldots,\mathbf{x}_N), \tag{1.3}$$

gives as results the ground-state wavefunction $\Psi(\mathbf{x}_1,\ldots,\mathbf{x}_N)$ and the ground-state energy E. The coordinates $\mathbf{x}_i = \{\mathbf{r}_i,\sigma_i\}$ are defined as the combination of the spatial coordinates $\mathbf{r}_i \in \mathbb{R}^3$ and the spin variable $\sigma_i \in \{\uparrow,\downarrow\}$. Since the Hamiltonian is spin-independent, the wavefunction $\Psi(\mathbf{x}_1,\ldots,\mathbf{x}_N)$ does not depend on the spin variables. As $\hat{H}$ is Hermitian and time-reversal invariant, its eigenvalues and eigenfunctions are real. The wavefunction must be normalized, i.e., $\int d\mathbf{x}_1\cdots d\mathbf{x}_N\,|\Psi(\mathbf{x}_1,\ldots,\mathbf{x}_N)|^2 = 1$, and $|\Psi(\mathbf{x}_1,\ldots,\mathbf{x}_N)|^2$ is interpreted as the probability distribution of finding the electrons in the positions $(\mathbf{x}_1,\ldots,\mathbf{x}_N)$ when the system is measured.

Although the Hamiltonian is spin-independent, the electron spin does play a role in the simulations because the wavefunction must obey the Fermi-Dirac statistics. The wavefunction must be antisymmetric under the exchange of the coordinates of any pair of electrons,

$$\Psi(\ldots, \mathbf{x}_i, \ldots, \mathbf{x}_j, \ldots) = -\Psi(\ldots, \mathbf{x}_j, \ldots, \mathbf{x}_i, \ldots) \qquad \forall\, i \neq j, \tag{1.4}$$

and this constraint restricts the set of wavefunction ansätze. The desired antisymmetry is typically imposed by designing particular ansätze, such as the Slater determinants, in which the desired property is enforced as a hard constraint. However, soft constraint approaches can also be applied in those simulations that rely on the minimization of a cost function.

1.2 Hartee-Fock and post-Hartree-Fock methods

The Hartree-Fock (HF) method is the starting point of all quantum chemical calculations, as it is the simplest theory that incorporates the antisymmetry of the wavefunction [21]. The simplest wavefunction with the required antisymmetry is a Slater determinant,

$$D(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \phi_1(\mathbf{x}_1) & \cdots & \phi_1(\mathbf{x}_N) \\ \vdots & & \vdots \\ \phi_N(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_N) \end{vmatrix}, \tag{1.5}$$

where $\phi_1, \ldots, \phi_N$ are one-electron orbitals, usually assumed to be products of spatial and spin factors, $\phi_i(\mathbf{x}_j) = \phi_i(\mathbf{r}_j)\delta_{\sigma_i,\sigma_j}$. The HF method uses a single Slater determinant as a variational trial function, and this trial function is optimized by minimizing the expectation value of $\hat{H}$ with respect to the orbitals $\phi_i(\mathbf{r}_j)$. With this approach, a set of coupled self-consistent equations is obtained for the orbitals ϕ_i [22].

The HF method is able to retrieve the exchange effects arising from the antisymmetry of the many-electron wavefunction, but relies on the strong approximation of neglecting the correlations caused by the electron-electron Coulomb repulsion. Even if the electron correlation energies only account for a fraction of the total energy of the system, they can lead to large deviations from the experimental measurements. Post-Hartree-Fock methods propose different strategies to include this correlation energy, improving the mean-field assumption of the HF method. Examples of these methods are the Møller–Plesset perturbation theory (MP), in which the energy correlations are treated as a small perturbation of the total energy, and configuration interaction (CI) methods, which use a linear combination of Slater determinants as a variational ansatz for the true many-electron wavefunction. The CI is expensive in general, as very large numbers of determinants are needed to describe the wavefunction accurately. In fact, the number of required determinants grows exponentially with the system size, and thus the full CI (FCI) method can be applied effectively only to small molecules. Moreover, many determinants are needed to describe the gradient discontinuities that arise when two electrons are in the same position. A possibility to reduce the number of determinants is to choose only the most important ones, i.e., the determinants with the largest overlaps with the true ground state wavefunction. These determinants are typically chosen from the low-energy excitations of the reference HF determinant (i.e., of the HF wavefunction), but also other wavefunctions can be used. A more advanced approach is the coupled cluster (CC) method, which implicitly includes all excitations of the reference wavefunction by applying the exponential of a cluster operator T to the reference wavefunction. The choice of the cluster operator T is crucial to guarantee the size consistency of the solution. Coupled cluster methods are capable of retrieving accurate results, but they are still very expensive in large systems. For example, a CC calculation with single and double excitations (i.e., CCSD) scales as N^6.

1.3 Variational Monte Carlo

An alternative approach to model many-body wavefunctions is given by the large family of Quantum Monte Carlo (QMC) algorithms. Algorithms belonging to this family are diverse and propose different methods to solve the many-body problem, but they all rely on the use of Monte Carlo integration techniques to handle the multi-dimensional integrals. We focus here on the specific Variational Monte Carlo algorithm (VMC), as it is particularly effective in solving the many-body problem when neural networks are used as a wavefunction ansatz.

In VMC, the first step is to define a wavefunction ansatz, $\Psi_{\boldsymbol{\theta}}(\mathbf{x}_1, \dots, \mathbf{x}_N)$, that depends on the parameters $\boldsymbol{\theta}$ [23]. The expected energy of the system is given by the Rayleigh quotient:

$$E_{\boldsymbol{\theta}} = \frac{\langle \Psi_{\boldsymbol{\theta}} | \hat{H} | \Psi_{\boldsymbol{\theta}} \rangle}{\langle \Psi_{\boldsymbol{\theta}}^2 \rangle} = \mathbb{E}_{\mathbf{x} \sim \Psi_{\boldsymbol{\theta}}^2} \left[\Psi_{\boldsymbol{\theta}}(\mathbf{x})^{-1} \hat{H} \, \Psi_{\boldsymbol{\theta}}(\mathbf{x}) \right] = \mathbb{E}_{\mathbf{x} \sim \Psi_{\boldsymbol{\theta}}^2} \left[E_L(\mathbf{x}) \right], \tag{1.6}$$

in which the shorthand $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ is used for clarity and the symbol $\mathbb{E}_{\mathbf{x} \sim \Psi_{\boldsymbol{\theta}}^2}$ denotes the average over points sampled from the distribution $\Psi_{\boldsymbol{\theta}}^2$. The term $E_L(\mathbf{x}) = \Psi(\mathbf{x})^{-1} \hat{H} \, \Psi(\mathbf{x})$ is known as the *local energy*. In other words, the expected energy is the average of the local energy $E_L(\mathbf{x})$. Samples from the distribution proportional to $\Psi_{\boldsymbol{\theta}}^2$ are generated using Monte Carlo methods, and with these samples an estimation of $E_{\boldsymbol{\theta}}$ is computed for a given $\boldsymbol{\theta}$. This estimation can be used to optimize the ansatz with respect to the parameters $\boldsymbol{\theta}$ to ultimately find the best approximation of the ground-state wavefunction.

Naturally, the key component is the choice of the wavefunction parameterization, since the evaluation of the wavefunction and its derivatives is the most computationally expensive task. In quantum chemistry problems, trial wavefunctions are commonly chosen within the Slater-Jastrow ansatz [21, 24],

$$\Psi(\mathbf{x}) = e^{J(\mathbf{x})} \sum_k D_k(\mathbf{x}), \tag{1.7}$$

which takes a truncated linear combination of Slater determinants, $\sum_k D_k$, and multiplies it by a Jastrow factor, $e^{J(\mathbf{x})}$. Each determinant D_k is calculated using a set of one-particle orbitals $\{\phi_1^k, \dots, \phi_N^k\}$, that is a subset of length N of a larger set of one-particle orbitals $\{\phi_1, \dots, \phi_{N'}\}$, with $N' \geq N$. Instead, the multiplicative Jastrow factor $e^{J(\mathbf{x})}$ captures close-range correlations and includes the cusps that arise in the equations when two electrons are in the same position. The form of the $J(\mathbf{x})$ function depends on the desired field of application and on the order of correlations of the electron-electron and electron-nuclear interactions to be considered. The Jastrow factor is symmetric and non-negative, and thus the total wavefunction is still antisymmetric. An additional improvement to this ansatz is the application of a backflow transformation prior to orbital evaluation, which changes the position of every electron by an amount that depends on the position of nearby electrons [25].

The accuracy provided by the Slater-Jastrow-backflow ansatz allows it to be nowadays the default ansatz for many-electron problems in three dimensions.

1.3.1 Variational Monte Carlo and machine learning

By exploiting the recent advances of ML and deep learning techniques, in 2017 Carleo and Troyer combined the VMC algorithm with an artificial neural network ansatz for the wavefunction [14]. Quantum states defined by this variational class are called Neural Network Quantum States (NNQS). Key advantages over previous approaches is the theoretical convergence guarantee of NN approximations and the possibility to increase the accuracy of the approximation by increasing the number of trainable parameters. The original work of Carleo and Troyer is focused on the application of NNQS to one- and two-dimensional spin systems, but successive studies have highlighted the possibility to apply this approach also to the many-electron Schrödinger equation [19, 26, 27]. The principal achievement of these studies is the explicit encoding of the fermionic symmetry inside the wavefunction ansatz, enabling electronic structure calculations that are significantly more accurate than other VMC calculations.

In particular, the Fermionic Neural Network, or *FermiNet* [19], ansatz obtains a remarkable advantage over previous VMC approaches by allowing a greater flexibility in the choice of orbitals that appear in the Slater determinants (see the $\phi_i(\mathbf{x}_j)$ orbitals of Eq. (1.5)). In particular, more general electron-electron interactions can be encoded in the ansatz by generalizing the single-particle orbitals, $\phi_i(\mathbf{x}_j)$, to orbitals that depend on the coordinates of all the electrons, $\phi_i(\mathbf{x}_j; \{\mathbf{x}_{/j}\})$. The notation $\{\mathbf{x}_{/j}\}$ indicates the set of all the coordinates except $\mathbf{x}_j$. Of course, the orbitals $\phi_i(\mathbf{x}_j; \{\mathbf{x}_{/j}\})$ have to be invariant to any change in the order of the variables in $\{\mathbf{x}_{/j}\}$ to conserve the antisymmetry of the total wavefunction Ψ. The innovation of FermiNet is the construction of a set of functions that satisfy this permutation invariance using neural networks. Remarkably, the orbitals obtained with this procedure allow to introduce arbitrary correlations between electrons in the many-body wavefunction.

Other studies have successively focused on the improvement of the FermiNet results. For example, the PauliNet ansatz is designed to incorporate the multireference HF method as a baseline [28]. In PauliNet, the ad-hoc functional forms previously used for the Jastrow factor and the backflow transformations are replaced by NN representations. For this reason, the convergence of PauliNet is more robust and the iterations are faster than in the FermiNet. More recent approaches, such as the Psiformer [23] and the QiankunNet [29], propose to exploit the recent success of transformer architectures in natural language processing tasks to capture the electron-electron correlations more efficiently and without imposing a fixed functional form.

Chapter 2

Physics-Informed Machine Learning

Algorithms able to combine data-driven learning with prior knowledge on the properties and symmetries of the problem are not new in the field of machine learning. By designing problem-specific network architectures and optimization algorithms, model training can be drastically simplified, both in terms of computational time and memory resources. In addition, the embedding of prior information reduces the amount of data needed during training. One notable example of ML architecture that simplifies the training procedure by exploiting the features of the input data is the Convolutional Neural Network (CNN) architecture [30–32]. In CNNs, the highly-overparametrized architecture of Fully Connected Neural Networks (FCNNs) is substituted by an architecture that optimizes a set of filters (or kernels) that slide along input features. When kernels are slid along the input features, they provide translation-equivariant responses known as feature maps. Then, feature maps are usually down-sampled and the described process is iterated according to the network architecture. During the training step, CNNs recognize patterns present in the training dataset and store them in the kernel weights, and during the evaluation step those patterns are used to scan new input data. The sliding mechanism of the kernels allows CNNs to spot a pattern also in a different position with respect to the position in the training dataset, whereas the FCNNs require the exact matching of the pattern positions in the training data. Therefore, despite the fact that exact translational invariance is not achieved in CNNs because of the down-sampling transformations, invariance to input features' translations is retrieved in practice. Therefore, input data can be analyzed more efficiently by neglecting redundant features arising from symmetry considerations.

The CNN example shows in practice how a well-known symmetry can positively impact a learning model if that symmetry is taken into account in the architecture of the model. Convolutional Neural Networks exploit what is called an *inductive bias* to simplify the model. In this chapter, after a general introduction of the possible ways to induce a bias in physics-informed learning, we focus on the particular case of Physics-Informed Neural Networks that employ a different type of bias, the *learning bias*, to simulate dynamical systems in the small data regime.

2.1 Biases

Every predictive model needs some assumptions to allow generalization, as in the absence of those assumptions the model tends to reproduce exactly the input data. This behavior is a well-known issue in the field of machine learning and is commonly known as *overfitting* [33]. This issue usually arises when the complexity of the model that is being learned is not supported by a sufficient amount of training data. The predictions of an overfitted model poorly generalize to input data not belonging to the training set, as the intricate correlations of data cannot be approximated by a model that tends to reproduce the training data.

In order to reduce overfitting and accelerate training, biases need to be included in the architecture and in the training of ML models. In physics-informed learning, three types of bias can be recognized [16].

- **Observational biases**. This is the most common type of bias and is responsible for the recent success of ML. An observational bias is enforced in an ML model by providing enough data to cover the input domain of a given task. Thanks to the constant growth of sensor networks able to gather data at different spatio-temporal scales, we are able to gather a consistent amount of data and to use it to develop increasingly accurate models. To effectively bias learning, training data should reflect the underlying physical principles that dictate their generation. Observed data can indirectly encode these principles into ML models, provided that a large amount of data is available and that these data are representative of the whole input domain. However, observational data might be generated by expensive experiments or large-scale computations, and consequently, the acquisition of a consistent amount of data might be infeasible. This issue limits the advantages of purely data-driven approaches in those tasks and requires one to consider additional sources of bias to successfully train an ML model.

- **Inductive biases**. Another possibility to encode prior knowledge into ML models is to design NN architectures that implicitly embed knowledge of the system. As already discussed above, CNNs are the most prominent example of the effectiveness of inductive biases, but there exists a multitude of architectures that apply the same principle, such as graph neural networks [34] and equivariant networks [35] . These architectures can be generalized to satisfy more general symmetries, e.g., rotations, reflections, and more general gauge symmetries. In the study of Hamiltonian systems, specific architectures can be designed to preserve their symplectic structure [36]. In quantum many-body systems, the anti-symmetry on the exchange of input variables can be hard coded in the network by transforming the input variables using a determinant of a matrix-valued function [19]. In the solution of differential equations, input variables can be transformed to satisfy initial conditions and boundary conditions [37, 38]. Despite their conceptual simplicity, inductive biases can be effectively implemented only for a limited set of simple and well-defined symmetries, while implicitly encoding more complicated symmetries or conservation laws is still a difficult task.
- **Learning bias**. Prior information can also be embedded in the learning process by penalizing solutions that do not satisfy some constraints [15, 39, 40]. This strategy aims to impose those constraints in a soft manner by adding a loss that depends on the deviations from the constraints. The network is trained by minimizing a loss that is composed of different terms, each of them associated to a particular condition that has to

be satisfied. There is a great flexibility in the choice of soft penalty constraints, and therefore the soft constraint is a good candidate to impose symmetries and conservation laws that cannot be easily implemented with an inductive bias. However, this advantage comes at the cost of an increased training complexity. The trained ML model is, in fact, a compromise between all the loss terms that have to be minimized simultaneously. If a solution that simultaneously satisfies all these constraints does not exist, then the trained model will depend on the entity of each loss component during the minimization of the loss. This issue is traditionally tackled by tuning the weights associated with each loss component and by computing the final loss as a weighted loss. This bias is at the core of a very important class of physics-informed models, the Physics-Informed Neural Networks (PINNs) [15], to which Section 2.2 is dedicated.

In general, good performances can be obtained when a combination of the three aforementioned biases is used. For instance, a dynamical system might be simulated by combining knowledge of the system at certain space-time instances with partial information regarding the equations that govern its dynamics. Alternatively, the evolution of a system governed by a well-known Partial Differential Equation (PDE) might be simulated only by knowing the initial and boundary conditions of the PDE, without requiring any observational data. Physics-informed ML is able to unify under a common framework both the cases of completely known physical laws in the small-data regime and of partially known physical laws in the big-data regime.

To simplify the learning process, all the prior information available must be encoded in the training procedure. Examples of such prior knowledge are

- Equations (e.g., trajectories, kinematics, and other laws).
- Differential equations (e.g., ordinary/partial/stochastic differential equations).
- Physical constraints (e.g., conservation of energy/momentum/-mass/probability, boundary/surface conditions, positive definition of certain variables).

- Symmetries (e.g., translations/rotations equivariance, many-body symmetries, and other group symmetries).

2.2 Physics-Informed Neural Networks

After having introduced the general framework of physics-informed learning, the particular architecture of PINNs is analyzed [15, 16], and the components of learning are delineated. The PINN structure is particularly suited to infer deterministic functions compatible with an underlying physical law when a limited number of observations is available. In particular, Multi-Layer Perceptron (MLP) architectures can be applied to different problems because they do not incorporate any specialized inductive biases. Other architectures, e.g., CNN or Fourier Feature Networks, can be applied when a specialized inductive bias is desired. In the following, the term PINNs is improperly used to refer to the particular architecture of physics-informed MLP, which is the main focus of this thesis.

2.2.1 Mathematical formulation

Consider a PDE of the following form

$$\mathcal{N}[u](\mathbf{x}) = f(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{2.1}$$

$$\mathcal{B}[u](\mathbf{x}) = g(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega, \tag{2.2}$$

where $\mathcal{N}[\cdot]$ is a differential operator acting on the domain $\Omega \subset \mathbb{R}^n$ and $\mathcal{B}[\cdot]$ formalizes the boundary conditions in $\partial\Omega$. The function $u : \overline{\Omega} \longrightarrow \mathbb{R}$ is the quantity of interest that is governed by Eq. (2.1)-(2.2). The formalism can also consider vector-valued functions for $u(\mathbf{x})$, but only the real-valued case is presented for simplicity. Time-dependent problems can be incorporated into this formalism by considering time as a special component of $\mathbf{x}$ and by treating the initial conditions as a special kind of boundary condition. The function $u(\mathbf{x})$ is approximated by a neural network $u_{\boldsymbol{\theta}}(\mathbf{x})$, where $\boldsymbol{\theta}$ defines the parameters of the network that are updated during the training phase.

2.2.2 Losses

After the initialization of the network parameters $\boldsymbol{\theta}$, the physics-informed model is trained by minimizing the loss function $\mathcal{L}(\boldsymbol{\theta})$. When Eq. (2.1)-(2.2) are the only training data available for the solution of the PDE, the loss function is

$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\mathcal{N}}\,\mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta}) + \omega_{\mathcal{B}}\,\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}), \tag{2.3}$$

where $\mathcal{L}_{\mathcal{N}}$ and $\mathcal{L}_{\mathcal{B}}$ express the residuals of Eq. (2.1) and (2.2), respectively, as a function of $\boldsymbol{\theta}$. More precisely,

$$\mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta}) = \frac{1}{N_f}\sum_{i=1}^{N_f}\left[\mathcal{N}[u_{\boldsymbol{\theta}}](\mathbf{x}_i^f) - f(\mathbf{x}_i^f)\right]^2, \qquad \{\mathbf{x}_i^f\}_{i=1\dots N_f} \in \Omega, \tag{2.4}$$

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{N_g}\sum_{i=1}^{N_g}\left[\mathcal{B}[u_{\boldsymbol{\theta}}](\mathbf{x}_i^g) - g(\mathbf{x}_i^g)\right]^2, \qquad \{\mathbf{x}_i^g\}_{i=1\dots N_g} \in \partial\Omega, \tag{2.5}$$

where $\{\mathbf{x}_i^f\}_{i=1\dots N_f}$ and $\{\mathbf{x}_i^g\}_{i=1\dots N_g}$ are batches of training points belonging to the computational domain. These points are randomly sampled within the computational domain to enforce a uniform approximation of $u(\mathbf{x})$ in the domain and prevent overfitting. The loss weights $\omega_{\mathcal{N}}, \omega_{\mathcal{B}}$ define the relative importance of each loss component.

Other information available on the PDE solution can be enforced as additional terms in $\mathcal{L}(\boldsymbol{\theta})$. For instance, a regularization loss $\mathcal{L}_{\text{reg}}(\boldsymbol{\theta})$ or a supervised loss $\mathcal{L}_u(\boldsymbol{\theta})$ can be added to the total loss with their respective weights $\omega_{\text{reg}}, \omega_u$. While the supervised loss is conceptually similar to the PDE losses of Eq. (2.4)-(2.5), it is different from the latters because the supervised loss of $u(\mathbf{x})$ relies on the presence of a dataset $\{\mathbf{x}_i^u, u_i\}_{i=1\dots N_u}$ of the true solution. The supervised loss can be expressed as

$$\mathcal{L}_u(\boldsymbol{\theta}) = \frac{1}{N_u}\sum_{i=1}^{N_u}\left[u_{\boldsymbol{\theta}}(\mathbf{x}_i^u) - u_i\right]^2, \tag{2.6}$$

and the total loss is defined as a weighted sum of all the loss terms,

$$\mathcal{L}(\boldsymbol{\theta}) = \omega_{\mathcal{N}}\,\mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta}) + \omega_{\mathcal{B}}\,\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \omega_u\,\mathcal{L}_u(\boldsymbol{\theta}) + \omega_{\text{reg}}\,\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}). \tag{2.7}$$

All the previous losses have been defined using the mean-squared error (MSE) metric to define the contribution of each residual point to the loss, but other metrics such as the mean-absolute error (MAE) metric can be used as well.

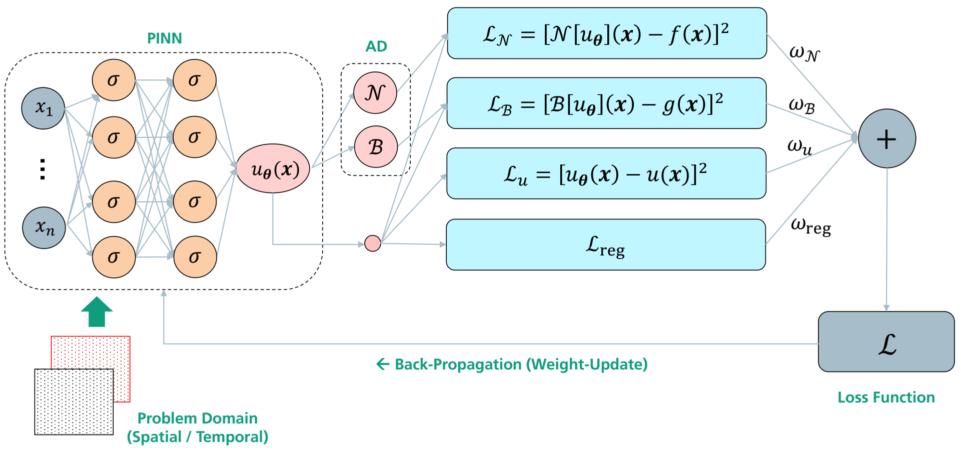

PINN
x_1
x_n
σ
$u_\theta(x)$
AD
$\mathcal{N}$
$\mathcal{B}$
$\mathcal{L}_\mathcal{N} = [\mathcal{N}[u_\theta](x) - f(x)]^2$
$\mathcal{L}_\mathcal{B} = [\mathcal{B}[u_\theta](x) - g(x)]^2$
$\mathcal{L}_u = [u_\theta(x) - u(x)]^2$
$\mathcal{L}_{\mathrm{reg}}$
$\omega_\mathcal{N}$
$\omega_\mathcal{B}$
ω_u
ω_{reg}
+
$\mathcal{L}$
← Back-Propagation (Weight-Update)
Loss Function
Problem Domain (Spatial / Temporal)

Figure 2.1: Scheme of a PINN

2.2.3 Network

The core of the PINN is a fully connected neural network that takes as input the components of $\mathbf{x}$, $\{x_1, \ldots, x_n\}$, and returns the function $u_{\boldsymbol{\theta}}$ evaluated at $\mathbf{x}$. A schematic view of the network is presented in the dashed "PINN" rectangle of Fig. 2.1.

The approximation capabilities of the network depend on the size of the network, since a larger network stores more parameters in the weights of layer-to-layer connections. Complex functions need large networks to properly approximate them, but training large networks requires a great deal of computational resources. Each problem has its own ideal network size. Small networks give rise to underparametrized models, which fail to capture the complex structure of the underlying function, whereas large networks generate models that are too expensive to be trained. The trade-off between these two extreme scenarios is the ideal network size. Current researches on PINNs use shallow networks of a few hidden layers with approximately a hundred of neurons in each layer [41].

Another feature of the network that impacts the learning process is the activation function. In ML, the activation function has the important role of introducing nonlinearities in the NN model. The choice of activation function is crucial for the good performance of the NN model because gradients are propagated in different ways through different activation functions. In traditional ML problems, activation functions are chosen to be fast to compute and to avoid problems such as vanishing or exploding gradients. The Rectifier Linear Unit (ReLU) is one of the most used activation functions, and it can be computed with a simple comparison operator [42]. However, most of the activation functions used in traditional ML problems are not infinitely differentiable, as they usually have a point of non-differentiability at the origin. Physics-informed ML models comprehend situations in which higher-order derivatives are required, and therefore the activation functions have to be differentiable infinitely many times. Typical activation functions used in PINNs are the hyperbolic tangent (tanh), the sigmoid, and the Sigmoid Linear Unit (SiLU, also known as *swish*) [43].

The output layer of the NN might also be transformed to satisfy hard constraints or to simplify the convergence of the network.

2.2.4 Training

Once the architecture of the NN and the loss function have been defined, the PINN is trained by minimizing the loss function $\mathcal{L}$ as a function of the NN parameters $\boldsymbol{\theta}$. The update of the NN parameters can be written as

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} - \eta \, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{2.8}$$

where $\eta > 0$ is the learning rate and the partial derivative with respect to the components of $\boldsymbol{\theta}$ is computed efficiently using AD. Usually, this update rule is performed by advanced and efficient optimization algorithms that can quickly converge to the optimal solution without getting trapped in local minima. The Adam optimizer is typically used in modern ML applications [44]. The convergence properties of the training procedure can also be tuned by decreasing the learning rate η during training. This learning rate decrease allows the optimization to asymptotically stabilize and is a good method to prevent continual fluctuations around an optimal point. However, the learning rate has to be decreased slowly to allow the optimizer to completely explore the parameter space. An example of scheduled learning rate decrease is the inverse time decay, $\eta(s) \sim 1/s$, for which $\sum_s \eta(s) = +\infty$ and therefore all the parameter space can be reached.

Since the PINN approach comprehends both data-driven and data-free problems, a strategy is necessary to autonomously generate training data. For the solution of PDEs, inner and boundary points can be sampled, respectively, inside the domains Ω and $\partial\Omega$, and the output of the PINN is evaluated at these points. Automatic Differentiation techniques allow the computation of derivatives of arbitrary order with respect to the input variables $\mathbf{x}$. Unlike traditional mesh-based approaches, training data can be randomly sampled inside the domains following a given distribution. Therefore, the PINN method is compatible with non-uniformly distributed training data, since the distribution of training points does not impact the computation of the derivatives with AD. Thanks to the intrinsic randomness of the training set, the training points can be resampled during the training in order to reduce overfitting and to speed up convergence, improving generalizability.

2.3 Motivations and advantages

This novel approach improves considerably many aspects of modern ML techniques.

First, as already stated in the preceding sections, combining prior knowledge and data allows one to constrain the learning process to a lower-dimensional manifold of possible models, requiring less training data with respect to a purely data-driven approach. This aspect brings consistent advantages when data are scarce or absent in a particular range of inputs. Although traditional ML models retrieve excellent interpolation performances with an adequate training set, their performance usually drops when the model is evaluated outside the domain of training data. Embedding the physical principles can help the ML model to generalize better and to succeed in both interpolation and extrapolation tasks.

Second, physics-informed ML can efficiently approximate the solution of both direct and inverse problems by exploiting partial prior information and noisy or incomplete data. The trustworthiness of training data is fundamental in traditional ML approaches, in which uncertainties that affect the data are also transferred to the final model. Instead, physics-informed ML models can combine incomplete prior knowledge and imperfect data to outperform existing numerical grid-based methods for ill-posed direct problems and inverse problems [45], whereas this advantage is less prominent in well-posed problems that do not require any data assimilation.

Third, physics-informed learning can exploit highly optimized routines and Automatic Differentiation techniques, contained in specialized libraries such as TensorFlow [46] and PyTorch [47], in the field of scientific computing, with consequent performance advantages. Fundamental connections between ML and scientific computing can also be found at the theoretical level. Interestingly, most successful architectures in ML are analogous to established numerical algorithms. For example, CNNs are analogous to finite difference methods in translationally equivariant PDE discretizations [48], while recurrent NN architectures are analogous to Runge-Kutta schemes [49]. The discovery of such analogies can potentially be a spark in the implementation of new ML architectures, informed by long-standing traditional computational schemes.

Finally, the ability of ML to tackle high-dimensional problems can solve the curse of dimensionality that characterizes traditional computational methods. In fact, the precision of mesh-based computational approaches (e.g., finite-difference methods for solving PDEs) decreases for high dimensions because of the exponential increase in the number of function evaluation points. In ML, mesh-free approaches can be designed instead, thanks to the ability of computing derivatives with respect to any point in the domain space using Automatic Differentiation. Meshless approaches allow for great flexibility in the choice of domain points distribution along each dimension, potentially favoring some dimensions over others in the domain space. Alternatively, a recent work by Hu et al. presents a strategy to tackle the curse of dimensionality directly at the backpropagation step, by updating at each step only a selection of dimensions in order to speed up convergence and to avoid the instantiation of high-dimensional points [50].

2.4 Limitations

The capabilities of physics-informed learning techniques are accompanied by certain limitations that need to be taken into account to design and train effective physics-informed models.

Most importantly, training a physics-informed ML model is a difficult task because of the complicated structure of the loss. When the loss is composed of many terms that have to be minimized simultaneously, the total loss becomes highly non-convex, and its minimization is complicated by the different loss components competing with each other. The weights of each loss component must be chosen accurately to avoid the predominance of one loss component over the others [51]. Without this preliminary step, the training algorithm might focus only on minimizing the largest loss component. The problem can be mitigated by designing appropriate model architectures and new training algorithms specific to these non-convex minimization problems. However, the current physics-informed learning approach requires users to design effective NN architectures and to tune most of the hyperparameters empirically, which are very time-consuming processes.

Training a physics-informed ML model is not only a difficult task due to the complex structure of the loss landscape, but is also computationally expensive. Deep networks can store a large amount of model parameters, which are updated at each training step by estimating the gradient of the loss with respect to every parameter. This set of operations is at the core of the success of ML and the speed up of those operations is therefore crucial to enhance the performances of ML architectures. Specialized hardware such as GPUs and TPUs can drastically reduce the training time of some specific architectures, but for large physics and engineering problems the computational resources are still inadequate to solve ML problems of high complexity.

Furthermore, in the current stage, traditional computational techniques still outperform physics-informed ML in some specific tasks, such as PDE diffusion problems with high convection or viscosity parameters [52]. More generally, using physics-informed ML as a drop-in replacement for traditional methods does not always lead to better performance, and a more careful preliminary study of the physical problem is needed to improve the quality of the physics-informed ML model [53, 54].

Moreover, the performance of physics-informed ML is strongly affected by the training time [54]. Physics-informed ML models are convenient for tasks that allow the exploitation of the fast inference times, while they are of limited use when a relative small amount of evaluations is needed.

2.5 Software

The efficient implementation of PINNs requires highly optimized routines to run the most common and expensive operations, such as backpropagation of gradients. At present, there are a variety of ML libraries, like TensorFlow [46], PyTorch [47], Keras [55], and JAX [56], that offer efficient implementations of the most common ML functions. On top of those libraries, also specific physics-informed ML libraries have been developed to exploit effectively the low-level routines of ML libraries, while providing additional features that are useful in the field of physics-informed learning. Some of the libraries developed so far include DeepXDE [57], SimNet [58], NeuroDiffEq [59], and ADCME [60]. Most of the libraries for physics-informed ML are written in Python, as it is the dominant programming language in ML, but some others are written in Julia. Libraries such as DeepXDE, SimNet, and NeuroDiffEq can be used as solvers, i.e., users only need to define the problem, and the libraries manage the implementation details in the background. Alternatively, libraries such as ADCME only wrap low-level functions of ML libraries into high-level functions, and users have to implement the solution of the physics-informed problem autonomously. Regardless of the details, the main task of these libraries is to provide utilities to access AD algorithms contained in specialized software such as TensorFlow.

Chapter 3

Quantum systems simulations with physics-informed neural networks

This section presents different methods for simulating quantum systems with physics-informed approaches. After a first review of existing approaches, the PINN architecture employed in this thesis is explained in all its components.

3.1 Literature review of existing physics-informed approaches

In Sect. 1, a first ML approach to solve the Schrödinger equation for quantum systems is formulated using a VMC algorithm. An alternative approach to the solution of the many-body Schrödinger equation consists in reformulating the objective function to be minimized in terms of a physics-informed loss of the kind of Eq. (2.7). The physics-informed method is still a variational approach, since the trial wavefunction belongs to the class of functions parametrized by an artificial neural network. However, the physics-informed loss $\mathcal{L}(\boldsymbol{\theta})$ differs from the typical VMC objective function because $\mathcal{L}(\boldsymbol{\theta})$ is a quantity that depends on the residuals of the Schrödinger equation. To be more precise, the loss term $\mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta})$ introduced in Eq. (2.4) is evaluated as

$$\mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta}) = \frac{1}{N_f} \sum_{i=1}^{N_f} \left[\hat{H}\, \Psi_{\boldsymbol{\theta}}(\mathbf{r}^i) - E\, \Psi_{\boldsymbol{\theta}}(\mathbf{r}^i) \right]^2 , \tag{3.1}$$

where the $3N$-dimensional points $\{\mathbf{r}^i\}_{i=1\dots N_f}$ are sampled inside the computational domain. Although PINNs have demonstrated remarkable results in the solution of PDEs, the application of physics-informed algorithms to eigenvalue problems has received little attention until now. The necessity of simultaneously computing the eigenvalue E and the eigenfunction $\Psi_{\boldsymbol{\theta}}$ complicates the training procedure, and an extension of the traditional physics-informed approach is required to efficiently compute the eigenvalue E.

The first demonstration of the possibility of solving eigenvalue problems with neural networks and physics-informed loss is the work of Lagaris et al. [12]. In that paper, the authors use a neural network ansatz for the trial wavefunction and define a loss function proportional to Eq. (3.1), although the concept of physics-informed learning was not yet formulated at the time the paper was published. Differently from the definition in Eq. (3.1), however, the loss function of [12] is divided by the squared norm of the wavefunction, $||\Psi_{\boldsymbol{\theta}}||^2 = \int_\Omega d^3\mathbf{r}\ |\Psi_{\boldsymbol{\theta}}(\mathbf{r})|^2$, where the integral is computed over the computational domain Ω. This loss function can be interpreted as the average of a mean squared error operator for a quantum system described by an unnormalized wavefunction $\Psi_{\boldsymbol{\theta}}$, and therefore the loss definition is physically motivated. However, this loss requires the calculation of two integrals at every iteration, one for the squared norm of the wavefunction and one for the calculation of the eigenvalue E as the average value of the Hamiltonian operator. Lagaris et al. [12] simplify the calculations by eliminating the dependence of these integrals on the parameters $\boldsymbol{\theta}$ that define a given wavefunction, but in principle also the contribution of these integrals have to be taken into account in the backpropagation of the gradients. Despite the simplicity of the approach and the use of a single hidden layer in the MLP architecture, this algorithm is able to efficiently retrieve eigenvalues and eigenfunctions for single-particle problems. The approach can also be easily extended to estimate energies above the ground state.

The idea of using neural networks to solve the eigenvalue problem in quantum mechanics has been proposed again recently by Jin et al. [61] in the novel framework of PINNs. In that paper, the authors propose the architecture shown in Fig. 3.1 to compute the wavefunction $\Psi_{\boldsymbol{\theta}}$ and the eigenvalue E simultaneously. Similarly to the work of Lagaris et al. [12], the output of the

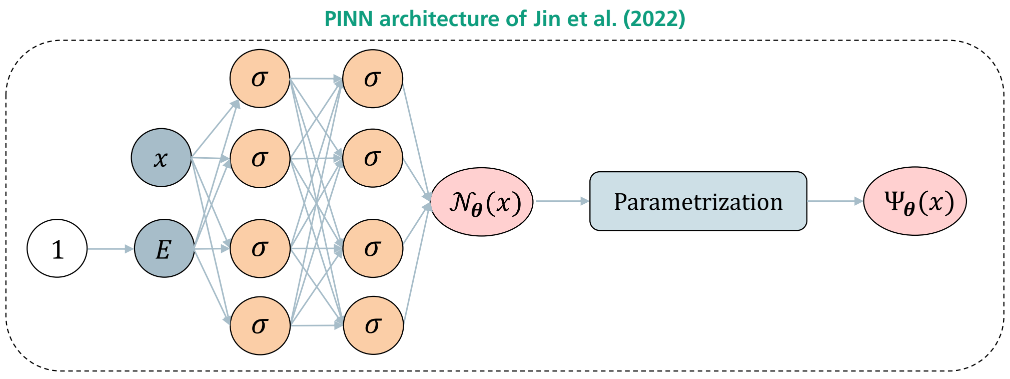

PINN architecture of Jin et al. (2022)
1
x
E
σ
$\mathcal{N}_{\boldsymbol{\theta}}(x)$
Parametrization
$\Psi_{\boldsymbol{\theta}}(x)$

Figure 3.1: PINN architecture for solving eigenvalue problems in [61, Figure 1] (adapted).

MLP is the desired wavefunction. More specifically, the eigenvalue is generated from an affine transformation (i.e., a linear transformation composed with the addition of a bias term) of a constant input of ones and then fed to the PINN together with the input data x. The parameters of the affine transformation determine the eigenvalue and are updated during training. With this approach, the eigenvalue is computed very quickly at each iteration step and the expensive evaluation of an integral is avoided. The loss function designed by Jin et al. [61] is composed of a PDE loss, a normalization loss, and an orthogonality loss. While the PDE loss is comparable to the loss of Eq. (2.4), the other two losses are regularization terms that simplify convergence. The orthogonality loss, in particular, is useful in determining new eigenfunctions that are orthogonal to the eigenfunctions already discovered. Although applied only to one-dimensional systems, this algorithm shows good performance in discovering eigenvalues, and the authors suggest the possibility of implementing the same algorithm in more complicated systems.

In a successive article, the architecture of [61] is expanded to simulate the H_2^+ molecule as a function of the coordinates of the electron and the internuclear distance [62]. The simulation of H_2^+ focuses on the ground state and, therefore, only the PDE loss and the normalization loss are required to train the PINN. The energy for a given distance of the nuclei is now computed by an NN that takes as input the nuclei coordinates. During training, this energy NN learns a continuous representation of the eigenenergy as a function of the nuclei coordinates. At the same time, another network is devoted to the estimation of the wavefunction. This other network is necessary to compute the PDE loss, which combines the wavefunction and the energy and updates the parameters of both networks. To simplify training, prior information is added to the network architecture. For example, the output of the network uses the Linear Combination of Atomic Orbitals (LCAO) approximation as a baseline, and the network learns the difference from this baseline. Moreover, a feature transformation unit transforms the input coordinates into hydrogen atom s-orbitals and gives them to the actual MLP unit. The combination of all these transformations allows to reduce the total number of parameters required to obtain a good simulation and to speed up convergence consistently.

In the next section, an architecture inspired by [61, 62] is employed to simulate the prototypical systems of H and H_2^+ with PINNs. A study on the features of the physics-informed approach is conducted to estimate the most crucial aspects that will allow to scale up the algorithm to more complex systems.

3.2 Proposed approach and methodology

For the hydrogen atom, the many-electron Hamiltonian depicted in Eq.(1.2) simplifies to

$$\hat{H} = -\frac{1}{2}\nabla^2 - \frac{1}{|\mathbf{r}|}, \tag{3.2}$$

where $\mathbf{r} \in \mathbb{R}^3$ denotes the spatial coordinates of the electron, and the position of the proton is used as the origin of the reference frame. Instead, the Hamiltonian of the hydrogen molecular ion is

$$\hat{H} = -\frac{1}{2}\nabla^2 - \frac{1}{|\mathbf{r} - \mathbf{R}_1|} - \frac{1}{|\mathbf{r} - \mathbf{R}_2|}, \tag{3.3}$$

where $\mathbf{r} \in \mathbb{R}^3$ is once again the spatial coordinates of the electron and $\mathbf{R}_{1,2} \in \mathbb{R}^3$ are the positions of the two nuclei (i.e., the position of the two protons). Hamiltonians (3.2) and (3.3) are defined in Hartree atomic units and, therefore, all calculations performed in this thesis are consistent with this unit system. In particular, the unit of energy is the Hartree (E_h) and the unit of length is the Bohr radius (a_0). Therefore, all variables defined in atomic units are nondimensionalized, which is a standard procedure in PINNs to normalize inputs and outputs and simplify training. To perform the calculations, the entire $\mathbb{R}^3$ domain of the spatial coordinates $\mathbf{r}$ is restricted to a smaller box-shaped subset Ω in which the particle can be found almost certainly. The computational domain is chosen as large as possible to avoid introducing approximation errors, but at the same time as narrow as possible to increase the density of the collocation points sampled inside the domain. The domain used here for one-dimensional radial problems is $\Omega = [0, 10]$, and the domain used for three-dimensional problems is $\Omega = [-10, 10]^3$.

The output of interest is the wavefunction $\Psi_{\boldsymbol{\theta}}(\mathbf{r})$, which is computed to be as close as possible to the true ground-state wavefunction, $\Psi(\mathbf{r})$. The ultimate goal of the PINN model is to solve the Schrödinger equation,

$$\hat{H}\,\Psi(\mathbf{r}) = E\,\Psi(\mathbf{r}), \tag{3.4}$$

by estimating $\Psi(\mathbf{r})$ and E simultaneously. Differently from the general treatment of Section 1, the specific systems of the hydrogen atom and of the hydrogen molecular ion contain only one electron, and thus the wavefunction does not depend on the spin of the electron.

3.2.1 Architecture

The architecture proposed to solve the eigenvalue problem of Eq. (3.4) is a slight modification of the model presented in Fig. 3.1. The main difference from the architectures proposed in the literature is that our PINN model uses an external trainable variable to store the eigenvalue. This choice is justified by the remarkable results obtained by PINNs in inverse problems [63], in which the trainable variables are used to estimate unknown parameters of the PDEs. However, the discreteness of the eigenvalue spectrum introduces some additional complications with respect to traditional inverse problems. In the context of eigenvalue equations, the energy variable is updated during training according to the backpropagated gradients of the losses and using the same gradient descent rule described in Eq. (2.8). An intuitive explanation of the

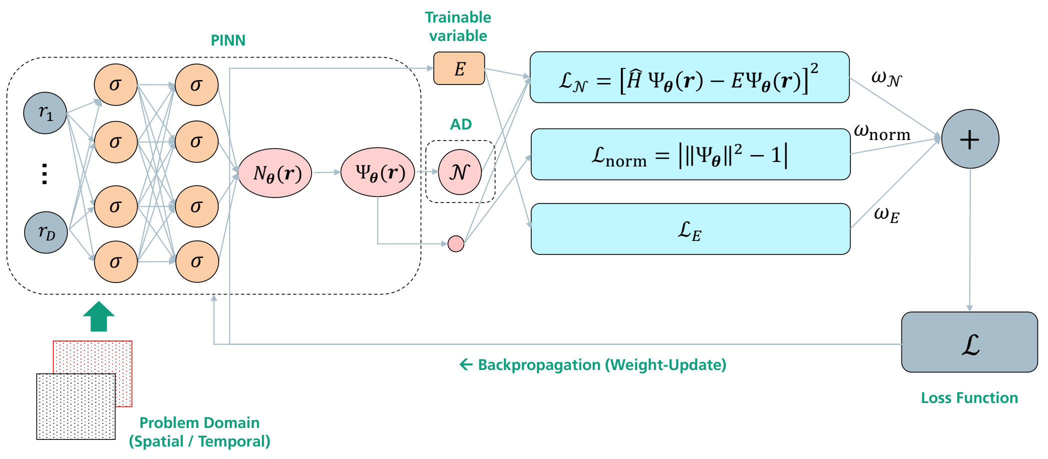

Trainable variable
PINN
E
AD
r_1
r_D
σ
$N_\theta(r)$
$\Psi_\theta(r)$
$\mathcal{N}$
$\mathcal{L}_\mathcal{N} = [\hat{H}\,\Psi_\theta(r) - E\Psi_\theta(r)]^2$
$\mathcal{L}_{\text{norm}} = |\|\Psi_\theta\|^2 - 1|$
$\mathcal{L}_E$
$\omega_\mathcal{N}$
ω_{norm}
ω_E
$+$
$\mathcal{L}$
← Backpropagation (Weight-Update)
Loss Function
Problem Domain (Spatial / Temporal)

Figure 3.2: PINN architecture for solving eigenvalue problems of H and H_2^+.

ability of the network to find the correct eigenvalue is as follows. PINNs solve direct problems by minimizing the residuals, as is clear from the PDE loss of Eq. (2.4). When the direct problem is well defined, the training procedure scans the whole parameter space in search of the best approximation of the true PDE solution. Under such conditions, the loss curve typically flattens when the approximation capabilities of the NN are saturated, and no improvements are possible without increasing the complexity of the model (i.e., without increasing the number of parameters of the network). However, if the eigenvalue equation is solved with the same strategy used for direct problems, the choice of the eigenvalue is crucial to avoid learning false solutions. False solutions are clearly visible from the loss landscape because the PDE loss quickly reaches a plateau and it starts oscillating around that value. If we now freeze the NN variable and make the eigenvalue trainable to minimize the residuals of Eq. (2.4), the variable tends to another eigenvalue and the process can be started again. This iterative approach is analogous to a self-consistent (or fixed-point) algorithm, but with the difference that the energy is not computed exactly at each step. This simplification allows to speed up the training consistently, but has the drawback of having an eigenvalue that might change too quickly.

The underlying NN architecture used in this thesis is a MLP with 3 dense hidden layers and 64 neurons in each layer, unless different values for these parameters are specified. The activation function for each intermediate neuron is the hyperbolic tangent.

Along with the explanation of the training loss components, some code snippets that cover the implementation in the DeepXDE library [57] are provided. The code is written in TensorFlow `2.10` and DeepXDE `1.9`, and it is executed by the DeepXDE library using the Tensorflow `1.x` backend and the compatibility routines provided by TensorFlow `2.10`.

PDE loss

The PDE loss is almost equivalent to the loss described in Eq. (3.1),

$$\mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta}, E) = \frac{1}{N_f} \sum_{i=1}^{N_f} \left[\hat{H}\,\Psi_{\boldsymbol{\theta}}(\mathbf{r}^i) - E\,\Psi_{\boldsymbol{\theta}}(\mathbf{r}^i)\right]^2, \tag{3.5}$$

but the dependence on the eigenvalue variable is made explicit here. The collocation points $\{\mathbf{r}^i\}_{i=1\ldots N_f}$ are sampled inside the computational domain Ω according to the Hammersley sequence, a low-discrepancy (or quasirandom) sequence [64]. Low-discrepancy sequences are typically used as a replacement of uniformly distributed random numbers in tasks such as numerical integration, because the points from a low-discrepancy sequence are more evenly distributed than pseudorandom numbers.

```
def pde(x, y):
"""
Input:
    - x: tf.Tensor, shape [num_domain, 1].
      Coordinates of the training points.
    - y: tf.Tensor, shape [num_domain, 1].
      Predictions of the network.
Output:
    - residuals: list of tf.Tensor.
      Residuals for each PDE loss (only 1 in this case).
"""
    residuals = []
    # radial Schroedinger equation (Hydrogen atom)
    du_drr = dde.gradients.hessian(y, x, i=0, j=0)

    residuals.append(
        -0.5 * du_drr
        -(1.0/tf.norm(x, axis=1, keepdims=True)) * y[:,0:1]
        - energy * y[:,0:1]
    )
    return residuals
```

Code 3.1: Function that defines the PDE loss in the radial Schrödinger equation for the H atom.

Boundary conditions

The computational domain is chosen large enough to contain the electron almost certainly. Therefore, the boundary conditions of Eq. (2.2) are translated into this model by setting to zero the wavefunction at the boundary points, i.e.,

$$\Psi_{\boldsymbol{\theta}}(\mathbf{r}) = 0, \qquad \mathbf{r} \in \partial\Omega, \tag{3.6}$$

so that the probability of finding the electron around the domain boundaries is approximately null. As already remarked, this approximation becomes increasingly better as the computational domain enlarges. Despite the possibility of imposing this boundary condition with a boundary loss, similar to the loss defined in Eq. (2.5), the output of the network can be parametrized to satisfy the Dirichlet boundary condition of Eq. (3.6) by design. The chosen parametrization is of the form

$$\Psi_{\boldsymbol{\theta}}(\mathbf{r}) = \mathbb{I}_{\mathbf{r}\notin\partial\Omega} \cdot N_{\boldsymbol{\theta}}(\mathbf{r}), \tag{3.7}$$

where $\mathbb{I}_{\mathbf{r}\notin\partial\Omega}$ is an arbitrary function that vanishes at the boundary $\partial\Omega$ and $N_{\boldsymbol{\theta}}(\mathbf{r})$ is the output of the neural network before the transformation [39]. The boundary conditions enforced in this way are called *hard boundary conditions*. This approach is generalizable to more complex boundary conditions [38], but this parametrization is enough for our model. The parametrization chosen in the present model is

$$\Psi_{\boldsymbol{\theta}}(\mathbf{r}) = \left[\prod_{i=1}^{D} \left(1 - e^{-\alpha(r_i - r_L)}\right)\left(1 - e^{-\alpha(r_R - r_i)}\right)\right] \cdot N_{\boldsymbol{\theta}}(\mathbf{r}), \tag{3.8}$$

where $r_L < r_R$ are the left and right coordinates, respectively, of the computational domain along each dimension and r_i denotes the i-th component of $\mathbf{r}$. The number of dimensions D is left implicit to fit both one- and three-dimensional problems within this definition. The coefficient α is chosen to be equal to $\alpha = 10/(r_R - r_L)$ to ensure a fast decay of the exponential far from the boundary. This particular choice of the function $\mathbb{I}_{\mathbf{r}\notin\partial\Omega}$ allows to have a null wavefunction at the boundary, while the NN output is not influenced by this parametrization in the bulk of the computational domain. The coefficient α is tunable, but it cannot be too large to avoid large gradients during backpropagation.

```python
def output_transform(x, y):
"""
Input:
    - x: tf.Tensor, shape [num_domain, 1].
      Coordinates of the training points.
    - y: tf.Tensor, shape [num_domain, 1].
      Predictions of the network before parametrization.
Output:
    - y_out: tf.Tensor, shape [num_domain, 1].
      Parametrized predictions of the network.
"""
y_out = (
    (1-tf.exp(-10.0*(x - x_min))) *
    (1-tf.exp(-10.0*(x_max - x))) *
    y[:,0:1]
)
return y_out
```

Code 3.2: Function that parametrizes the network output in the radial Schrödinger equation for the H atom.

The use of a hard boundary condition avoids the use of an additional loss function of the type of Eq. (2.5), namely a *soft boundary condition*. In fact, with the addition of any new loss the training becomes less stable due to the competition with all the other losses in the training procedure. When the number of boundary conditions enforced in a soft way grows, a consistent balancing effort is needed to allow the simultaneous optimization of all the soft boundary conditions. However, hard boundary conditions are easy to implement only for relatively simple domains (e.g., multidimensional rectangles, hyperspheres, etc.), while soft boundary conditions are easier to implement for more complex domain geometries.

In conclusion, no loss is required to satisfy the boundary conditions in this model.

Normalization loss

Unlike objective functions described in the Sect. 1.3 of VMC algorithms, the loss function in Eq. (3.5) does not impose any condition on the normalization of the wavefunction. It is well known from the linearity of the Schrödinger equation that an unnormalized wavefunction can be normalized at the end of the calculations without changing the structure of the solution. However, when simulating the Schrödinger equation with PINNs, the normalization of the wavefunction impacts the quality of the solutions consistently. In fact, the solution $\Psi_{\boldsymbol{\theta}}(\mathbf{r}) = 0$ is a valid solution of Eq. (3.4) and minimizes exactly the PDE loss in Eq. (3.5). Therefore, in the absence of any other regularization, the PINN tends to learn the null function with a very high probability.

An additional loss is added to remove the null solution from the possible solutions of the model. In particular, the normalization of the wavefunction is imposed through the normalization loss

$$\mathcal{L}_{\text{norm}}(\boldsymbol{\theta}) = \left| \|\Psi_{\boldsymbol{\theta}}\|^2 - 1 \right| , \tag{3.9}$$

where $\|\Psi_{\boldsymbol{\theta}}\|^2 = \int_{\Omega} d^D\mathbf{r} \; |\Psi_{\boldsymbol{\theta}}(\mathbf{r})|^2$. The critical part in the evaluation of this normalization loss is the estimation of the D-dimensional integral. In fact, there is not currently a standard way of implementing this integral in TensorFlow, and all the traditional integration routines do not support the backpropagation of gradients. This problem is typically handled by specific architectures, such as the Tensor Neural Network (TNN) architecture defined in [65, 66], or by Monte Carlo (MC) methods, as in VMC algorithms.

In our PINN approach, the collocation points $\{\mathbf{r}^i\}_{i=1\ldots N_f}$ are sampled inside the computational domain Ω following the Hammersley low-discrepancy sequence [64]. Quadrature schemes such as the Gaussian quadrature rule increase the computational cost of the algorithm, because additional collocation points need to be sampled to perform the integrations correctly. Also, the application of one-dimensional integration algorithms (e.g., Simpson's rules) for each dimension is not possible when the domain points are randomly distributed in space. For these reasons, in this thesis the integrals are estimated by exploiting the properties of the low-discrepancy distribution of collocation points in Ω. Integration methods that use low-discrepancy sequences are denoted *quasi-Monte Carlo methods.* The prefix "quasi" indicates that the integrals are evaluated using deterministic quasirandom sequences instead of pseudorandom number. The advantages are two. First, no new collocation points are sampled since the training points already belong to a low-discrepancy distribution. Second, quasi-MC methods have better convergence rates than traditional MC methods if N_f is large enough [67]. In practice, the integral is approximated as

$$\|\Psi_{\boldsymbol{\theta}}\|^2 = \int_\Omega d^D\mathbf{r}\ |\Psi_{\boldsymbol{\theta}}(\mathbf{r})|^2 \approx \sum_{i=1}^{N_f} \frac{V(\Omega)}{N_f} \left|\Psi_{\boldsymbol{\theta}}(\mathbf{r}^i)\right|^2 := \|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}, \tag{3.10}$$

where the symbol $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$ denotes the approximated value computed over N_f collocation points and $V(\Omega)$ is the volume of the computational domain. The normalization loss used in the simulation then takes the form

$$\mathcal{L}_{\text{norm}}(\boldsymbol{\theta}) = \left|\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f} - 1\right| = \left|V(\Omega)\sum_{i=1}^{N_f}\frac{1}{N_f}\left|\Psi_{\boldsymbol{\theta}}(\mathbf{r}^i)\right|^2 - 1\right|, \tag{3.11}$$

in which an accurate estimation of the integral is computed at each step efficiently. This additional loss effectively implements a regularization on the wavefunction that prevents the PINN to learn the null solution.

```
def normalization_loss(y):
"""
Input:
    - y: tf.Tensor, shape [num_domain, 1].
```

```
      Predictions of the network (after parametrization).
Output:
    - norm_loss: tf.Tensor, shape [1].
      Parametrized predictions of the network.
"""
    norm_loss = tf.math.abs(
        geom_volume * tf.reduce_mean(y) - 1.0
    )
    return norm_loss
```

Code 3.3: Function that defines the normalization loss.

However, this loss only enforces a soft constraint on the norm of the wavefunction, and the final wavefunction might be unnormalized nevertheless. The main task of the normalization loss is, therefore, to ensure that the norm of the wavefunction is finite and as close as possible to 1. At the end of training, unnormalized wavefunctions can be eventually normalized by computing a more precise integral with traditional integration techniques, since no backpropagation is required after the training has finished. This final normalization is a heavy computational task but is computed only once, while the estimation of the integral in the training loop is less accurate but fast. A study on the approximation performance of $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$ is carried out in Sect. 4.4.

Eigenvalue losses

The last ingredient necessary for our PINN architecture is the ability to vary the eigenvalue during training. Without a specific eigenvalue loss, the updates of the eigenvalue variable would be determined only by the PDE loss, and no specific behavior can be imposed on the eigenvalue evolution. Experiments performed with only $\mathcal{L}_{\text{PDE}}$ and $\mathcal{L}_{\text{norm}}$ have shown that the eigenvalue behaves in an unpredictable way if the function is not properly normalized (see Sect. 4.2.2 and 4.3.2).

Therefore, an additional loss is added to the model to specifically target the eigenvalue. Ideally, an eigenvalue loss for the problem of finding the ground-state energy should be identically zero where the eigenvalue is equal to the true ground-state energy. If the true ground-state energy is known exactly, then the whole problem becomes a simpler direct problem. Of course, we are interested in solving the full eigenvalue problem, in which the true ground-state eigenvalue has to be computed by the PINN in an unsupervised way. Therefore, the eigenvalue loss must try to minimize the eigenvalue as much as possible, without knowing the lowest valid eigenvalue. Two different losses have been investigated in this model:

$$\mathcal{L}_{E1}(E) = E, \tag{3.12}$$

$$\mathcal{L}_{E2}(E) = (E - E_{\text{ref}})^2, \tag{3.13}$$

which are both possible to influence the eigenvalue evolution. For example, the loss $\mathcal{L}_{E1}$ decreases the eigenvalue by a fixed amount at each iteration, since the gradient of $\mathcal{L}_{E1}$ with respect to the eigenvalue E is constant and equal to 1. The loss $\mathcal{L}_{E1}$ is not a traditional loss, as it is not null when the true eigenvalue is found, but is useful to lower the eigenvalue during training. Otherwise, the loss $\mathcal{L}_{E2}$ is more robust but it needs a reference energy E_{ref}. The only requirement is that this reference energy must be closer to the ground-state energy than to any other valid eigenvalue. Therefore, $\mathcal{L}_{E2}$ allows to restrict the eigenvalue range, and the width of such range can be tuned by the weight loss of $\mathcal{L}_{E2}$. Interestingly, this same approach is generalizable to discover wavefunctions and excited energies above the ground-state energy. Also the loss $\mathcal{L}(E) = \exp(E)$, suggested in [41], has been tested, but the evolution of the eigenvalue with this loss is less stable than with other losses.

3.2.2 Training

The generic loss described in Eq. (2.7) takes the form

$$\mathcal{L}(\boldsymbol{\theta}, E) = \omega_{\mathcal{N}}\, \mathcal{L}_{\mathcal{N}}(\boldsymbol{\theta}, E) + \omega_{\text{norm}}\, \mathcal{L}_{\text{norm}}(\boldsymbol{\theta}) + \omega_E\, \mathcal{L}_E(E), \tag{3.14}$$

and the goal of the training procedure is to minimize this loss. The initial values of the NN parameters $\boldsymbol{\theta}$ are given by the random Glorot uniform initializer [68]. The update rule for the network parameters $\boldsymbol{\theta}$ is analogous to the update rule of Eq. (2.8), and the update rule for the eigenvalue variable E is

$$E \longleftarrow E - \eta\, \frac{\partial \mathcal{L}(\boldsymbol{\theta}, E)}{\partial E}, \tag{3.15}$$

where $\eta > 0$ is the same learning rate used to update $\boldsymbol{\theta}$ and is indicated without any dependence on the training progress to simplify the notation. Actually, in the present approach, the learning rate η is decreased during training as

$$\eta(s) = \eta(0) \cdot [1 + s/\Delta s]^{-1}, \tag{3.16}$$

where $s \in [0, s_{\text{tot}} - 1]$ indicates the index of the current training iteration, s_{tot} is the total number of training iterations, $\eta(0)$ is the starting learning rate, and Δs is the decay step parameter that characterizes the velocity of the decay. In an iteration of the gradient descent algorithm, the loss (3.14) is calculated on all the training points sampled within the computational domain Ω. This single iteration is called "epoch", in analogy with the terminology used in traditional ML algorithms because the PINN learning algorithm computes the loss $\mathcal{L}(\boldsymbol{\theta}, E)$ over all the training points. All the simulations are performed using the Adam optimizer [44]. As already stated in the PDE loss defined above, the training points are sampled following the Hammersley low-discrepancy sequence [64], unless otherwise specified. Training points are resampled 10 times during the whole training procedure.

Chapter 4

Results

This section is dedicated to the analysis of the results obtained with the PINN architecture for the simulation of the H atom and of the H_2^+ molecule. The quality of the simulations is assessed by considering quantitative metrics, and the results are compared with the values present in the literature.

4.1 Metrics and datasets

If a reference solution is defined as $\{\mathbf{r}^i, \Psi^i\}_{i=1\ldots N_r}$, the predictions of the PINN evaluated at the same domain points are $\{\mathbf{r}^i, \Psi_{\boldsymbol{\theta}}(\mathbf{r}^i)\}_{i=1\ldots N_r}$. The Mean Absolute Error (MAE) of the wavefunction squared,

$$\epsilon_{L1} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\left|\Psi_{\boldsymbol{\theta}}(\mathbf{r}^i)\right|^2 - \left|\Psi^i\right|^2\right|, \tag{4.1}$$

is the metric used to evaluate the performance of the PINN models in this thesis. The advantage of using this metric is that it has the same units of the wavefunction squared, and this loss can be easily compared with the typical values of the wavefunction.

The number of samples N_r typically depends on the available datasets. The ground-state energy and the wavefunction of the H atom are known analytically, so $N_r = 100$ and $N_r \sim (50)^3$ uniformly distributed points have been chosen to evaluate the metric in the one- and three-dimensional cases, respectively. For the case of the hydrogen molecular ion, H_2^+, the predictions of our

PINN have been compared with the prediction in [62]. This choice is motivated by the excellent agreement between the energy predictions of [62] and the reference energies reported in the literature [69], for different values of the internuclear distance between the two nuclei of the H_2^+ molecule. The dataset for H_2^+ has $N_r = 150$ points sampled along the line that passes through the two nuclei.

The simulations are run on a NVIDIA Quadro RTX 5000 GPU with 16 GB of memory. The training times are referred to simulations run with this setup.

4.2 One-dimensional H atom

The first system to be investigated is the one-dimensional hydrogen atom. By exploiting the angular symmetry of the problem, the Schrödinger equation in Eq. (3.4) with Hamiltonian (3.2) can be simplified to the one-dimensional equation

$$\left[-\frac{1}{2}\frac{\partial^2}{\partial r^2}-\frac{1}{r}\right]R(r)=E\,R(r), \tag{4.2}$$

defined in terms of the radial coordinate $r=|\mathbf{r}|$ and of the auxiliary radial function $R(r)$. The function $R(r)$ depends on the total wavefunction $\Psi(r)$ as $R(r)=r\,\Psi(r)$ [1]. This auxiliary function is convenient because it avoids the singularity at $r=0$, since it is possible to demonstrate that $R(r)$ must satisfy $\lim_{r\to 0^+} R(r)=0$ to be a valid solution. Therefore, the boundary conditions are that $R(r)$ must vanish at the boundary of the domain $[0,\infty)$, which is restricted to the computational domain $\Omega=[0,10]$. A parametrization of the type of Eq. (3.8) ensures that $R(r)$ vanishes at the boundary by construction. The parametrized output of the network is indicated as $R_{\boldsymbol{\theta}}(r)$, with the subscript $\boldsymbol{\theta}$ indicating the dependence on the network parameters. The normalization condition is recast in terms of the radial function as $\int_\Omega dr\,[R_{\boldsymbol{\theta}}(r)]^2=1$. The analytical ground state solution is

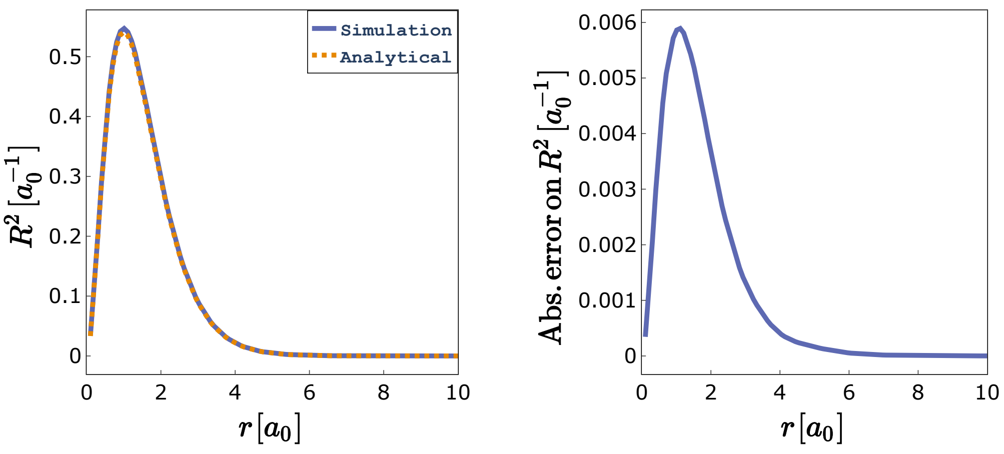
$$R(r)=2\,r\,e^{-r}, \tag{4.3}$$

which corresponds to the ground state energy $E=-0.5$. Simulations of the one-dimensional hydrogen atom use these references to assess the performance of the PINN approach.

4.2.1 No eigenvalue estimation

The Schrödinger equation for the radial component of the wavefunction is first solved by fixing the eigenvalue E to the analytical value of the ground state energy, $E=-0.5$. Therefore, the Schrödinger eigenvalue equation becomes a simpler one-dimensional equation that can be easily solved in the framework of PINNs, as already demonstrated in the literature [61]. In this setup, the loss is composed by the PDE loss $\mathcal{L}_{\mathcal{N}}$ and the normalization loss $\mathcal{L}_{\text{norm}}$.

The training of the PINN model gives the results of Fig. 4.1 and 4.2. The



(a) Radial function squared. (b) Absolute error.

Figure 4.1: Results for the one-dimensional H atom with fixed eigenvalue.

absolute errors of Fig. 4.1b show a shape similar to the estimated wavefunction, probably because the norm of the solution is different from 1. In fact, no final rescaling of the solution is performed to keep the simulation results intact, but a rescaling operation can surely increase the quality of the results. Furthermore, the training is interrupted after 50k epochs, but the loss curves of Fig. 4.3 suggest that a longer training might benefit the accuracy of the trained model.

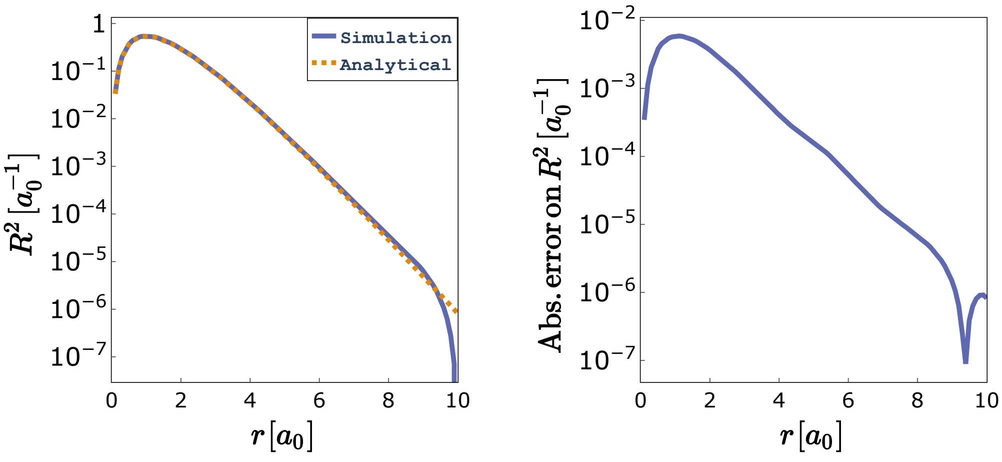
The final metrics are shown in the first row of Tab. 4.2.

4.2.2 With eigenvalue estimation

Hyperparameter	Values
Initial energy	[-1, -0.75, -0.5, -0.25]
Initialization period	[0, 10k, 20k]

Table 4.1: Hyperparameters used to assess the importance of the initialization energy for the one-dimensional H atom with eigenvalue estimation.

In this model, the eigenvalue E is a trainable variable, and its value evolves during training. This model is substantially different from the one with eigenvalue fixed to $E = -0.5$, as the estimation of the eigenvalue adds an additional



(a) Radial function squared (log scale). (b) Absolute error (log scale).

Figure 4.2: Results for the one-dimensional H atom with fixed eigenvalue (log scale).

layer of complexity to the usual PINN training. In fact, the updates of the wavefunction parameters $\boldsymbol{\theta}$ and the updates of E are deeply correlated. The additional degree of freedom given by E allows the PINN to converge more easily to a solution, but there is no guarantee that the final solution corresponds to the lowest eigenvalue. In this context, the choice of the initialization value of the energy variable is fundamental in determining the energy to which the training converges. In the simulations of Fig. 4.4-4.6, the model converges to the ground state solution if $E = -1.0$ is chosen as the initialization value of the energy trainable variable.

A typical training of this model proceeds as follows. If the initialization value of E is below the ground state energy, the first part of training shows an increase in the value of E. During this first phase, the norm of the wavefunction is close to zero, as is visible in Fig. 4.5, where the normalization loss is flat for the first $\sim$ 20k epochs. At the same time, the value of E slowly increases. Until a valid eigenvalue is found, the PINN minimizes the total loss by giving as output a wavefunction which norm is close to zero. Figure 4.6 shows the evolution of the trainable eigenvalue variable during training. The same behavior has been observed in many experiments performed with this one-dimensional model when E is fixed below the true ground state. When

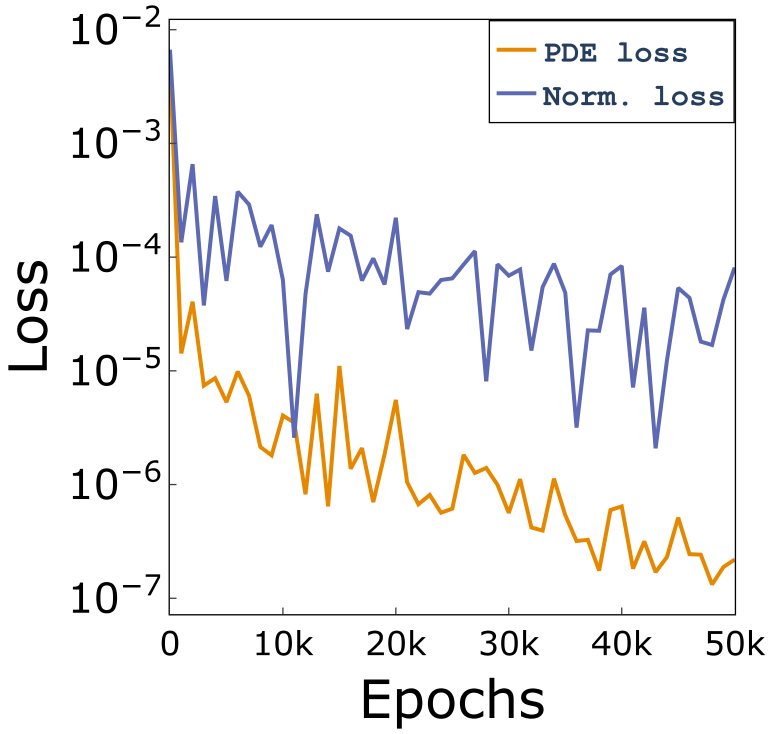
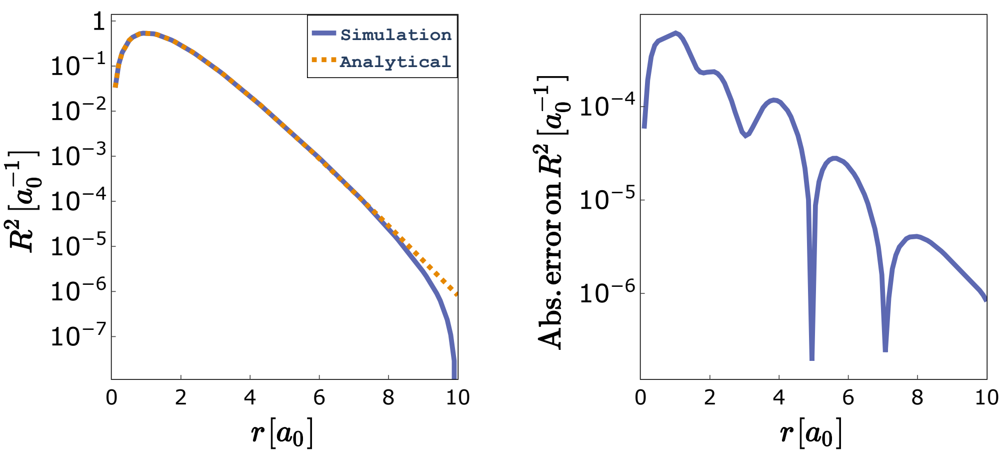


Figure 4.3: Evolution of PDE and normalization losses along epochs for the one-dimensional H atom with fixed eigenvalue.

E approaches a valid eigenvalue, the model starts to learn the true ground state wavefunction with the correct normalization. In Fig. 4.4a and 4.4b, the squared wavefunction and the absolute errors with respect to the true solution show that the results are comparable with the results of Sect. 4.2.1. The estimation of the eigenvalue in the second row of Tab. 4.2 is computed as the mean and the standard deviation of the E value in the last 20% training epochs.

Figure 4.7 shows the impact that a value of the initial energy has on the simulation. Different experiments have been performed by fixing the energy variable to the initialization value for 0, 10k, or 20k iterations. After this initialization period, the training is performed as usual, the only difference being that the wavefunction starts from a different initialization than the random one when the number of initialization epochs is greater than 0. The ϵ_{L1} metrics in Fig. 4.7a demonstrate that the initialization period does not impact the per-



(a) Radial function squared (log scale). (b) Absolute error (log scale).

Figure 4.4: Results for the one-dimensional H atom with eigenvalue estimation (log scale).

formance of the training procedure in one-dimensional problems. Instead, the initial energy is of fundamental importance in determining the results of the simulation. In fact, Fig. 4.7b clearly shows that for initial energies $E \leq -0.5$, the final energy is close to the true ground state energy $E = -0.5$. For initial energies above this value, the eigenvalue tends to a value of E close to -0.125, which is the energy of the first excited state of the hydrogen atom. Consequently, also the ϵ_{L1} metrics corresponding to the initial energy $E = -0.25$ are two orders of magnitude higher than those from the other simulations, since a solution associated with a different eigenvalue is discovered.

In synthesis, the current PINN architecture is able to simulate the one-dimensional eigenvalue problem without the need to calculate the energy of the wavefunction at each iteration. The simplicity of the one-dimensional problem allows us to obtain good simulations without enforcing any of the eigenvalue losses presented in Sect. 3.2.1. Any additional loss would further stabilize the simulations and give more control over the dynamic of the eigenvalue [61]. The one-dimensional eigenvalue simulations are highly dependent on the choice of the initialization energy, which ultimately determines the eigenvalue of the final solution. The selection of the correct eigenvalue might be enforced by adding a $\mathcal{L}_{e2}(E) = (E - E_{\text{ref}})^2$ loss, which penalizes energies that are far away

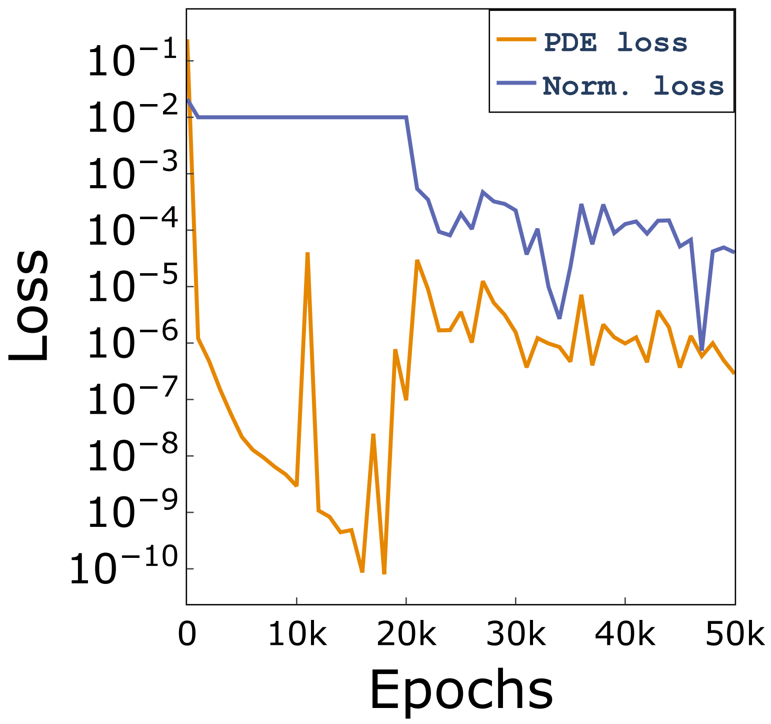


Figure 4.5: Evolution of PDE and normalization losses along epochs for the one-dimensional H atom with eigenvalue estimation.

from the reference energy E_{ref}. This approach works if E_{ref} is closer to the true energy than to any other energy in the spectrum, therefore, only partial knowledge of the true energy is needed to enforce such a constraint.

	ϵ_{L1}	E	$\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$	Training time
Fig. 4.1-4.2	$1.2 \cdot 10^{-3}$	-0.5 (fixed)	1.012	30 s
Fig. 4.4	$1.1 \cdot 10^{-4}$	-0.49998 ± 0.00006	0.999	90 s

Table 4.2: Simulation metrics of the one-dimensional H atom.

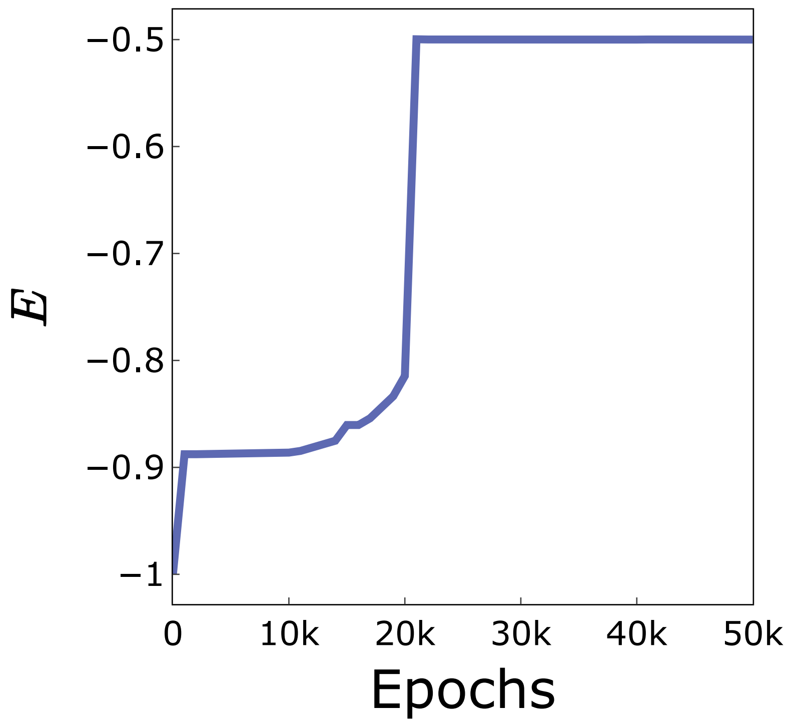
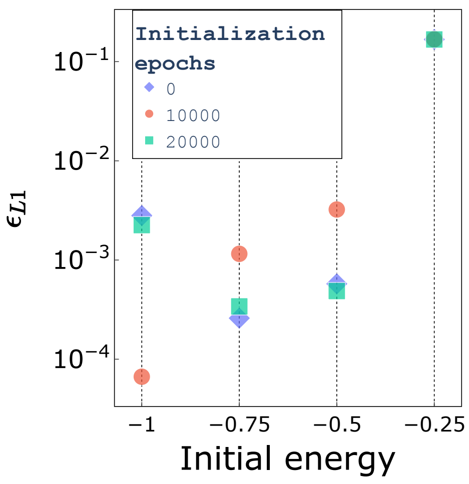


Figure 4.6: Evolution of eigenvalue along epochs for the one-dimensional H atom.

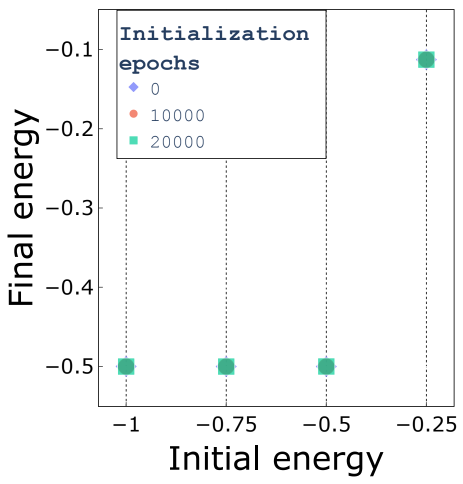
4.3 Three-dimensional H atom

The next model under investigation is the three-dimensional hydrogen atom. The Schrödinger equation in Eq. (3.4) is not simplified to a one-dimensional equation as in the previous section, but the full three-dimensional problem is considered to learn how a larger number of dimensions can impact the training of the model. The Schrödinger equation is, therefore, identical to Eq. (3.4) with Hamiltonian (3.2). The computational domain is $\Omega = [-10, 10]^3$ and the wavefunction $\Psi_{\boldsymbol{\theta}}(\mathbf{r})$ vanishes at the boundary. The normalization condition is $\int_\Omega d^3\mathbf{r}\ |\Psi_{\boldsymbol{\theta}}(\mathbf{r})|^2 = 1$ and the analytical ground state solution is

$$\Psi(\mathbf{r}) = \frac{1}{\sqrt{\pi}} e^{-|\mathbf{r}|}, \tag{4.4}$$



(a) Metric ϵ_{L1} as a function of the initialization energy for different initialization periods.



(b) Correlation between initialization energy and final energy for different initialization periods.

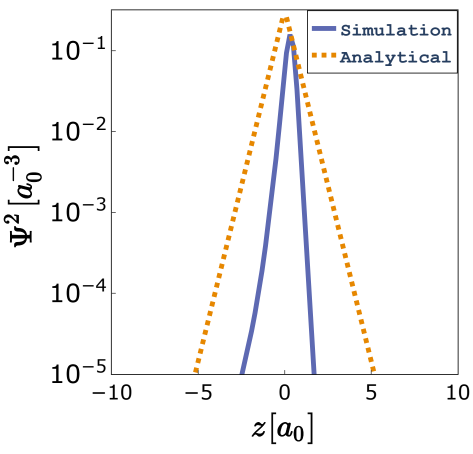
Figure 4.7: Importance of the initialization energy for the one-dimensional H atom with eigenvalue estimation.

which corresponds to the ground state energy $E = -0.5$.

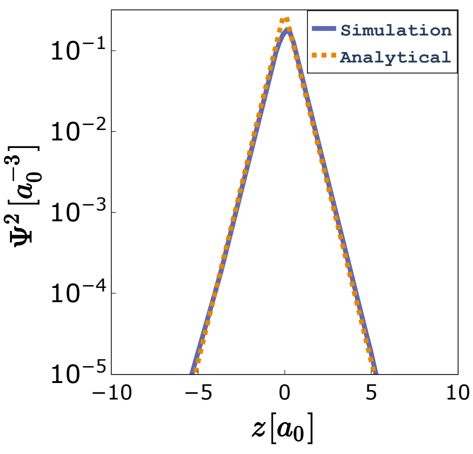
4.3.1 No eigenvalue estimation

The three-dimensional simulations of the hydrogen atom are more complex than the one-dimensional simulations of the same system, as a larger domain Ω is required and more domain points are needed to cover the entire domain. As shown in Fig. 4.9a, the precision of the three-dimensional simulation decreases around the origin of the system, where the nucleus of the hydrogen atom is located. Around this particular point, $|\mathbf{r}| = 0$, the potential energy of the electron becomes infinity and the wavefunction is not differentiable. This is a common problem in quantum chemistry, and is aggravated in many-electrons systems. In the simple case of one-electron systems, a slightly modified potential can help to avoid divergences at the origin. In these simulations, the potential within a cutoff distance $r_{\min} = 10^{-7}$ from the origin is fixed to the value of the potential at $|\mathbf{r}| = r_{\min}$ to avoid divergences.

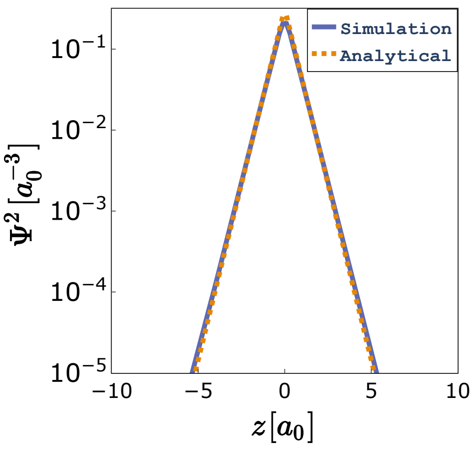
Different simulations have been performed to assess whether simulation errors at the origin are an effect of this modified potential or an intrinsic effect



(a) $N_f = 3$k, 5 dense layers.



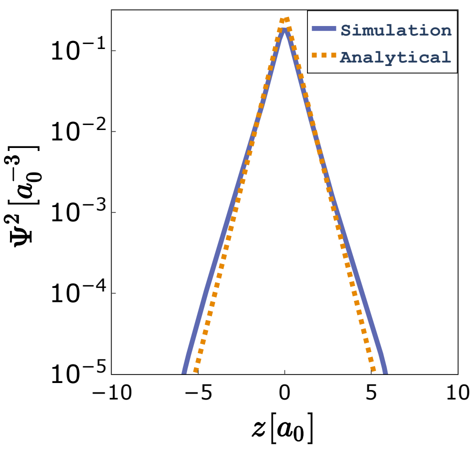
(b) $N_f = 30$k, 5 dense layers.



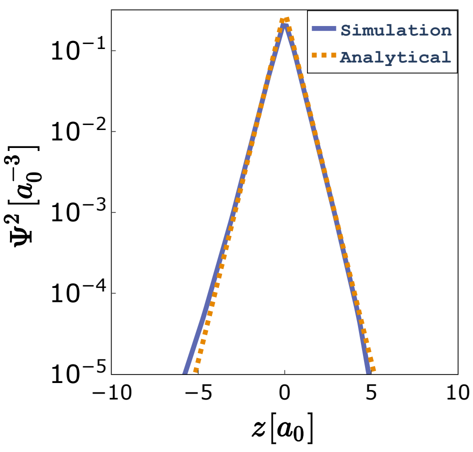
(c) $N_f = 300$k, 5 dense layers.

Figure 4.8: Results for the three-dimensional H atom with fixed eigenvalue for different numbers of domain points N_f. The other coordinates are $x = 0, y = 0$.

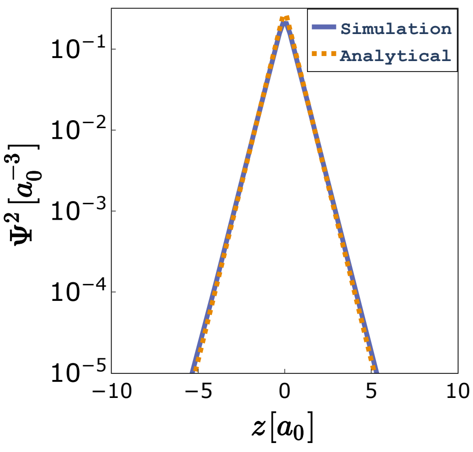
of the PINN approximations. Simulations presented in Fig. 4.8-4.9 explore the effect of increasing the domain points or the number of NN dense layers on the ability to correctly simulate the wavefunction at the origin. The results confirm the expected behavior: more domain points and larger networks improve the simulation results at the origin, but the training time consequently increases (see Fig. 4.11 and the training times in Tab. 4.5). Figures 4.10a and 4.10b show a collection of ϵ_{L1} metrics from different simulations. The hyperparameters varied during these simulations are shown in Tab. 4.3. Surprisingly, simulations performed with larger networks (i.e., with more dense layers and/or more neurons per layer) are not always associated with better results, but we expect that the repetition of those experiments would trace out stochastic fluctuations. However, the trends observed in the data allow us to prove that large values of N_f always increase the quality of the simulations. In Fig. 4.10a, models with more neurons in each dense layer perform better when the training data are sufficient (i.e., for $N_f > 30$k), while the PINN model undergoes overfitting when training data are scarce. At the same time, models with less neurons in each dense layer provide already good results for $N_f \sim 3$k compared to more complex models, but their quality improves only slightly for larger N_f. A similar behavior can be observed in Fig. 4.10b, in which the complexity of the model is tuned by increasing the depth of the network, i.e., the number of dense layers. Similarly, the complex model underperforms the two simpler models when $N_f < 10$k, while the ϵ_{L1} error of the complex model decreases monotonically for $N_f \geq 10$k. The behavior of the ϵ_{L1} metrics of the models with 3 and 4 dense layers is, instead, less regular for $N_f \geq 10$k compared to the ϵ_{L1} metrics of the model with 5 dense layers. The first part of Table 4.5 shows that the best model is the one of Fig. 4.9c and 4.8c, but is also the model that requires the longest training. Overall, the PINN model has demonstrated its ability to correctly approximate the cusp in the wavefunction at $|\mathbf{r}| = 0$, provided that the network is large enough and that enough domain points are sampled. Of course, increasing these two hyperparameters requires more computational resources for training, and therefore the choice of hyperparameters must depend on the desired level of accuracy and of the available resources. In more complex systems, an alternative to avoid the rapid increase of required resources is to use approximations such as LCAO as a baseline. Nevertheless, this study is fundamental to understand the typical number of domain points and network sizes required to perform a simulation in three-dimensional domains.



(a) $N_f = 300\text{k}$, 3 dense layers.



(b) $N_f = 300\text{k}$, 4 dense layers.



(c) $N_f = 300\text{k}$, 5 dense layers.

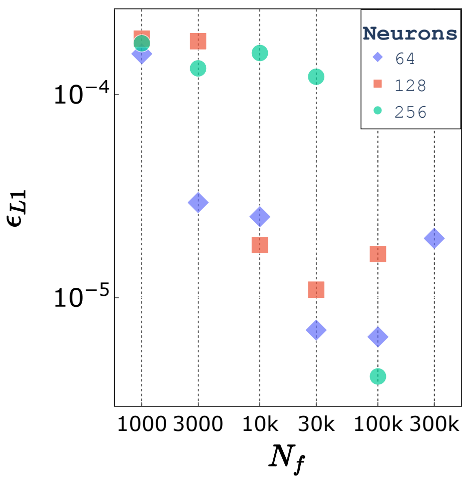
Figure 4.9: Results for the three-dimensional H atom with fixed eigenvalue for different numbers of dense layers. The other coordinates are $x = 0, y = 0$.

4.3.2 With eigenvalue estimation

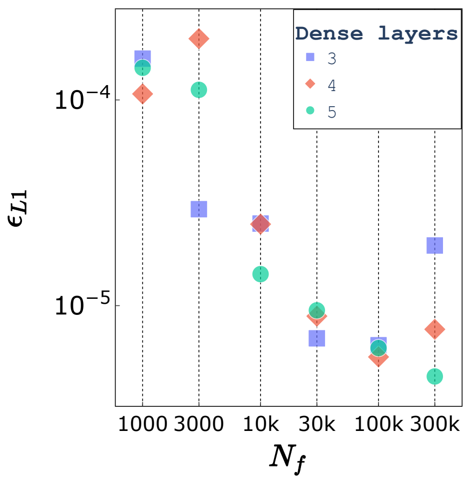
The same strategy used to estimate the radial solution of the hydrogen atom in Sect. 4.2.2 is applied at first to the H atom simulations in three dimensions. An example of these results are presented in Fig. 4.12a and 4.13a. The

Hyperparameter	Values
N_f	[1k, 3k, 10k, 30k, 100k, 300k]
Dense layers	[3, 4, 5]
Neurons per layer	[64, 128, 256]

Table 4.3: Hyperparameters used to assess the importance of N_f and of the network size for the three-dimensional atom with fixed eigenvalue.



(a) Metric ϵ_{L1} as a function of domain points and neurons in each layer, for 3 dense layers.



(b) Metric ϵ_{L1} as a function of domain points and dense layers, for 64 neurons in each layer.

Figure 4.10: Importance of N_f and network size for the three-dimensional H atom with fixed eigenvalue.

PINN clearly learns a solution that is not the ground state solution, even if the initial value of the energy variable is well below $E = -0.5$. In the analogous one-dimensional simulations, the convergence value of E completely depends on the initial value of the energy variable. Therefore, while in the one-dimensional simulations the problem of convergence to an higher eigenvalue is solved by initializing the energy variable to a value below $E = -0.5$, in the three-dimensional simulations the simple initialization of the energy variable is not enforcing any condition on the eigenvalue of convergence. Figure 4.14a clearly shows that the eigenvalue converges in a fraction of the total iterations to an high eigenvalue close to 0. Many experiments run with different learn-

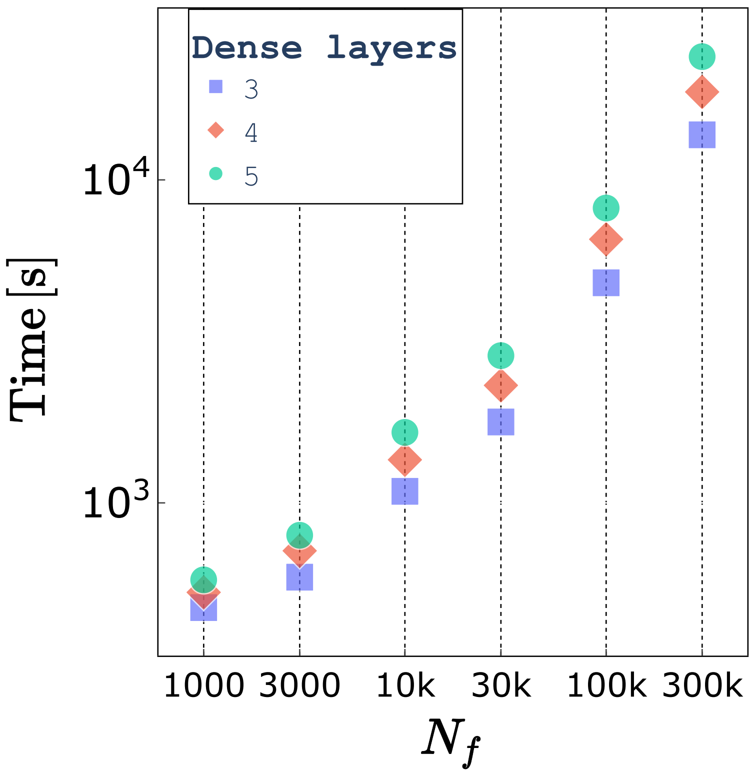


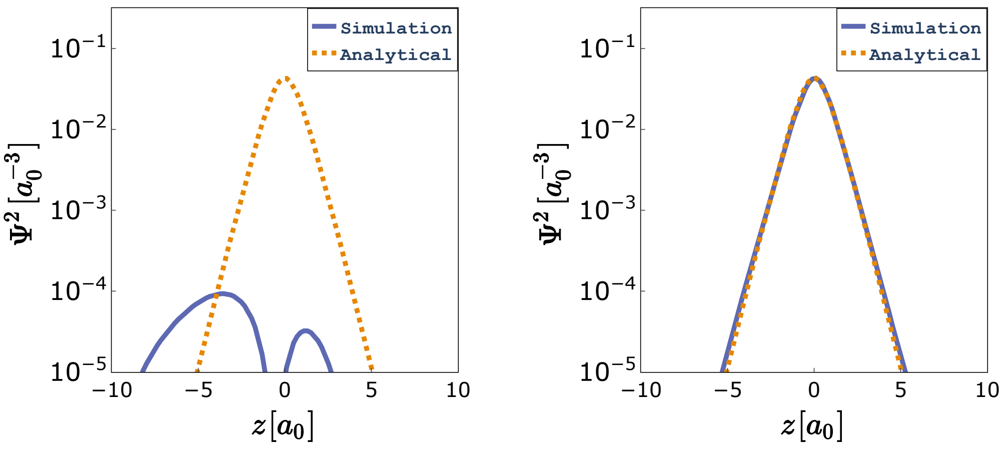
Figure 4.11: Training time as a function of domain points N_f and network size, for 64 neurons in each layer and for the three-dimensional H atom with fixed eigenvalue.

ing parameters and/or at different initialization energies always show a similar eigenvalue dynamics.

We propose a strategy to improve the control on the eigenvalue dynamics, and we use the results that demonstrate the validity of this strategy to have precious insights on the training process. The strategy is to initialize the wavefunction at a low initial E for some number of epochs and then repeat the usual training with the initialized wavefunction. The idea is that the random initialization of the network parameters $\boldsymbol{\theta}$ causes the initial random wavefunction $\Psi_{\boldsymbol{\theta}_i}$ to randomly oscillate around the null value. The true energy associated to such a random wavefunction is likely to be very close to $E = 0$, therefore, in the early training steps the initialization value of E quickly tends to this high energy thanks to the gradients derived from the PDE loss of Eq.(3.5).

Hyperparameter	Values
Initial energy	[-1, -0.75, -0.5]
Initialization period	[20k, 50k, 100k]
$\eta(0)$	[0.0003, 0.001]
Δs	[1k, 3k]

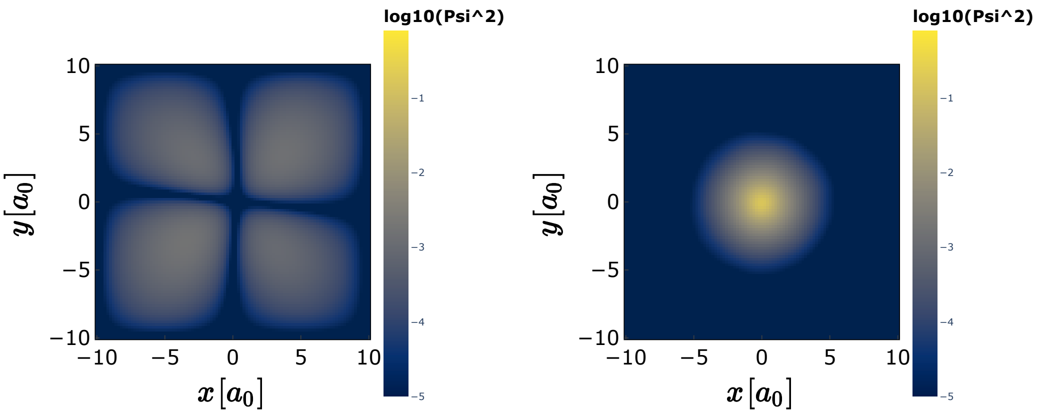
Table 4.4: Hyperparameters used to assess the importance of the wavefunction initialization for the three-dimensional H atom with eigenvalue estimation.



(a) Without initialization. (b) With 20k initialization epochs.

Figure 4.12: Results for the three-dimensional H atom with eigenvalue estimation for different numbers of initialization epochs. The other coordinates are $x = -1, y = 0$.

At the same time, a low initial value of E modifies the shape of the wavefunction towards a more compact wavefunction centered at the origin. After a transient phase, these two effects compensates and the eigenvalue reaches the convergence value. In the one-dimensional system, this effect is not visible because the training is simple and the wavefunction converges quickly, while in the three-dimensional system the convergence of the wavefunction is more slow. Therefore, in three dimensions the initial random wavefunction causes E to converge to an high value. A preliminary initialization of the wavefunction can instead drive the wavefunction to be more compact around the origin, especially if the initialization value of E is well below the true ground state energy.

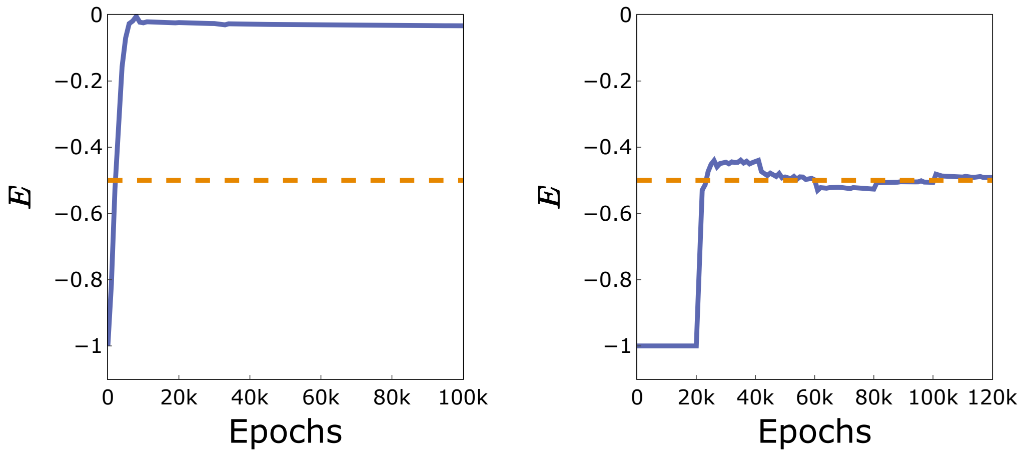


(a) Without initialization. (b) With 20k initialization epochs.

Figure 4.13: Wavefunction squared at $z = 0$ for the three-dimensional H atom with eigenvalue estimation for different numbers of initialization epochs.

After this initialization phase, the eigenvalue variable is released and is more likely to converge to the true ground state energy $E = -0.5$. Figure 4.12b and 4.13b show the results after a preliminary initialization of 20k epochs at the value $E = -1.0$, and Fig. 4.14b shows that the eigenvalue oscillates around $E = -0.5$ in the last part of training, and the eigenvalue variable is likely to converge to the true value after a longer training. The second part of Table 4.5 clearly shows that the preliminary initialization of the wavefunction is able to simplify the convergence of the simulation towards the correct ground-state energy.

Different simulations have been performed to assess the effect of the simple initialization of the wavefunction in the simulation results. Figure 4.15a shows the effect of different initialization energies in the determination of the final energy, with and without initialization of the wavefunction. Hyperparameters used to perform these simulations are presented in Tab. 4.4. Interestingly, while all uninitialized simulations converge to a final value of the energy close to ~ -0.1, the initialization process allows the convergence to the true value for some values of the hyperparameters. By properly tuning these hyperparameters, it is possible to make the convergence to the true ground state more robust, but our focus here is only on the demonstration of the beneficial effects



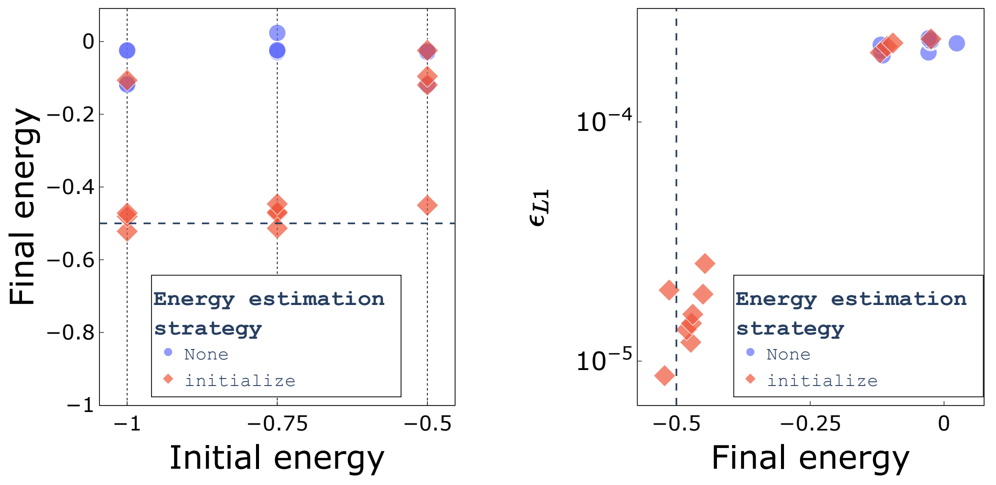
(a) Without initialization. (b) With 20k initialization epochs.

Figure 4.14: Evolution of eigenvalue along epochs for the three-dimensional H atom with eigenvalue estimation for different numbers of initialization epochs.

of the wavefunction initialization. Figure 4.16 shows the improvements of the metric ϵ_{L1} thanks to a preliminary initialization of the wavefunction. The difference of ϵ_{L1} between the uninitialized and the initialized cases is interpreted so that positive differences are associated with an improvement of the initialized case over the uninitialized case. Figure 4.16 shows that, for the same hyperparameters, the initialization of the wavefunction improves the quality of the final solutions.

	ϵ_{L1}	E	$\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$	Training time
Fig. 4.8a	$1.1 \cdot 10^{-4}$		0.180	10 min
Fig. 4.8b	$9.5 \cdot 10^{-6}$		0.963	40 min
Fig. 4.9a	$2.0 \cdot 10^{-5}$	-0.5 (fixed)	0.970	230 min
Fig. 4.9b	$7.7 \cdot 10^{-6}$		0.966	310 min
Fig. 4.8c, 4.9c	$4.5 \cdot 10^{-6}$		0.966	400 min
Fig. 4.12a	$2.2 \cdot 10^{-4}$	-0.0328 ± 0.0003	0.969	25 min
Fig. 4.12b	$1.1 \cdot 10^{-5}$	-0.492 ± 0.007	0.963	15 min

Table 4.5: Simulation metrics of the three-dimensional H atom.



(a) Correlation between initial energy and final energy. Initialized for 100k epochs.

(b) Correlation between final energy and ϵ_{L1} metric. Initialized for 100k epochs.

Figure 4.15: Effect of wavefunction initialization for the three-dimensional H atom with eigenvalue estimation. Different points with the same symbol are associated with different $\eta(0)$ and Δs (both plots) and different initial energies (only right plot), see Tab. 4.4.

4.4 Importance of training distributions

Hyperparameter	Values
N_f	[1k, 2k, 3k, 5k, 10k, 20k, 30k, 50k, 100k]
Training distribution	[Hammersley, uniform, pseudorandom]

Table 4.6: Hyperparameters used to assess the importance of the training point distribution in the estimation of $\|\Psi_{\boldsymbol{\theta}}\|^2$ for the three-dimensional H atom with fixed eigenvalue.

An evaluation of the norm estimation and of the quality of the final solutions is performed for different training distributions. The use-case under evaluation is the three-dimensional hydrogen atom, and the training focuses on the solution with known eigenvalue, i.e., on the same model of Sect. 4.3.1. Compatibly with the definition in Sect. 3.2.1, the squared norm of the wavefunction computed at each training iteration is denoted as $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$. As already described, the normalization loss $\mathcal{L}_{\text{norm}}$ depends on the estimation of $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$ in each training iteration. Since the estimation of the norm has to be repeated

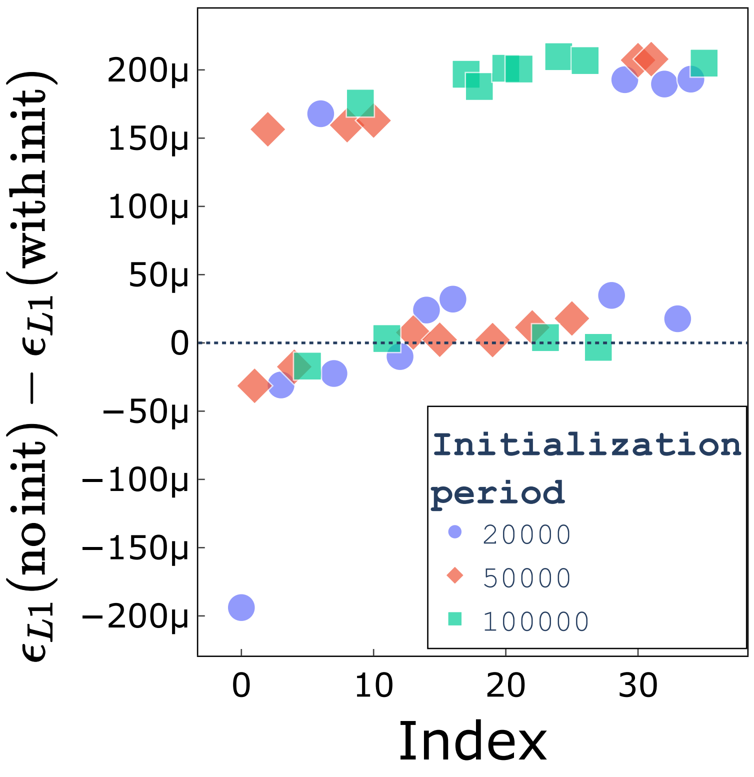


Figure 4.16: Difference of metrics ϵ_{L1} with and without initialization for the three-dimensional H atom with eigenvalue estimation. For the experiments without initialization, the "initialization period" variable has no effect.

many times, using a low number of domain points N_f is more convenient to decrease the training time, but at the same time it decreases the quality of the approximation $\|\Psi_{\boldsymbol{\theta}}\|^2 \approx \|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$. A large N_f is better in terms of the quality of the norm approximation, but significantly increases the training time. A trade-off must be found between these two extremes.

In these experiments, at the end of the training, the squared norm of the final wavefunction is evaluated on a larger number of domain points N_f', with $N_f' \gg N_f$. For the present simulations, the norm squared $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f'}$ is considered as the "true" squared norm of the wavefunction and is used as a reference to assess the quality of $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$. The reference $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f'}$ is computed on a regular grid of $N_f' = 100^3 = 10^6$ domain points using the same approximation of Eq. (3.10). This simplistic approach is motivated by its

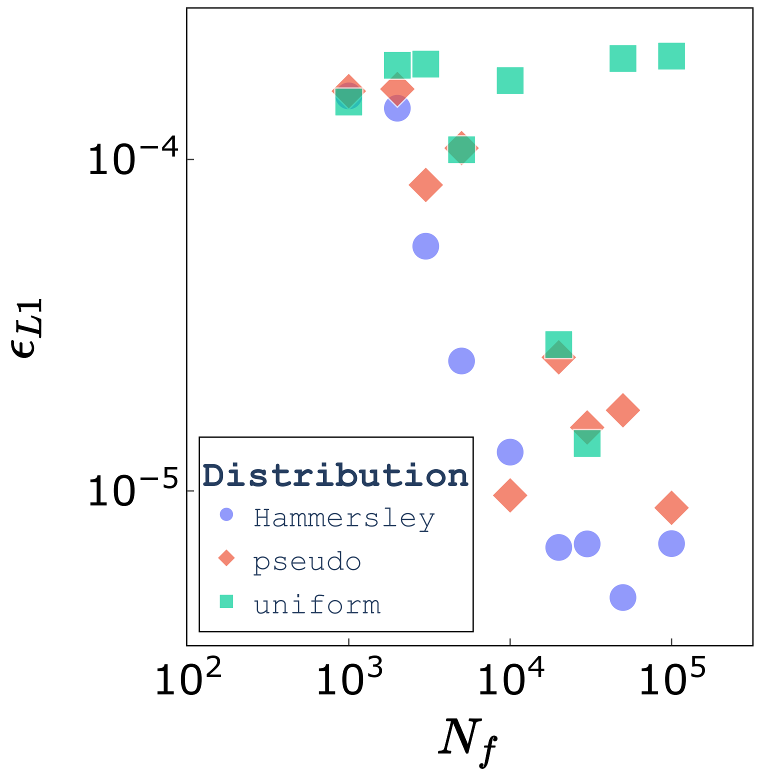


Figure 4.17: Metric ϵ_{L1} as a function of domain points N_f for different distributions of training points for the three-dimensional H atom with fixed eigenvalue.

good performance relative to other more sophisticated integration routines. For the simulations of this thesis, the reference $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ has demonstrated a similar performance to SciPy integration routines, with relative differences $\leq 0.1\%$, while taking only a fraction of the evaluation time. At the same time, the typical relative differences between $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f}$ and $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ are of the order of $\sim 3\%$, and thus $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ is considered a good estimation of the true squared norm $\|\Psi_{\boldsymbol{\theta}}\|^2$ of the wavefunction.

Figures 4.17-4.19 show the relations between the number of domain points N_f, the squared norm $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ of the final wavefunction evaluated on $N'_f = 10^6$ domain points, and the ϵ_{L1} metric with respect to the analytical solution. The relative error on the norm estimation, presented in Fig. 4.19b in absolute value, is defined as $(\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f} - \|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f})/\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$. The hyperparameters used in these

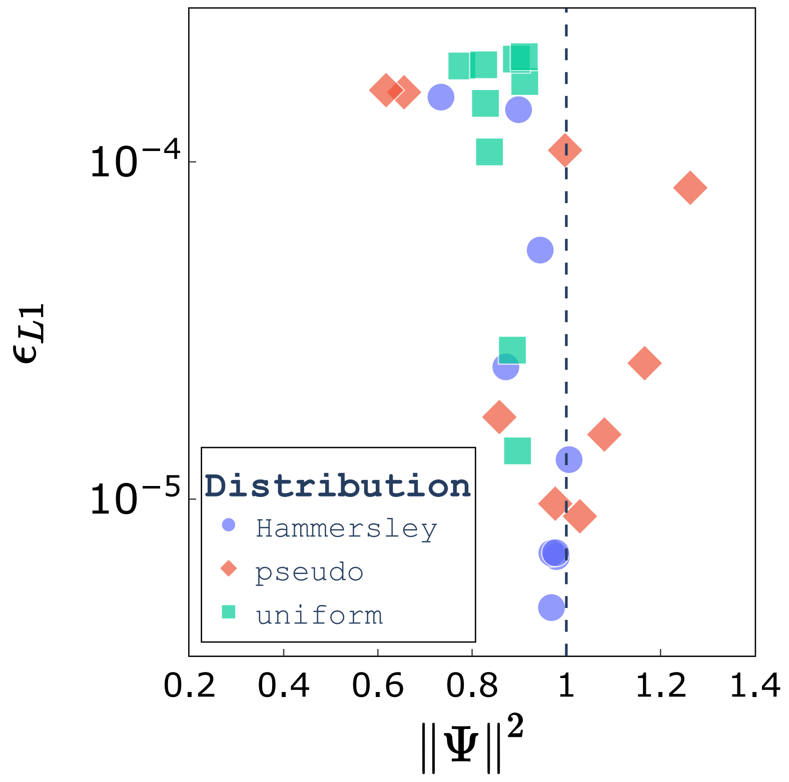
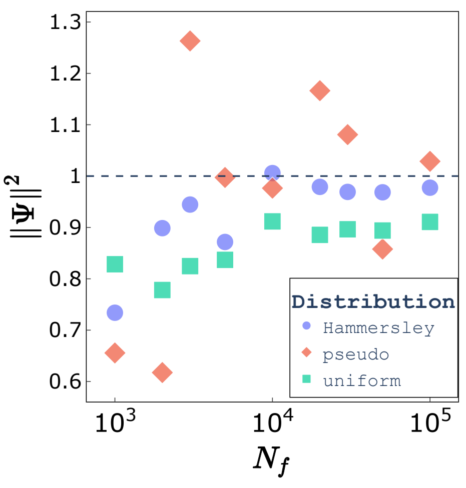
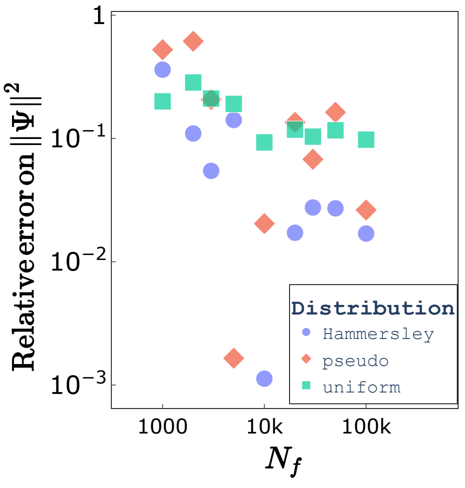


Figure 4.18: Correlation between the metric ϵ_{L1} and the norm squared $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ for different distributions of training points for the three-dimensional H atom with fixed eigenvalue. Different points with the same symbol are associated with different N_f, see Tab. 4.6.

simulations are described in Tab. 4.6. The “Hammersley” distribution samples the training points in Ω according to the low-discrepancy Hammersley sequence [64], the “uniform” distribution samples the points in a regular grid, and the “pseudo” distribution samples the training points randomly. Figure 4.17 shows that the best simulations, evaluated in terms of the metric ϵ_{L1}, are obtained with the Hammersley distribution. The metrics of the simulations performed with Hammersley decrease monotonically when the number of domain points increases, but this trend stops for $N_f \geq 20$k domain points. A similar pattern is also observed for the pseudorandom distribution, but with higher metrics with respect to the Hammersley simulations with the same hyperparameters. Instead, no clear pattern is observable for the simulations with “uniform” do-



(a) Norm squared $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$.



(b) Relative error of $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f} \approx \|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$.

Figure 4.19: Quality of the norm estimation as a function of N_f and training point distribution for the three-dimensional H atom with fixed eigenvalue.

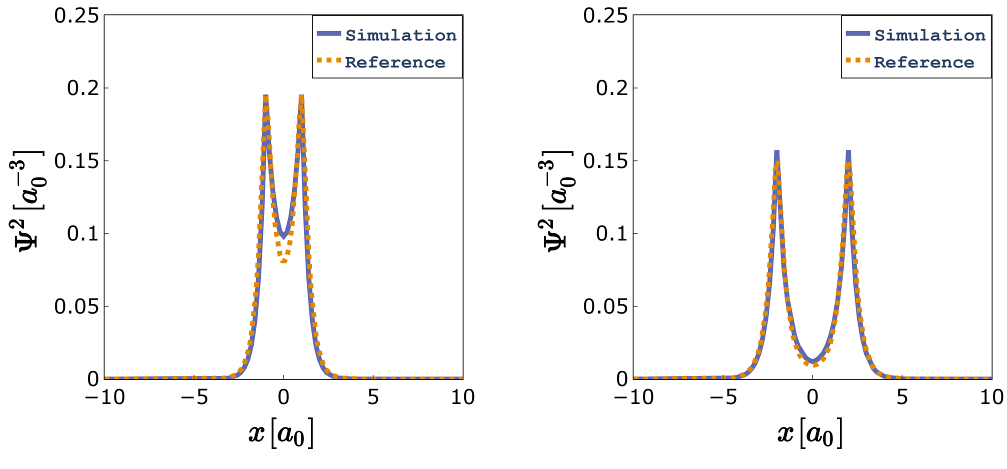
main points distribution. A possible explanation is that, for large values of N_f, the benefits of increasing the domain points are saturated and other factors limit the further improvement of the model. Figure 4.18 demonstrates that the simulations with the best metric are also the simulations with $\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ closer to 1. The simulations performed with the Hammersley distribution have, on average, a squared norm that is closer to 1 than the simulations performed with other distributions (see Fig. 4.19a). The relative error (in absolute value) of the approximation $\|\Psi_{\boldsymbol{\theta}}\|^2_{N_f} \approx \|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$ is instead depicted in Fig. 4.19b for different distributions. This picture allows us to understand how the precision on the estimation of the true squared norm $\|\Psi_{\boldsymbol{\theta}}\|^2$ (evaluated at the end of training over N'_f samples, as explained above) scales with the number of domain points N_f. While the uniform distribution simulations always commit an error $\geq 10\%$ on the estimation of the norm, the pseudorandom and Hammersley distributions reach precisions of a few percents for large values of N_f. The non-monotonic behavior over the number of domain points N_f that characterizes Fig. 4.19b might be corrected by taking averages over different repetitions of the experiments.

In synthesis, Fig. 4.17-4.19 support the choice of the Hammersley distribution to sample training points.

4.5 Three-dimensional H_2^+ molecule

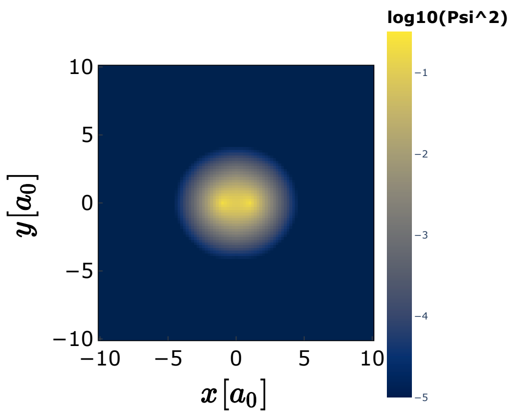
Finally, the knowledge acquired on the H atom models are applied to the three-dimensional hydrogen molecular ion H_2^+. The Hamiltonian (3.3) describes the physical system. We fix the positions of the nuclei to $\mathbf{R}_{1,2} = (\pm R, 0, 0)$, where R is defined as half the internuclear distance. The computational domain is $\Omega = [-10, 10]^3$ and the wavefunction $\Psi_{\boldsymbol{\theta}}(\mathbf{r})$ vanishes at the boundary. The normalization condition is $\int_\Omega d^3\mathbf{r}\ |\Psi_{\boldsymbol{\theta}}(\mathbf{r})|^2 = 1$. No analytical solutions of this system exist, so we use the simulations in the quantum chemistry literature as a reference. In particular, the table in [69] is used as a reference for the energy, and the wavefunctions of [62], evaluated along the internuclear line, are used as a reference solution. To simplify convergence, the LCAO approximation is used to give a baseline for the wavefunction and the network is trained to learn the corrections to this approximation. Given the nonexistence of analytical solutions of this system, only the study of the eigenvalue problem is performed.

4.5.1 Eigenvalue estimation

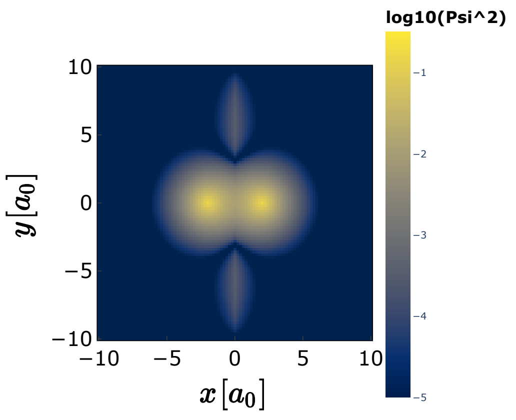


(a) $R = 1$. (b) $R = 2$.

Figure 4.20: Results for the three-dimensional H_2^+ molecule with eigenvalue estimation for different internuclear distances. The other coordinates are $y = 0, z = 0$.



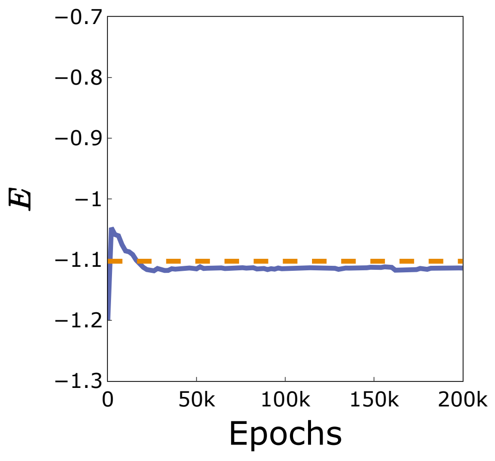
(a) $R = 1$.



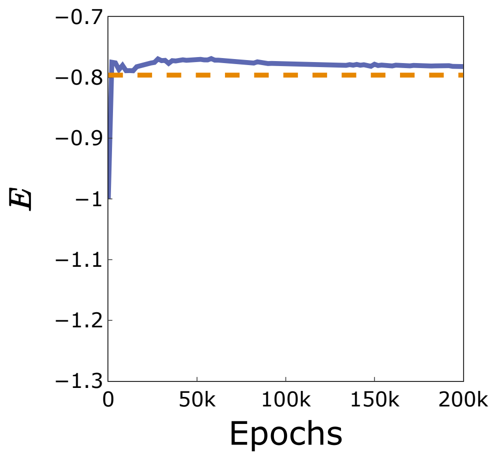
(b) $R = 2$.

Figure 4.21: Wavefunction squared at $z = 0$ for the three-dimensional H_2^+ molecule with eigenvalue estimation for different internuclear distances.

After the demonstration in Sect. 4.3.2 that a preliminary initialization of the wavefunction is beneficial in three-dimensional systems, this section is devoted to the solution of the eigenvalue equation with the eigenvalue controlled by the loss $\mathcal{L}_{e2}(E) = (E - E_{\text{ref}})^2$, defined in Eq. (3.13). The main difference of this strategy with respect to the initialization strategy is that the use of a quadratic loss on the eigenvalue requires the knowledge of a reference energy E_{ref} to which the eigenvalue E is kept close. In the hydrogen atom case, the reference energy is also the true analytical energy, but in more complex systems, such as H_2^+, approximate energies are valid candidates for E_{ref}. In fact, the true ground state energies differ from the energies predicted by the Hartree-Fock or LCAO approximations only by a small fraction of the total energy. Using these approximations as a reference value and properly tuning the loss weight ω_E corresponding to $\mathcal{L}_{e2}$, the width of the explored eigenvalue range can be tuned. A large value of ω_E tends to restrict the explored range of eigenvalues considerably, and thus the true ground state energy might be penalized if the range is too narrow. Instead, a small value of ω_E tends to enlarge the explored range of eigenvalues, and more than one valid eigenvalues can be found in such a large range. Therefore, the choice of ω_E is a crucial hyperparameter. Other losses similar to $\mathcal{L}_{e2}$ can be designed to satisfy particular

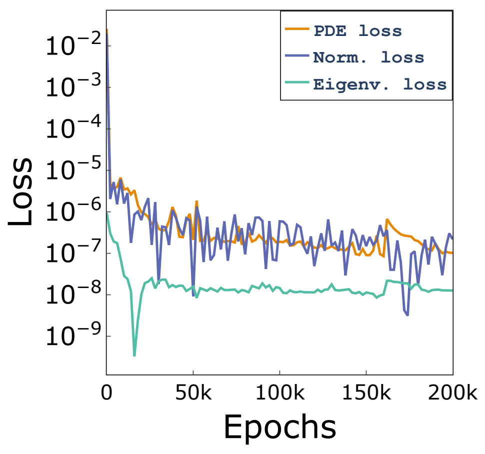


(a) $R = 1$.

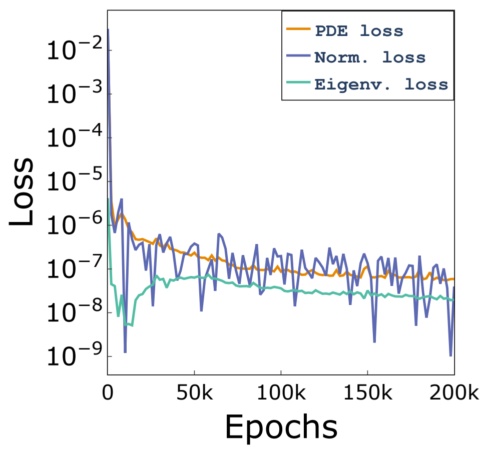


(b) $R = 2$.

Figure 4.22: Evolution of eigenvalue along epochs for the three-dimensional H_2^+ molecule with eigenvalue estimation for different internuclear distances.



(a) $R = 1$.



(b) $R = 2$.

Figure 4.23: Evolution of PDE, normalization, and eigenvalue losses along epochs for the three-dimensional H_2^+ molecule with eigenvalue estimation for different internuclear distances.

constraints on E. For example, if E_{ref} is an upper limit for the true ground state energy, an asymmetric loss can penalize the eigenvalues above this reference energy more strongly. However, the width of the eigenvalue needs to be tuned for every choice of the eigenvalue loss.

Figures 4.20 and 4.21 show the results of the simulations for $R = 1$ and $R = 2$. Comparisons in Fig. 4.20 with the reference solutions obtained by [62] show that our simulations are compatible with the reference solution, even if the simulation with $R = 1$ slightly overestimates the electron density in the region between the two nuclei. This effect is less visible for the $R = 2$ simulations, as the LCAO baseline becomes more correct for more distant nuclei. However, the planar distributions of Fig. 4.21 taken at the $z = 0$ plane show an opposite pattern. In fact, the distribution of Fig. 4.21a has the desired shape on the $z = 0$ plane, while the distribution of Fig. 4.21b shows two additional lobes on the same plane. This inaccuracy is probably caused by a wrong estimation of the eigenvalue for $R = 2$, as happens in the simulation of Fig. 4.13a, but in this case the error is mitigated by the presence of the LCAO baseline. As expected, the final eigenvalue for $R = 2$ overestimates the value in the literature by approximately 2%, while the final eigenvalue for $R = 1$ underestimates the value in the literature by 1%. Simulations have been carried out using the energy values in the literature as a reference energy for the loss $\mathcal{L}_{e2}$, to simplify the training as much as possible. Figure 4.22 shows the evolution of the losses for the two values of R. In both plots, the eigenvalue curve has a transient behaviour in which tends to increase the eigenvalue, but after this phase the curve stabilizes. Also, both of the curves start drifting to the direction of the true eigenvalue (either up or down), so the discrepancies with respect to the literature energies can be in principle reduced with a longer training. Also, the effect of the loss weight ω_E has not been explored for the current simulations, but we can expect to improve the quality of the current H_2^+ simulations with a dedicated hyperparameter tuning.

	ϵ_{L1}	E_{ref}	E	$\|\Psi_{\boldsymbol{\theta}}\|^2_{N'_f}$	Training time
Fig. 4.20a	$2.5 \cdot 10^{-3}$	-1.103	-1.115 ± 0.001	0.975	230 min
Fig. 4.20b	$1.7 \cdot 10^{-3}$	-0.7961	-0.7810 ± 0.0006	0.970	230 min

Table 4.7: Simulation metrics of the three-dimensional H_2^+.

Chapter 5

Conclusions and Outlooks

In this thesis, a physics-informed architecture is proposed to simulate the Schrödinger equation for one-electron quantum systems with a data-free approach.

The simulations of the radial one-dimensional Schrödinger equation for the hydrogen atom demonstrate that the proposed architecture can easily solve one-dimensional quantum problems. With the addition of a trainable variable, the traditional physics-informed approach is extended to retrieve the eigenvalue of the simulated wavefunction, effectively requiring only partial information on the eigenvalue to be able to simulate the system. This additional trainable variable increases the potentiality of the network. Still, it also complicates the training process as many wavefunctions corresponding to different eigenvalues become valid solutions associated with local minima in the loss landscape. A practical way to simplify convergence to the ground state wavefunction in our one-dimensional problem is to choose a sufficiently low initial eigenvalue, but this strategy might not be as robust in more complicated systems.

The possibility of simulating the three-dimensional Schrödinger equation for the hydrogen atom has also been demonstrated with the proposed architecture. The problem is more complex, even if the underlying system is the same, and these simulations are used to understand the difficulties encountered when scaling up the number of dimensions. In particular, the three-dimensional problem with fixed eigenvalue has a divergence at the position of the nucleus. Therefore, the precision of the neural network predictions decreases in proximity to this point. The precision of the simulations improves when the number of network parameters increases and when more training data are sampled inside the computational domain. Thus, the inaccuracies can be alleviated with more computational resources. Similarly to the one-dimensional case, the three-dimensional model with fixed eigenvalue is generalized to estimate the eigenvalue of the solution. The convergence is less robust than in the one-dimensional case, and initializing the eigenvalue variable to a low value does not work in this scenario. The solution is to freeze the eigenvalue for some number of epochs, and it is demonstrated that the quality of the solutions improves.

Finally, the more complex system simulated in this thesis is the hydrogen molecule ion, composed of two protons and one electron. The simulations are greatly simplified by using the Linear Combination of Atomic Orbitals solution as a baseline and by learning the difference from this solution. For this reason, no initialization of the wavefunction is required to converge close to the true eigenvalue when a baseline wavefunction is provided. Instead, the evolution of the eigenvalue is controlled by adding a loss term to restrict the possible eigenvalues. The results of the hydrogen molecular ion are promising but still require more careful analysis to reach the same performance obtained in the simulations of the hydrogen atom in three dimensions.

The natural next step in simulating quantum systems with a physics-informed neural network is the generalization to systems with more than one electron. The physics-informed approach presented in this thesis requires some modifications to be adapted to many-electron systems, such as a neural network architecture designed explicitly for the many-electron problem. Previous work in the literature might inspire the design of such architectures [19, 28]. Moreover, when many electrons are added to the system, an efficient and differentiable integration strategy is required to evaluate the normalization loss. The current integration approach is fast and accurate for the systems treated in this thesis. Still, a careful performance analysis in more than three dimensions is required to assess its scalability. The stability of the eigenvalue estimation can be enhanced by implementing more sophisticated networks with more parameters, as suggested in other works [62].

Overall, the idea of implementing physics-informed learning into ab initio simulations of quantum systems promises to increase the efficiency and flexibility of the simulations, but the research is currently at an early stage. More research in this field will lead to further advances in exploiting physics-informed principles and increase the efficiency of quantum systems simulations.

Bibliography

[1] David J. Griffiths and Darrell F. Schroeter. *Introduction to Quantum Mechanics*. 3rd ed. Cambridge University Press, 2018. ISBN: 9781316995433.

[2] W. Heisenberg. "Über quantentheoretische Umdeutung kinematischer und mechanischer Beziehungen." In: *Zeitschrift für Physik* 33 (1925), pp. 879–893. DOI: 10.1007/BF01328377.

[3] E. Schrödinger. "An Undulatory Theory of the Mechanics of Atoms and Molecules". In: *Phys. Rev.* 28 (6 1926), pp. 1049–1070. DOI: 10.1103/PhysRev.28.1049.

[4] David J. Thouless. *The quantum mechanics of many-body systems*. Courier Corporation, 2014. ISBN: 978-0126915600.

[5] Alexander L. Fetter and John Dirk Walecka. *Quantum theory of many-particle systems*. Courier Corporation, 2012. ISBN: 9780070206533.

[6] Norbert Nemec. "Diffusion Monte Carlo: Exponential scaling of computational cost for large systems". In: *Physical Review B* 81.3 (2010), p. 035119. DOI: https://doi.org/10.1103/PhysRevB.81.035119.

[7] Brian L. Hammond, William A. Lester Jr, and Peter James Reynolds. *Monte Carlo methods in ab initio quantum chemistry*. World Scientific, 1994. ISBN: 9789810203221.

[8] W. Kohn and L.J. Sham. "Self-Consistent Equations Including Exchange and Correlation Effects". In: *Phys. Rev.* 140 (4A 1965), A1133–A1138. DOI: 10.1103/PhysRev.140.A1133.

[9] R.G. Parr and Y. Weitao. *Density-Functional Theory of Atoms and Molecules*. International Series of Monographs on Chemistry. Oxford University Press, 1994. ISBN: 9780195357738.

[10] Román Orús. “A practical introduction to tensor networks: Matrix product states and projected entangled pair states”. In: *Annals of Physics* 349 (2014), pp. 117–158. DOI: https://doi.org/10.1016/j.aop.2014.06.013.

[11] Simone Montangero. *Introduction to tensor network methods : numerical simulations of low-dimensional many-body quantum systems*. Springer Cham, Switzerland, 2018. ISBN: 9783030014094.

[12] Isaac E. Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. “Artificial neural network methods in quantum mechanics”. In: *Computer Physics Communications* 104.1-3 (1997), pp. 1–14. DOI: https://doi.org/10.1016/S0010-4655(97)00054-4.

[13] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, et al. “Machine learning and the physical sciences”. In: *Reviews of Modern Physics* 91.4 (2019), p. 045002. DOI: https://doi.org/10.1103/RevModPhys.91.045002.

[14] Giuseppe Carleo and Matthias Troyer. “Solving the quantum many-body problem with artificial neural networks”. In: *Science* 355.6325 (2017), pp. 602–606. DOI: https://doi.org/10.1126/science.aag2302.

[15] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378 (2019), pp. 686–707. DOI: https://doi.org/10.1016/j.jcp.2018.10.045.

[16] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, et al. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. DOI: https://doi.org/10.1038/s42254-021-00314-5.

[17] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, et al. “Automatic Differentiation in Machine Learning: a Survey”. In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43. URL: http://jmlr.org/papers/v18/17-468.html.

[18] Paul Adrien Maurice Dirac. “Quantum mechanics of many-electron systems”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 123.792 (1929), pp. 714–733. DOI: https://doi.org/10.1098/rspa.1929.0094.

[19] David Pfau, James S. Spencer, Alexander G.D.G. Matthews, et al. “Ab initio solution of the many-electron Schrödinger equation with deep neural networks”. In: *Physical Review Research* 2.3 (2020), p. 033429. DOI: `https://doi.org/10.1103/PhysRevResearch.2.033429`.

[20] M. Born and R. Oppenheimer. “Zur Quantentheorie der Molekeln”. In: *Annalen der Physik* 389.20 (1927), pp. 457–484. DOI: `https://doi.org/10.1002/andp.19273892002`.

[21] W.M.C. Foulkes, Lubos Mitas, R.J. Needs, et al. “Quantum Monte Carlo simulations of solids”. In: *Reviews of Modern Physics* 73.1 (2001), p. 33. DOI: `https://doi.org/10.1103/RevModPhys.73.33`.

[22] John A. Pople and Robert K. Nesbet. “Self-consistent orbitals for radicals”. In: *The Journal of Chemical Physics* 22.3 (1954), pp. 571–572. DOI: `https://doi.org/10.1063/1.1740120`.

[23] Ingrid von Glehn, James S. Spencer, and David Pfau. *A Self-Attention Ansatz for Ab-initio Quantum Chemistry.* 2023. arXiv: `2211.13672 [physics.chem-ph]`.

[24] Robert Jastrow. “Many-body problem with strong forces”. In: *Physical Review* 98.5 (1955), p. 1479. DOI: `https://doi.org/10.1103/PhysRev.98.1479`.

[25] Richard P. Feynman and Michael Cohen. “Energy spectrum of the excitations in liquid helium”. In: *Physical Review* 102.5 (1956), p. 1189. DOI: `https://doi.org/10.1103/PhysRev.102.1189`.

[26] Jiequn Han, Linfeng Zhang, and E. Weinan. “Solving many-electron Schrödinger equation using deep neural networks”. In: *Journal of Computational Physics* 399 (2019), p. 108929. DOI: `https://doi.org/10.1016/j.jcp.2019.108929`.

[27] Kenny Choo, Antonio Mezzacapo, and Giuseppe Carleo. “Fermionic neural-network states for ab-initio electronic structure”. In: *Nature communications* 11.1 (2020), p. 2368. DOI: `https://doi.org/10.1038/s41467-020-15724-9`.

[28] Jan Hermann, Zeno Schätzle, and Frank Noé. “Deep-neural-network solution of the electronic Schrödinger equation”. In: *Nature Chemistry* 12.10 (2020), pp. 891–897. DOI: `https://doi.org/10.1038/s41557-020-0544-y`.

[29] Honghui Shang, Chu Guo, Yangjun Wu, et al. *Solving Schrödinger Equation with a Language Model.* 2023. arXiv: `2307.09343 [quant-ph]`.

[30] Les Atlas, Toshiteru Homma, and Robert Marks. “An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification”. In: *Neural Information Processing Systems*. Ed. by D. Anderson. American Institute of Physics, 1987.

[31] Yann LeCun and Yoshua Bengio. “Convolutional Networks for Images, Speech, and Time Series”. In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, 255–258. ISBN: 0262511029.

[32] R. Venkatesan and B. Li. *Convolutional Neural Networks in Visual Computing: A Concise Guide*. Data-Enabled Engineering. CRC Press, 2017. ISBN: 9781498770408.

[33] Xue Ying. “An Overview of Overfitting and its Solutions”. In: vol. 1168. 2. IOP Publishing, 2019, p. 022022. DOI: `10.1088/1742-6596/1168/2/022022`.

[34] Michael M. Bronstein, Joan Bruna, Yann LeCun, et al. “Geometric deep learning: going beyond euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. DOI: `10.1109/MSP.2017.2693418`.

[35] Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, et al. “Gauge equivariant convolutional networks and the icosahedral CNN”. In: *International conference on Machine learning*. PMLR. 2019, pp. 1321–1330.

[36] Pengzhan Jin, Zhen Zhang, Aiqing Zhu, et al. “SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems”. In: *Neural Networks* 132 (2020), pp. 166–179. DOI: `https://doi.org/10.1016/j.neunet.2020.08.017`.

[37] Zeyu Liu, Yantao Yang, and Qingdong Cai. “Neural network as a function approximator and its application in solving differential equations”. In: *Applied Mathematics and Mechanics* 40.2 (2019), pp. 237–248. DOI: https://doi.org/10.1007/s10483-019-2429-8.

[38] Pola Lydia Lagari, Lefteri H. Tsoukalas, Salar Safarkhani, et al. “Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions”. In: *International Journal on Artificial Intelligence Tools* 29.05 (2020), p. 2050009. DOI: https://doi.org/10.1142/S0218213020500098.

[39] Isaac E. Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. “Artificial neural networks for solving ordinary and partial differential equations”. In: *IEEE transactions on neural networks* 9.5 (1998), pp. 987–1000. DOI: https://doi.org/10.1109/72.712178.

[40] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, et al. “Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data”. In: *Journal of Computational Physics* 394 (2019), pp. 56–81. DOI: https://doi.org/10.1016/j.jcp.2019.05.024.

[41] Henry Jin, Marios Mattheakis, and Pavlos Protopapas. *Unsupervised Neural Networks for Quantum Eigenvalue Problems*. 2020. arXiv: 2010.05075 [physics.comp-ph].

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[43] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (GELUs)”. In: (2023). arXiv: 1606.08415 [cs.LG].

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

[45] Liu Yang, Xuhui Meng, and George Em Karniadakis. "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data". In: *Journal of Computational Physics* 425 (2021), p. 109913. DOI: `https://doi.org/10.1016/j.jcp.2020.109913`.

[46] Martín Abadi, Paul Barham, Jianmin Chen, et al. "TensorFlow: a system for Large-Scale machine learning". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[47] Adam Paszke, Sam Gross, Francisco Massa, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, et al. Vol. 32. Curran Associates, Inc., 2019.

[48] Raul González-García, Ramiro Rico-Martìnez, and Ioannis G. Kevrekidis. "Identification of distributed parameter systems: A neural net based approach". In: *Computers & chemical engineering* 22 (1998), S965–S968. DOI: `https://doi.org/10.1016/S0098-1354(98)00191-4`.

[49] R. Rico-Martinez, I. Kevrekidis, and K. Krischer. *Nonlinear system identification using neural networks: dynamics and instabilities*. 1995.

[50] Zheyuan Hu, Khemraj Shukla, George Em Karniadakis, et al. *Tackling the Curse of Dimensionality with Physics-Informed Neural Networks*. 2023. arXiv: `2307.12306 [cs.LG]`.

[51] Rafael Bischof and Michael Kraus. "Multi-Objective Loss Balancing for Physics-Informed Deep Learning". In: (2021). arXiv: `2110.09813 [cs.LG]`.

[52] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, et al. "Characterizing possible failure modes in physics-informed neural networks". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 26548–26560.

[53] Stefano Markidis. "The old and the new: Can physics-informed deep-learning replace traditional linear solvers?" In: *Frontiers in big Data* 4 (2021), p. 669097. DOI: `10.3389/fdata.2021.669097`.

[54] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, et al. "Scientific machine learning through physics–informed neural networks: Where we are and what's next". In: *Journal of Scientific Computing* 92.3 (2022), p. 88. DOI: https://doi.org/10.1007/s10915-022-01939-z.

[55] Francois Chollet. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[56] Roy Frostig, Matthew James Johnson, and Chris Leary. "Compiling machine learning programs via high-level tracing". In: *Systems for Machine Learning* 4.9 (2018).

[57] Lu Lu, Xuhui Meng, Zhiping Mao, et al. "DeepXDE: A deep learning library for solving differential equations". In: *SIAM review* 63.1 (2021), pp. 208–228. DOI: 10.1137/19M1274067.

[58] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, et al. "NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework". In: *International conference on computational science*. Springer. Springer International Publishing, 2021, pp. 447–461. DOI: https://doi.org/10.1007/978-3-030-77977-1_36.

[59] Feiyu Chen, David Sondak, Pavlos Protopapas, et al. "Neurodiffeq: A python package for solving differential equations with neural networks". In: *Journal of Open Source Software* 5.46 (2020), p. 1931. DOI: https://doi.org/10.21105/joss.01931.

[60] Kailai Xu and Eric Darve. "ADCME: Learning Spatially-varying Physical Fields using Deep Neural Networks". In: (2020). arXiv: 2011.11955 [math.NA].

[61] Henry Jin, Marios Mattheakis, and Pavlos Protopapas. "Physics-Informed Neural Networks for Quantum Eigenvalue Problems". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9891944.

[62] Marios Mattheakis, Gabriel R. Schleder, Daniel T. Larson, et al. *First principles physics-informed neural network for quantum wavefunctions and eigenvalue surfaces*. 2022. arXiv: 2211.04607 [cs.LG].

[63] Shengze Cai, Zhicheng Wang, Frederik Fuest, et al. "Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks". In: *Journal of Fluid Mechanics* 915 (2021), A102. DOI: `doi:10.1017/jfm.2021.135`.

[64] J.M. Hammersley and D.C. Handscomb. *Monte carlo methods.* Springer Netherlands, 1964. ISBN: 9780412158704.

[65] Yifan Wang, Pengzhan Jin, and Hehu Xie. *Tensor Neural Network and Its Numerical Integration.* 2023. arXiv: `2207.02754 [math.NA]`.

[66] Yifan Wang, Yangfei Liao, and Hehu Xie. *Solving Schrödinger Equation Using Tensor Neural Network.* 2022. arXiv: `2209.12572 [physics.comp-ph]`.

[67] Rudolf Schürer. "A comparison between (quasi-) Monte Carlo and cubature rule based methods for solving high-dimensional integration problems". In: *Mathematics and computers in simulation* 62.3-6 (2003), pp. 509–517. DOI: `https://doi.org/10.1016/S0378-4754(02)00250-1`.

[68] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256. URL: `https://proceedings.mlr.press/v9/glorot10a.html`.

[69] H. Wind. "Electron energy for H2+ in the ground state". In: *The journal of chemical physics* 42.7 (1965), pp. 2371–2373. DOI: `https://doi.org/10.1063/1.1696302`.