Machine-Learning for Lattice Models in and out of Equilibrium

Johannes Sandberg

LUND
UNIVERSITY

Thesis submitted for the Degree of Master of Science
Duration: 12 months

Supervisor: Claudio Verdozzi
Cosupervisor: Najmeh Abiri

Division of Mathematical Physics
Department of Physics
Faculty of Science

Submitted: December 14 2020

Acknowledgments

First I would like to thank Claudio Verdozzi for his excellent work supervising this thesis, for always being available to give advice, and for introducing me to this interesting topic. I would also like to thank my co-supervisor Najmeh Abiri for sharing her knowledge on the machine-learning side of things, especially for the time-dependent part of this project. Finally I would like to thank all the wonderful people with whom I have shared an office at mathematical physics, for making my time working on this thesis that much more enjoyable.

Abstract

Machine-learning methods have in recent years seen a great deal of use in condensed matter physics. In this thesis we apply such methods, specifically machine-learning with artificial neural networks, to the equilibrium and non-equilibrium description of the Hubbard and Hubbard-Holstein models.

In the framework of ground-state density functional theory we reproduce results from the literature regarding machine learning for the energy functional of a Hubbard chain, and show that the approach also works for predicting the exchange-correlation potential, and is applicable also in the Hubbard-Holstein model.

Working in the framework of many-body Green's functions, we present a way to train a neural network on the self-energy of the Hubbard dimer. This self-energy is tested against the exact one, with which excellent agreement is found.

Moving on to time-dependent density functional theory, we try to represent the history-dependent exchange-correlation potential of a Hubbard dimer, subject to a specific type of time-dependent perturbation. Initial attempts using Long Short-Term Memory (LSTM) networks fail to go beyond even a simple adiabatic approximation. Applying instead a dense neural network found in the literature we achieve excellent results for this task, despite a lack of explicit history-dependence in the functional.

Contents

List of Abbreviations

DFT Density Functional Theory

KS Kohn-Sham

xc Exchange-Correlation

SOFT Site-Occupation Density Functional Theory

TD Time-Dependent

TDDFT Time-Dependent Density Functional Theory

GF Green's Function

ANN Artificial Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

RELU Rectified Linear Unit

MSE Mean Squared Error

RMSE Root Mean Squared Error

MLP Multilayer Perceptron

LSTM Long Short-Term Memory

1 Introduction

One of the major problems in modern physics is the many-body problem. Namely, how can we study and understand systems of many interacting quantum-mechanical particles. For a single particle moving in a potential the Schrödinger equation can be straightforwardly solved, but unfortunately many systems of interest involve more than one particle. If the particles are non-interacting the solution to the many-body Schrödinger equation will be a Slater determinant of single-particle orbitals, or a discriminant for bosons. When the particles interact this is no longer the case. The interaction will couple different determinant states, hence the state will be a superposition of Slater determinants. In principle, for continuum problems, there will be infinitely many possible Slater determinants, and even if we can truncate the Hilbert state the necessary number of such states to consider quickly grows to make this approach infeasible for large systems. Many systems in nature, and of practical interest for technological applications, involve many particles. For instance, an electron in a solid will feel the Coulomb field of all other electrons in the solid and a full description of matter requires one to take into account such interactions. Furthermore, many important phenomena are inherently tied to many-body effects, such as superconductivity and Mott insulator transitions.

Model Systems - One common approach to study a complicated system in physics is to replace it with a simpler model, which captures the essential features of the original system which one wishes to study. A common class of such model systems are lattice models, wherein the model is defined only on the discrete sites of a lattice, as opposed to a continuum. Many such models have found use in theoretical physics. Among these, one of the most important ones is the Hubbard Model, and in what follows we will focus mainly on this one. This model describes particles moving on a lattice, and allows for interaction between particles which share a lattice site. Specifically we will consider the fermionic version of this model, where the particles in question are fermions. Despite ignoring any long-range interactions or tunneling, and allowing only one orbital per site, the Hubbard model is one of

the most commonly used models for studying highly correlated electrons. It gives rise to many interesting many-body phenomena, and has also been used to study and benchmark numerical methods also applicable to more complex systems. A similar model which we will also consider is the Hubbard-Holstein model, which extends the Hubbard model by including a basic description of phonons, and electron-phonon interaction. This model has been used to study the interplay between electron-electron and electron-phonon interactions in a minimal setting. Studying these two interactions in combination is relevant for a complete understanding of real materials. We will discuss the Hubbard model and the Hubbard-Holstein models in sections 1.1 and 1.2 respectively.

Density Functional Theory - While model systems allow for a simple description and a qualitative understanding of many-body systems, obtaining quantitative predictions for real world systems is much more difficult. Furthermore, even simple model systems become prohibitively difficult to solve for large system sizes. It is thus relevant to find efficient calculation methods which can be extended to large systems. One of the most successful of such calculation methods is Density Functional Theory (DFT). In DFT one foregoes the traditional wave-functions and instead reformulates quantum mechanics in terms of the particle density. The wave-function will depend on the location and quantum numbers of each particle in the system, and as the number of particles grows the wave-function can become quite difficult to manage. Meanwhile, the density will be a function of just a single set of spatial coordinates, regardless of system size. Using the density as the fundamental variable therefore allows for the study of much larger systems than otherwise possible in a wave-function approach. Most practical applications of DFT use the Kohn-Sham method, in which one maps the interacting system to a non-interacting one with the same density. The problem then is to find the exchange-correlation (xc) potential as a functional of the density, which can then used to construct this auxiliary non-interacting system. In the time-dependent version of DFT (TDDFT) there is also the added complication of the xc-potential depending on the entire history of the density, and on the initial state of the system. This history-dependence, which is a consequence of going from a wave-function description to a density one, is notoriously difficult to take into account, and most applications simply ignore it. Finding a truly history-dependent functional is of great interest to practical TDDFT, and even a functional valid only for a lattice model would provide much needed insight into the general case. One of the goals of this thesis is to recreate this xc-potential functional for lattice models, using machine-learning methods. We will be discussing DFT, and TDDFT in more detail in sections 1.3 and 1.4 respectively, in particular the lattice versions

applicable to the Hubbard and Hubbard-Holstein models.

Green's Functions - One of the most powerful approaches to the many-body problem is that of many-body Green's Functions (GF). If the single-particle GF of a many-body system is known it can be used to calculate the expectation values of any single-particle operator, as well as the total energy of the system. The spectral form the GF also encodes the entire excitation spectrum of the many-body system in the form of its poles. As a result, it is of great interest to be able to calculate the GF, and the typical way to do this is using the many-body self-energy. The self-energy connects the many-body GF to the non-interacting GF via the Dyson equation. If the self-energy is known one can therefore calculate the non-interacting GF, which is much simpler than the interacting one, and then obtain the full interacting one. The self-energy is in general not known exactly, and one therefore must turn to approximations. Such approximations are often constructed from the perturbative expansion of the self-energy, by selectively neglecting certain terms. One important such approximation is the famous GW approximation [1]. If one considers the GF in frequency space it will be a sum of simple poles corresponding to the excitation energies of the system. Likewise one can represent the self-energy as a sum of simple poles, and for a lattice model of finite size the number of such poles will be finite. A machine-learning approach for the self-energy of lattice models can be attempted. Specifically one can try to train a neural network to predict the poles of the self-energy, and their residues, as this is then enough to reconstruct the exact self-energy. We will discuss many-body GFs and the self-energy in more detail in section 1.5.

Machine Learning - As we have slightly touched upon, the aim of this work is to explore the use of machine-learning for lattice models, specifically to represent the static and time-dependent xc-potential and the many-body self-energy. Machine-learning encompasses a wide range of methods for solving complicated problems. At the core of such methods is the idea that instead of coming up with an algorithm that can perform some difficult task, we can instead create an algorithm that can learn from a set of data how to solve the task. During the last decade there has been an increased interest in machine-learning, and machine-learning methods have seen great success in many different areas of science and technology. One of the most important approaches to machine-learning today is that of Artificial Neural Networks (ANN). Machine-learning with ANNs has been successfully applied to image classification, natural language processing, generative models, and many other problems that have previously been consider very difficult to solve. There has also been work done on applying machine-learning to many-body quantum mechanics. An early example is given in [2], where the self-energy

of the Anderson model is studied using machine-learning, albeit not using ANN. When it comes to DFT recent work includes [3] where the xc-potential of the Lieb Wu solution to the Hubbard model is learned for use in a local density approximation. In [4] the KS scheme is completely bypassed in favor for an orbital-free density functional using machine-learning, again using a different approach than ANN. Another recent example is [5] where the exact ground-state xc-potential of the Hubbard model is represented using an ANN approach. In this thesis we will be recreating this result, as well as applying this same approach to the Hubbard-Holstein model. Recently there has also been a lot of interest in using machine-learning for time-dependent systems. Other approaches than TDDFT has been considered [6], but the idea of using machine-learning for history-dependent xc-potentials is novel. During the writing of this thesis the first publication of such machine-learning TDDFT came to our attention [7]. To our knowledge we are the first to consider this approach for the Hubbard model, as well as the first to use recurrent neural networks for time-dependent xc-potentials. We will discuss ANNs in more detail in section 1.6.

Structure of this Thesis - For the remainder of this chapter we will go through all the necessary background for this thesis, i.e. we will go through the physical models under study, the methods of DFT, TDDFT, and many-body Green's functions, and finally machine-learning and ANNs.

In chapter 2 we will outline the calculations used to generate data for the machine-learning. Then, in chapter 3 we will go through the results of this thesis, in the following order. First the machine-learning for the xc-potential in ground-state DFT, then the machine-learning for the Hubbard dimer self-energies, and finally machine-learning for the history-dependent xc-potential in TDDFT for the Hubbard dimer. In the final chapter of the thesis we will give some concluding remarks and outlook.

1.1 The Hubbard Model

Originally introduced in 1963 [8] to study the electronic properties of narrow-band metals, the Hubbard Model is one of the most common models used to study strongly correlated electronic systems. The Hubbard model, in its simplest form, is given by the Hamiltonian

$$H = -J \sum_{\langle ij \rangle, \sigma} \left(c^\dagger_{i,\sigma} c_{j,\sigma} + \text{h.c.} \right) + \sum_{i,\sigma} v_i \hat{n}_{i,\sigma} + U \sum_i \hat{n}_{i,\uparrow} \hat{n}_{i,\downarrow} \tag{1.1}$$

with $\langle ij \rangle$ denoting nearest-neighbor pairs, and the annihilation and creation operators $c_{i,\sigma}$ and $c^\dagger_{i,\sigma}$ satisfying the standard fermionic anti-commutation

relations

$$\{c_{i,\sigma}, c_{j,\sigma'}\} = 0 \qquad \left\{c_{i,\sigma}, c^{\dagger}_{j,\sigma'}\right\} = \delta_{ij}\delta_{\sigma\sigma'}. \tag{1.2}$$

We can interpret the first part of the Hamiltonian as describing electrons moving between adjacent sites on the lattice with a rate J, the second term as describing an on-site energy or coupling to a constant external potential v, and the third term as describing a short range interaction between opposite spin electrons at the same site with interaction strength U. The behavior of the model depends on the ratio U/J. As this ratio goes to infinity we have the atomic limit with strongly localized states, while in the opposite limit we obtain the tight-binding Hamiltonian characterized by delocalized electron states. For intermediate values we have a competition between these two behaviors.

Despite it being in many ways an enormous over-simplification compared to any real many-electron system, this model does retain many of the essential features, and gives rise to a wide array of interesting behavior in its own right. In the one-dimensional homogeneous version of the model there exists an exact solution using the Bethe Ansatz [9], making it one of the few exactly solvable interacting quantum many-body systems. For higher dimensions, and in the inhomogeneous case, one has to resort to numerical methods and approximations. Among many other things the Hubbard model has been applied to the study of nano-scale structures [10], and fermionic superfluids [11], and it has been used as a model for cold atoms in optical traps [12]. The model has also found use as a test system for benchmarking computational approaches such as, for instance, density functional theory [13]. It can also serve as basis for more complicated models, one of which will be discussed in the next section.

1.2 The Hubbard-Holstein Model

The Hubbard-Holstein model combines the Hubbard model, discussed in the previous section, with the Holstein model [14]. In this model we attach phonons to the lattice sites, allowing interaction between phonons and electrons at the same site. The phonons are treated as simple harmonic oscillators, following the Hamiltonian

$$H_{ph} = \omega \sum_i b_i^{\dagger} b_i, \tag{1.3}$$

with the creation and annihilation operators b_i and $b_i^\dagger$ satisfying the standard bosonic commutation relations,

$$[b_i, b_j] = 0 \qquad \left[b_i, b_j^\dagger\right] = \delta_{ij}, \tag{1.4}$$

and ω being the phonon frequency. The electron-phonon interaction is given by

$$H_{e-ph} = \sqrt{2}g \sum_i (n_{i,\uparrow} + n_{i,\downarrow}) x_i, \tag{1.5}$$

with the phonon coordinate operator defined as $x_i = (b_i^\dagger + b_i)/\sqrt{2}$, and g being the electron-phonon interaction strength.

We can interpret this model as describing a lattice of diatomic molecules. In this interpretation the phonons correspond to intramolecular vibrations. The interaction term can then be interpreted as saying that the binding energy of each molecule is linear in the distance between the atoms, i.e. the coordinates x_i.

Originally the Holstein model was introduced in 1959 in order to study polarons [14], a type of quasiparticle arising from the interaction between electrons and ions in solids. In the pure Holstein model there is however no direct electron-electron interaction, in other words $U = 0$. The Hubbard-Holstein model on the other hand allows for the study of both the electron-electron and electron-phonon interactions, and the interplay between these interactions. A good motivation for studying such a model is because it is believed that the interplay between these interactions is central to a proper understanding of high-temperature superconductivity[15, 16].

1.3 DFT

The quantum many-body problem is difficult to solve. Even in simple models, such as the Hubbard model, the Hilbert space grows rapidly with the size of the system. This poses an issue since even for relatively small systems one soon reaches a point where the computations become too heavy for even the fastest supercomputers. As an example we can consider a Hubbard model with M sites, and N_σ electrons for each spin channel. The size of the Hilbert-space will then be

$$D_H = \binom{M}{N_\uparrow}\binom{M}{N_\downarrow} \tag{1.6}$$

where the two factors are the number of ways to distribute N_σ electrons of a given spin on the M lattice sites. If we take, for example, a 20 site

model at half-filling then the Hilbert Space dimension will be over 10^{10}, and to solve this system we would need to diagonalize a larger than $10^{10} \times 10^{10}$ matrix. Diagonalizing large matrices is a notoriously difficult task, and in practical calculations one often wish to study much larger systems in order to explore the thermodynamic limit, i.e. $L \to \infty$. It is clear that other approaches are required. One such approach to the many-body problem is Density Functional Theory (DFT), which we will now discuss in detail, first in general, and then for the specific cases of the Hubbard and Hubbard-Holstein models. Finally we will go into some detail also for the time-dependent version of this method.

DFT is a reformulation of quantum mechanics, in terms of the electron density $n(x,t)$, the foundations of which were laid the 1960s by Hohenberg, Kohn, and Sham [17, 18], and for which Kohn was awarded the 1998 Nobel prize in chemistry [19]. Since its inception DFT has been successfully applied to a variety of situations in condensed matter physics and chemistry. The original DFT, has since been extended to include among other things finite temperature [20], and time-dependent potentials [21].

At the core of DFT are the two Hohenberg Kohn (HK) theorems [17], the first of which states that there is a one-to-one mapping between the external potential and the ground-state density. The second theorem states that there exists a universal functional of the density, $F[n]$, such that the minima of

$$E[n] = F[n] + \int v(\mathbf{r})n(\mathbf{r})d\mathbf{r} \tag{1.7}$$

is the ground-state energy, and is obtained at the exact ground-state density. Here we mean by universal functional that the functional does not depend on the particular system considered, i.e. on the potential $v(\mathbf{r})$. The functional will however depend on the masses of the particles, and the nature of their interaction. When we consider DFT for lattice models the functional will also depend on the topology of the lattice. If we know this functional, or a good approximation thereof, we do not need to work with the complicated many-body wave-function, but can instead use the much simpler particle density.

While the HK theorems guarantee the existence of the function F, in practice we need to approximate it. It turns out that this is a difficult task, in particular when it comes to finding a kinetic energy functional of the density. Most practical applications of DFT use the so called Kohn-Sham (KS) method [18], which avoids this issue of finding a kinetic energy functional by mapping the system onto an auxiliary non-interacting system with the same ground-state density. We rewrite the energy as

$$E[n] = T_{KS}[n] + V_{ext}[n] + E_H[n] + E_{xc}[n] \tag{1.8}$$

where $T_{KS}[n]$ is the kinetic energy of a non-interacting system with density n, $V_{ext}[n]$ is the contribution from the external potential, $E_H[n]$ is the Hartree energy, and E_{xc} contains all other contributions to the energy. This will be the same energy as for a non-interacting system of particles subjected to the potential

$$v_{KS} = v + v_H + v_{xc} \tag{1.9}$$

with v being the external potential of the interacting system, v_H being the Hartree potential, and v_{xc} being the exchange-correlation potential, defined as the functional derivatives of E_H and E_{xc} respectively with respect to the density.

The HK theorems hold for any density which is the ground-state density for some potential, thus referred to as v-representable. It is known that not all functions which are valid densities satisfy this condition. Furthermore, in order for the KS method to be exact we require that the density be non-interacting v-representable, meaning that it can be represented as the ground-state density of a non-interacting system with some potential. For a review of the concept and open issues about v-representability, see [22]. Here, being interested in lattice systems, we consider that under rather general conditions non-interacting v-representability applies [23]. The problem now is to find v_{xc}. In this work we will be using Artificial Neural Networks to construct the exact xc-potential from exact diagonalization calculations.

1.3.1 DFT for the Hubbard Model

While DFT was originally formulated for continuum quantum mechanics, it can be used for lattice models as well. Perhaps the most straightforward such formulation is the Site-Occupation Functional Theory (SOFT) of Gunnarsson and Schönhammer, being the first attempt at a lattice version of DFT [24]. Besides SOFT there exists other formulations of ground-state DFT on a lattice [25, 26], but these will not be considered here. For a general review of DFT for lattice models we refer to [13].

In SOFT we have as the basic quantity the site-occupation $n_i = n_{i,\uparrow} + n_{i,\downarrow}$, which takes the role of the density. One can then prove a HK theorem, as in continuous DFT, which states that the ground-state energy is a functional of the site-occupation n. Specifically one can prove that the ground-state energy is

$$E[n] = F[n] + \sum_i v_i n_i, \tag{1.10}$$

with F again being a universal functional. It should be mentioned, however, that while we refer to this as a functional, in lattice DFT this is technically a

function of the site-occupations, $F[n] = F(n_1, n_2, \ldots, n_L)$ for a 1D lattice of length L. Likewise all the functional derivatives will be replaced by standard partial derivatives.

As in the continuum case we have the Kohn-Sham scheme where we rewrite this energy as

$$E\left[n\right] = T_{KS}\left[n\right] + \sum_i v_i n_i + E_H\left[n\right] + E_{xc}\left[n\right] \tag{1.11}$$

where E_H is the Hartree energy, and E_{xc} is the exchange-correlation energy. The contribution T_{KS} is the kinetic energy of an auxiliary non-interacting system. Minimization of this energy leads us to the same equations as for a non-interacting system subject to the potential

$$v_{KS,i} = v_i + v_{H,i} + v_{xc,i} \tag{1.12}$$

with $v_{H,i} = \partial E_H / \partial n_i$ being the Hartree potential, and $v_{xc,i} = \partial E_{xc} / \partial n_i$ being the xc-potential. In our case we have the same number of spin-up electrons as spin-down, and so $n_i = 2n_{i,\sigma}$ due to spin symmetry. We therefore have explicitly $v_{H,i} = Un_i/2$.

Assuming that non-interacting v-representability holds, the formalism thus far is exact, and if we knew the exact xc-potential we could solve the many-body problem exactly. Unfortunately we generally do not know the exact xc-potential, and so we typically have to resort to approximations. In continuum DFT the archetypal approximation is the Local Density Approximation (LDA), originally introduced by Kohn and Sham [18], which replaces the xc-potential locally by that of the uniform electron gas. The exact solution of the homogeneous 1D Hubbard model [9] opens up for a similar approximation in 1D lattice DFT, namely the Bethe Ansatz LDA (BALDA), introduced originally in [27]. In order to understand generally the properties of lattice DFT there has been considerable effort to reverse-engineer the exact xc-potential [43].

1.3.2 DFT for the Hubbard Holstein Model

Like for the Hubbard model, we can formulate a DFT for the Hubbard-Holstein model. The main complication is that we need to include the phonons, and account for their interaction with the electrons. In this HH Model DFT we have as the basic variables the site-occupation n_i, as for the Hubbard model, and the phonon coordinates x_i. Again one can prove a HK theorem which states that the total energy is a functional of these two variables, $E\left[n, x\right]$, which is minimized by the correct ground-state values of these quantities [28].

For the Kohn-Sham scheme the auxiliary system will now have non-interacting phonons, as well as electrons, and we require that both the occupation n_{KS} and the coordinates x_{KS} be the same as in the original system. The energy can then be decomposed as

$$E[n,x] = T_{KS,e}[n] + T_{KS,ph}[x] + \sum_i v_i n_i + E_{H,e}[n] + \\ + E_{H,e-ph}[n,x] + E_{xc}[n,x] \quad (1.13)$$

where $T_{KS,e}, T_{KS,ph}$ are the kinetic energy of the electrons and phonons in the auxiliary system, $E_{H,e}$ is the electron-electron Hartree energy, $E_{H,e-ph}$ is the electron-phonon Hartree energy, and E_{xc} is the xc-energy, which now also depends on the phonon coordinate. Again, minimizing this will give us the following Kohn-Sham potentials

$$v_{KS,i}[n,x] = v_i + v_{H,e,i}[n] + v_{H,e-ph,i}[x] + v_{xc}[n,x] \quad (1.14)$$

$$\eta_{KS,i}[n,x] = \eta_{H,i}[n] + \eta_{xc,i}[n,x]. \quad (1.15)$$

Here the the electronic potentials in the first equation are defined as the derivative of the corresponding energy term in (Eq. 1.13) with respect to the density. For example $v_{H,e,i} = \partial E_{H,e}/\partial n_i$. Likewise the phononic potentials, denoted by η, are defined as the derivatives with respect to the phonon coordinates x_i. The phononic Hartree potential is $\eta_{H,i} = g n_i$, and it turns out that $\eta_{xc} = 0$ [28]. Since we have $n_i = 2n_{i\sigma}$ we have $v_{H,e,i} = U n_i/2$, and $v_{H,e-ph,i} = \sqrt{2} g x$.

1.4 TDDFT

The density functional theory we have discussed thus far has all been for time-independent ground-state problems, for which DFT was originally formulated. We will now move on to Time-Dependent DFT (TDDFT), which allows for the study of time-dependent problems, as well as studying non ground-state properties. The basic theorem of TDDFT is the Runge-Gross Theorem, which is analogous to the HK theorem of ground-state DFT [21]. This theorem states that the state at time t is a functional of the initial state, at time t', and the time-dependent density $n(\mathbf{r}, t'')$ at all previous times $t' \leq t'' \leq t$. A striking feature of the time-dependent problem is the fact that we can have memory effects. As with the ground-state theory we can construct TDDFT for lattice models. In our case we are only concerned with TDDFT for the Hubbard Dimer, for which non-interacting v-representability is not an issue [29].

Just as in ground-state DFT, constructing a direct functional from the density is difficult, and typically one resorts to a Kohn-Sham approach, which readily generalizes to the time-dependent case. Again we make the ansatz of a non-interacting system which gives rise to the same density as in the interacting model. In this case, however, we assert that the time-dependent density must be the same, and as a consequence we require the Kohn-Sham potential to be time-dependent. As in the time-independent Kohn-Sham scheme we can split the Kohn-Sham potential into three components, namely the external potential, Hartree potential, and xc-potential. The external potential is known, and the Hartree potential is as in the non-interacting case only with the time-dependent density, but neither of these involve any kind of memory effect, depending only locally on time. It follows that all the memory effects enter into the problem via the xc-potential, which must be a functional of the density at all previous times.

As in ground-state DFT we have to resort to approximations for the xc-potential, now with the added issue of finding a way to include the dependence on the history of the system. Finding such approximations which account for memory effects, and non-locality in time, is a difficult task. Most work to date have been using some form of adiabatic approximation, in which we simply ignore this non-locality in time, instead regarding the xc-potential at a given time as a functional of the density at that time alone. Specifically, most TDDFT applications use the Adiabatic Local Density Approximation, or ALDA, which combines the adiabatic approximation with the local density approximation. However, there have been some effort to come up with non-adiabatic xc-potentials [30], as well as efforts to find the exact time-dependent xc-potential in simple cases [29, 31]. The need for such non-adiabatic approaches is illustrated by the shortcomings of the adiabatic approximation in for instance charge transfer [32]. Generally the adiabatic approximations is insufficient in presence of fast perturbations [33]. In this thesis we will not be using the adiabatic approximation, but instead we will be using a reverse-engineering procedure based on the one in [34] in order to find the exact xc-potential, with memory effects included. We will then try to reconstruct the exact xc-potential as a functional of all previous densities, using machine-learning.

1.5 Many-Body Green's Functions

The basic definition of the time-ordered Green's function for a system of electrons on a lattice is

$$iG_{ab}(t,t') = \langle \Psi_0 | \, T\left[c_a(t)c_b^\dagger(t')\right] | \Psi_0 \rangle \,, \tag{1.16}$$

where $|\Psi_0\rangle$ is the ground-state, and T is the time-ordering meta-operator. In principle we should also include indices for spin, but since we only consider spin compensated systems ($N_\uparrow = N_\downarrow$) we know that the Green's function must be diagonal in terms of spin. As a result of this we will leave the spin index implicit from now on. We define the time-dependent operators as,

$$c_a(t) = e^{i\hat{H}t} c_a e^{-i\hat{H}t}. \tag{1.17}$$

If $t < t'$ we can interpret this as the probability amplitude of an electron created on site a at time t to be later found on site b at time t'. Alternatively, if $t > t'$ we can interpret this as the amplitude of a hole created from removing an electron on site b at time t' to be found on site a at time t.

The GF contains a lot of information about the system and can be used calculate its total energy, as well as the expectation value of any single-particle operator. It is useful to find the Fourier transform of the GF. To begin with we explicitly write out the time-ordering using theta functions, and insert a complete set of eigenstates of the Hamiltonian,

$$\begin{aligned} iG_{ab}(t,t') = &\sum_n \theta(t-t') \, \langle \Psi_0 | \, c_a \, | n \rangle \, \langle n | \, c_b^\dagger \, | \Psi_0 \rangle \, e^{-i(E_n - E_0)(t-t')} \\ &- \sum_n \theta(t'-t) \, \langle \Psi_0 | \, c_b^\dagger \, | n \rangle \, \langle n | \, c_a \, | \Psi_0 \rangle \, e^{i(E_n - E_0)(t-t')}. \end{aligned} \tag{1.18}$$

Here E_n is the energy of the state $|n\rangle$, and E_0 is the energy of the N-body ground-state $|\Psi_0\rangle$. The time-dependence is now limited to the step functions and the exponentials. From,

$$\theta(\tau) = -\int_\infty^\infty \frac{d\omega}{2\pi i} \frac{e^{-i\omega(\tau)}}{\omega - i\delta}, \tag{1.19}$$

we get,

$$\int_{-\infty}^{\infty} d\tau e^{i\omega\tau} \theta(\tau) e^{-i(E_n - E_0)\tau} = \frac{i}{\omega - E_n + E_0 + i\delta}, \tag{1.20}$$

by use of the residue theorem. Similarly,

$$-\int_{-\infty}^{\infty} d\tau e^{i\omega\tau} \theta(-\tau) e^{i(E_n - E_0)\tau} = \frac{i}{\omega + E_n - E_0 - i\delta}. \tag{1.21}$$

Combining these we obtain the Källén-Lehmann representation,

$$G_{ab}(\omega) = \sum_n \frac{\langle\Psi_0(N)|\, c_b^\dagger \,|\Psi_n(N-1)\rangle \,\langle\Psi_n(N-1)|\, c_a \,|\Psi_0(N)\rangle}{\omega + E_n(N-1) - E_0(N) - i\delta} + \sum_n \frac{\langle\Psi_0(N)|\, c_a \,|\Psi_n(N+1)\rangle \,\langle\Psi_n(N+1)|\, c_b^\dagger \,|\Psi_0(N)\rangle}{\omega - E_n(N+1) + E_0(N) + i\delta}. \quad (1.22)$$

This representation of the time-ordered Green's function contains information about the spectrum of the system, in the form of the poles of the Green's function. Specifically, the poles are located at the excitation energies of the many-particle system.

In addition to the interacting many-body GF it is also useful to define the non-interacting GF. The definition is identical to the interacting case, but instead of using the interacting ground-state $|\Psi_0\rangle$ we use the non-interacting ground-state $|\Phi_0\rangle$. This non-interacting ground-state will be a Slater determinant of the single-particle orbitals $|\phi_n\rangle$, and we can again obtain a spectral representation

$$g_{ab}(\omega) = \sum_n^{\text{unocc.}} \frac{\phi_a^n \phi_b^{n*}}{\omega - \epsilon_n + i\delta} + \sum_n^{\text{occ.}} \frac{\phi_a^n \phi_b^{n*}}{\omega - \epsilon_n - i\delta}. \quad (1.23)$$

Here $\phi_a^n = \langle\emptyset|\, c_a^\dagger \,|\phi_n\rangle$, and the two sums run over unoccupied and occupied orbitals respectively. As in the interacting case the poles are located at the excitation energies, in this case corresponding to the orbital energies ϵ_n.

In addition to the time-ordered GF defined in Eq. (1.16) there exists other types of GFs. The ones we will mention here are the retarded and the advanced GFs, defined as

$$iG_{ab}^{\text{Ret.}}(t,t') = \langle\Psi_0|\left\{c_a(t)c_b^\dagger(t')\right\}|\Psi_0\rangle\,\theta(t-t'), \quad (1.24)$$

$$iG_{ab}^{\text{Adv.}}(t,t') = \langle\Psi_0| - \left\{c_a(t)c_b^\dagger(t')\right\}|\Psi_0\rangle\,\theta(t'-t). \quad (1.25)$$

We can find the Fourier transform of these functions in the same way as for the time-ordered version. The end result will be the same as (Eq. 1.22), apart from the sign of the $i\delta$ terms. For the retarded GF all the terms will have $+i\delta$ in the denominator, while for the advanced GF we have $-i\delta$ instead. In other words the retarded GF will have all its poles in the lower half of the complex plane, and the advanced GF will have all its poles in the upper half.

The interacting and non-interacting Green's functions are connected via the Dyson equation

$$G(\omega) = g(\omega) + g(\omega)\Sigma(\omega)G(\omega) \quad (1.26)$$

where $\Sigma(\omega)$ is the so called self-energy [35]. We note that the second term involves a matrix multiplication in terms of the site and spin indices, and that since both G and g are diagonal in spin Σ must be spin diagonal as well. Multiplying this equation by the matrix inverses G^{-1} and g^{-1} we end up with an expression of the self-energy

$$\Sigma(\omega) = g^{-1}(\omega) - G^{-1}(\omega). \tag{1.27}$$

We can split the self-energy into two parts, the first of which is simply the Hartree potential which is diagonal, frequency independent, and real. The second part of the self-energy is more complicated, and is the part that must typically be approximated. Since we can determine Σ numerically exactly in small systems, by virtue of its meromorphic structure Σ can be written in terms of poles and residues. It is then possible to extract the poles and residues, for instance by fitting the imaginary part of $\Sigma(\omega)$ with Lorentzians. We go into details about this process in section 2.3

1.6 Machine Learning

Machine-learning can be broadly grouped into supervised and unsupervised learning. In supervised learning the data provided to the machine-learning program consists of pairs of inputs and outputs, and the goal is that the program will learn to predict the correct outputs for inputs it has not seen before. We can think of this as showing the program a number of examples of correct solutions to some problem, and the program then learning how to correctly solve the problem in other cases. In unsupervised learning on the other hand the program is just shown input data, and the program is typically expected to find patterns independently. For instance one can use unsupervised learning to find clusters in data, or to create generative models that can create new data similar to what it has seen before. In this thesis we are only concerned with the former of these, namely supervised learning.

1.6.1 The Multilayer Perceptron

One of the main machine-learning methods used today is that of Artificial Neural Networks (ANN) [36]. We can consider a neural network as a collection of simple computational units, called neurons, each of which performs some simple calculation on its input and then outputs the result. These neurons are linked together such that the outputs of some neurons are used as input to other neurons. Such a neural network is often visualized as a directed graph where each neuron corresponds to a node, and an edge from

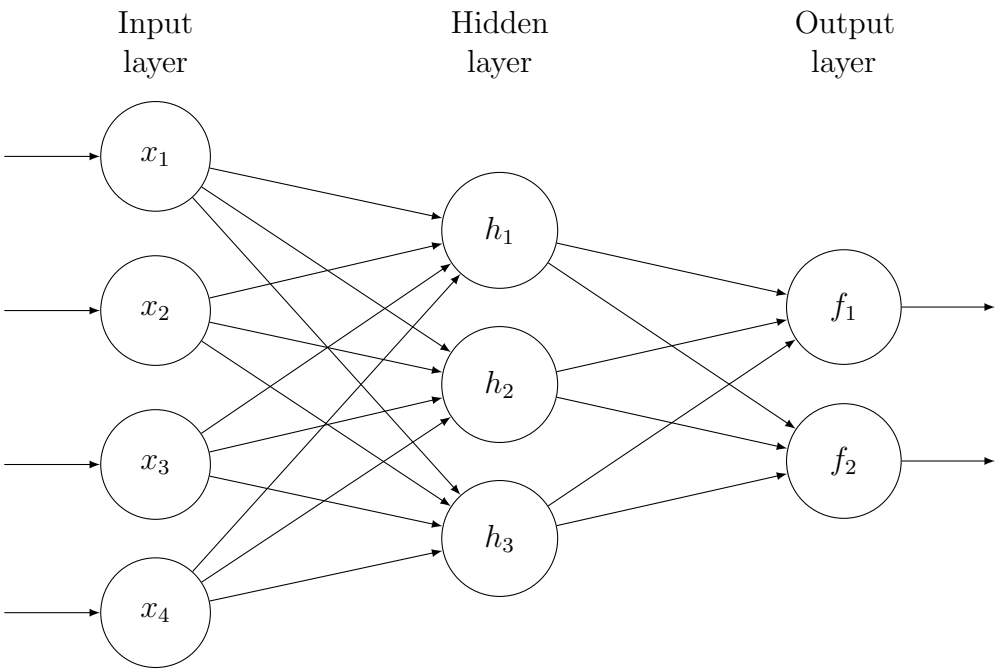

Fig. 1.1: Example of a multilayer perceptron, represented as a directed graph.

one node to another corresponds to the output of the first node being used as input to the second one. The specific calculation of each neuron is typically to perform a weighted sum of its inputs, and then to apply a non-linear function. Specifically the output is given by

$$y = \varphi \left[\sum_i \omega_i x_i + b \right], \tag{1.28}$$

where x_i are the inputs, ω_i are the weights, b is called a bias weight, and the function φ is referred to as the activation function. The weights, bias, and activation can differ from neuron to neuron.

One of the most important types of artificial neural network is the Multilayer Perceptron (MLP). In this type of network we have a network input, and output, and the neurons are organized into layers with the output of each layer being used as input to the next layer, as illustrated in figure 1.1. The first layer of the network is called the input layer and simply outputs the input to the network, while the final layer is called the output layer and gives the output of the network itself. In between these are the hidden layers, so called since their outputs are not directly observed when using the network.

The number of layers in an MLP is called the depth, while the number of neurons in a layer is called the width of that layer.

Overall the MLP implements a function $f(x)$, consisting of several nested non-linear functions. The goal of machine-learning with MLPs is to find the weights and biases such that this function approximates some other function as closely as possible. How exactly this is done in practice will be discussed in more detail in the next subsection. A number of theorems exists which show that MLPs are universal function approximators, i.e. that they can approximate any reasonably well-behaved function arbitrarily well. These theorems, called universal approximation theorems, hold in the limit of infinite network size, either infinite depth or infinite width, and typically put some conditions of the activation functions. While these theorems give some theoretical justification for using MLPs, in practice they do not give any way to actually find the weights such that a network of given size best approximates a given function. It is also not guaranteed that a given function can be well represented by a reasonably sized network, or that we can easily find the weights which makes the network represent said function. In practice, however, none of these considerations seem to be major issues for machine-learning applications. Typically the function we want to represent is only known indirectly via a finite set of samples, and we are merely looking for a good enough approximation. Finding the best possible representation on a finite dataset can also be undesirable due to overfitting, which we will discuss later. In the end, one of the main motivations for using MLPs is the vast number of highly successful applications to a wide variety of problems.

1.6.2 Training MLPs

As we have mentioned, an MLP allows us to quite generally represent functions. However, in order for this to be of actual use we need a way to train the network, that is to find the weights such that it approximates the function we want. In order to do this we need first a dataset, which we shall refer to as the training data, and a loss function that gives us a measure of how how accurate the network predictions are. The training dataset consists of samples of the function we want to implement. In other words the data consists of tuples (x^i, y^i) where x are inputs, y are the targets i.e. the value of the target function for input x, and i indexes the different samples in the dataset. As for the loss function, a simple choice is the Mean Square Error (MSE) given by

$$L(\boldsymbol{\omega}, x, y) = \frac{1}{N_y} \sum_j (f_j(x, \boldsymbol{\omega}) - y_j)^2 \tag{1.29}$$

where f is the function implemented by the MLP, $\boldsymbol{\omega}$ are the weights and biases, and j runs over the N_y components of the output. While there exists a wide variety of other more specialized loss functions, such as the cross-entropy typically used for classification problems, we will however only consider the MSE in this thesis.

The process of training the network consists of minimizing the loss function averaged over the training dataset using some form of gradient descent. While the most straightforward way to do the gradient descent is to use the gradient evaluated on the entire test dataset, so called Batch Learning, this approach is rarely used. In Stochastic Gradient Descent one instead calculates the gradient on a randomly sampled subset, called a minibatch, of the training dataset for each training step. The size of this subset, which can be just a single sample, is called the batch-size. Having a small batch-size means that each step is much easier to calculate, but one must do more training steps overall, since more steps are needed to cover the entire dataset. The number of training steps needed to cover the entire training dataset is called an epoch, and is commonly used as a measure of how long a network has been trained. Usually it turns out that stochastic gradient descent is more effective than batch-learning, but the ideal batch-size must be found by trial and error, and a bad choice of batch-size can impact learning negatively. The most simple approach to such a gradient descent is to simply update the weighs according to

$$\boldsymbol{\omega} \to \boldsymbol{\omega} - \alpha \nabla_{\boldsymbol{\omega}} E(\boldsymbol{\omega}) \tag{1.30}$$

with E being the average of the loss L over the current minibatch, $\nabla_{\boldsymbol{\omega}}$ being the gradient with respect to the weights, and the parameter α being referred to as the learning-rate. Usually α is decreased over time, allowing the model to explore large parts of the weight space before narrowing down to a local minima. The choice of learning-rate turns out to play a huge role in the effectiveness of training an MLP, and so choosing the correct one is an important but often difficult task requiring a lot of experimentation. Many attempts have been made to create new optimization procedures, with the goal of either making learning faster, or to reduce the negative impact of a poorly chosen learning-rate. One such optimizer is ADAM [37], which is the one we will use in this thesis.

As we have mentioned the training of a network requires the calculation of the gradient with respect to each of its weights. A large network can have a great number of weights, and so it may seem that calculating the gradient is a prohibitively difficult task. Fortunately due to the way the MLP is constructed the gradients can be efficiently calculated using the backpropagation algorithm. In the backpropagation algorithm we start at the end

of the network and calculate the gradient with respect to the weights of the final layer. Using the chain rule these gradients can then be used to calculate the gradient with respect to the weights of the second final layer, which can be used to calculate the next layer, and so on.

As we train the MLP on the training dataset the performance of the model as measured on this data will improve. In our case the MSE between the network output and the target will decrease. While this means that the model is learning, in the end what we actually want is for the model to generalize, that is to make good predictions on data it has not seen before. In other words we want it to perform well on data taken from the same source as the training data, but not in the training dataset. Typically the error made on new data, the generalization error, will initially decrease as we train the model. Eventually, however, we reach a point where even though the training error keeps decreasing the generalization error starts to increase. This behavior is referred to as overfitting, and is a general problem in machine-learning. We can understand this as the model learning to recognize the training dataset rather than the underlying function from which the dataset is sampled.

The process of trying to avoid overfitting is called regularization, and a variety of different regularization methods have been proposed. One common way to regularize a neural network is to add an extra term to the cost function which penalizes large weights. This can be thought of as smoothing out the function implemented by the network, and forcing unimportant weights to vanish, thereby reducing the complexity of the model. One such method is $L2$ regularization, also known as weight decay. In $L2$ regularization we add to the loss function a penalty term, proportional to the 2-norm of the weights:

$$L'(\boldsymbol{\omega}, x, y) = L(\boldsymbol{\omega}, x, y) + \lambda \sum_i \omega_i^2, \tag{1.31}$$

where i indexes all the weights of the network. Another common method is dropout, in which the neurons in the network are randomly switched off during the training steps. Dropout can be seen as a way to approximate an ensemble average of all possible models created by removing nodes from the original one, and this in turn has a regularizing effect. Both of these methods can help to reduce the generalization error, but at the cost of making the training more complicated and adding additional parameters which must be fine-tuned. For weight decay we need to choose the strength of the suppression term, and for dropout we must choose the probability of a neuron to be deactivated, both of which must typically be found by trial and error. Perhaps the simplest way to regularize a neural network is to simply set aside some data which we do not train on. This separate dataset is usually called

the validation dataset and is used keep track of the generalization error as training progresses. When the generalization error has not improved for a set number of epochs we can then stop training and revert to the optimal weights. Such an approach is referred to as early stopping, and is the one mainly used during this thesis.

1.6.3 Convolutional Neural Networks

An important variation on the MLP is the Convolutional Neural Network (CNN) [38]. In such a network we have one or more convolutional filters, with the kernels of the filters being trainable parameters. These filters can be applied to the input of the model, or to the output of earlier filters. In the one-dimensional case the output of such a filter is given by

$$h_i = \varphi\left[\sum_n x_{i-n} K_n + b_i\right] \tag{1.32}$$

with x being the input vector, φ being the activation function, b being the bias weight, and K being the kernel. Here the sum runs over the size of the kernel, which is smaller than the size of the input vector. We can compare this to the fully connected layers discussed previously, see Eq. (1.28). It is clear that while the two are quite similar, the convolutional layer can have quite significantly fewer trainable parameters. We can regard the convolutional layer as being a dense layer with a great deal of weight sharing, that is with a lot of nodes constrained to have the same input weights.

The CNN has found much use in image processing problems, for which it is very well suited. One notable property of the CNN is that it incorporates a degree of translational symmetry, as a result of the same kernel being applied to the entire input. This property can be useful when applied to machine-learning in physics, since it allows the network to automatically account for such symmetry in physical systems [5].

1.6.4 Recurrent Neural Networks

The MLPs and CNNs we have discussed so far have had a clear directionality in that each layer feeds into the next one, with no connections going from a layer to a previous one. We now allow for the possibility for nodes to connect to a previous layer, or even back to itself. Such a network with feedback is called a Recurrent Neural Network, and they find a natural application to problems involving series data, to which we will restrict our discussion. The data will now consist of a sequence, with some index t denoting the position

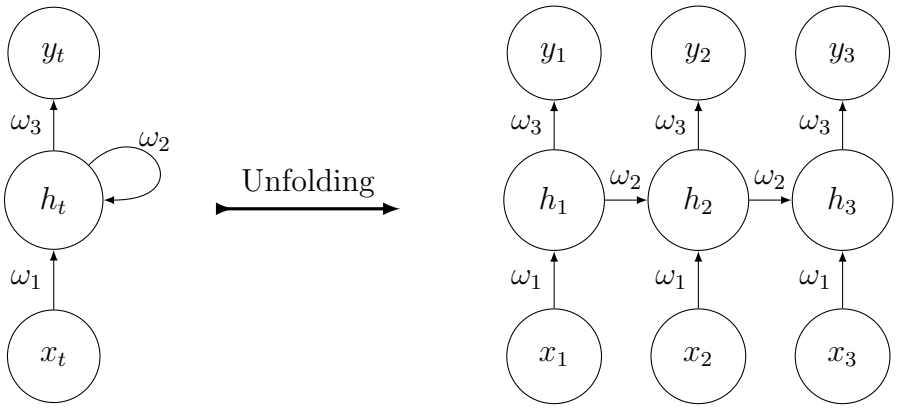

Fig. 1.2: Illustration of how a simple recurrent network can be unfolded into a multilayer perceptron. To the left is the original network, to the right is the network unfolded to the third step.

in the sequence. We will refer to this index as the time index, even though the sequence does not necessarily need to be a time-series. This means that the input to the model will have a time index, as will the output of the model, as well as the outputs of all the hidden nodes. By adding, say, a connection from a node back to itself we can then have the output of the node at t depend on the output at time $t-1$, and similarly for other feedback connections. In this way the network can represent correlations over long distances in t, and be thought to have a memory of previous inputs.

In order to train an RNN on a series of data we regard it as an MLP with a high degree of parameter sharing acting on the entire sequence. To illustrate the process we can imagine a network with a single input x, a single output y, and a single hidden node h which feeds back into itself, as illustrated on the left-hand side of figure 1.2. On the right-hand side of this figure we have unfolded the network into an MLP. We can imagine the process as applying copies of the same network to each input, and then adding in the feedback connections as going between these copies along the time direction. By unfolding the network like this we can calculate the gradients with respect to each input in the series, and then update the weights as for the ordinary MLP.

When training on long sequences there is a problem which can arise, namely that of vanishing gradients. If we let h_t denote the output of some hidden node at time index t with self-feedback, then the gradients used to

train the network will contain factors like

$$\frac{\partial h_i}{\partial h_j} = \frac{\partial h_i}{\partial h_{i-1}} \cdots \frac{\partial h_{j+1}}{\partial h_j}. \tag{1.33}$$

If these derivatives are small, then the gradient will become smaller, and each gradient descent step can become so small as to make training prohibitively slow. One common way to avoid the problem of vanishing gradients is to instead use a so called LSTM, or Long Short-Term Memory, network [39]. In such a network we replace the standard neurons with LSTM nodes, which implement a slightly more complicated operation than the traditional artificial neuron. While we will not go into the details about how the LSTM nodes work, we will mention that they allow for the capture of long-term dependencies in data while keeping the gradients from vanishing during training.

2 Numerical Calculations

As we have mentioned, in order to apply machine-learning to a problem we need data that can be used to train and to test the machine-learning model. In this chapter we will go through how we perform these calculations. To begin with we will go through the way we solve the model Hamiltonians under study, using exact diagonalization. In the second section we go through how to obtain the exact xc potential in ground-state DFT. The third section explains the way we calculate the Green's functions, self-energy, and the poles and residues of the self-energy. In the fourth section we go through the calculations in TDDFT, specifically how we perform the time-evolution and how we find the time-dependent xc potential. In the final section we cover the methodology of our machine-learning calculations.

2.1 Exact Diagonalization

In most of our calculations we need to diagonalize, at least partially, a lattice Hamiltonian. For ground-state DFT we need to find the eigenstate with the smallest energy for the interacting system, and the N_σ lowest orbitals for the non-interacting one. Similarly for the GF calculations we need to fully diagonalize the Hamiltonian so that we can construct the GF using the Källén-Lehmann representation. In order to perform such diagonalization we must first of all express the Hamiltonian as a matrix. To do this we define basis states by applying the creation operators to the empty state. Since the fermionic creation operators anti-commute the ordering of these operators is important in the definition of the basis state. For example the states $c^\dagger_{1,\uparrow}c^\dagger_{2,\uparrow}\left|\emptyset\right\rangle$ and $c^\dagger_{2,\uparrow}c^\dagger_{1,\uparrow}\left|\emptyset\right\rangle$ both represent a state with a spin-up electron at sites 1 and 2, but these two states differ by a sign. As long as we are consistent in the ordering of the fermionic creation and annihilation operators the choice of such a basis does not have any physical meaning.

For the Hubbard dimer we will now write out the basis states we use. In this particular case we have 4 states for $N_\uparrow = N_\downarrow = 1$, and 2 for the $N = 1$

and $N = 3$ cases. As we have mentioned previously the number of basis states grows quickly as we increase the size of the system. For systems larger than the dimer we will be using previously developed code [40] to obtain the basis and the relevant operators. The Hubbard basis states for the half-filled dimer are chosen as

$$|1\rangle = c_{1\uparrow}^\dagger c_{1\downarrow}^\dagger |\emptyset\rangle\,, \qquad |2\rangle = c_{1\uparrow}^\dagger c_{2\downarrow}^\dagger |\emptyset\rangle\,, \tag{2.1}$$

$$|3\rangle = c_{2\uparrow}^\dagger c_{1\downarrow}^\dagger |\emptyset\rangle\,, \qquad |4\rangle = c_{2\uparrow}^\dagger c_{2\downarrow}^\dagger |\emptyset\rangle\,. \tag{2.2}$$

Similarly for the single electron model we have

$$|1\rangle = c_{1\sigma}^\dagger |\emptyset\rangle\,, \qquad |2\rangle = c_{2\sigma}^\dagger |\emptyset\rangle\,, \tag{2.3}$$

and for the $N_\uparrow = 2$, $N_\downarrow = 1$ model

$$|1\rangle = c_{1\uparrow}^\dagger c_{2\uparrow}^\dagger c_{1\downarrow}^\dagger |\emptyset\rangle\,, \qquad |2\rangle = c_{1\uparrow}^\dagger c_{2\uparrow}^\dagger c_{2\downarrow}^\dagger |\emptyset\rangle\,. \tag{2.4}$$

For the HH model we construct the states in the same way as for the Hubbard model, except we also apply the phonon creation operators. Unlike for the electrons, the phonon creation operators commute with each other, and with the electron ones. This means that we get the same state, regardless of the order in which we apply the phonon creation operators. In principle the Hilbert space of the HH model is infinite due to the fact that we can add any number of phonons to one of the Hubbard basis states, and get a valid HH basis state. For practical calculations we therefore need to truncate the Hilbert space by defining a maximum number of phonons M. This cutoff must be chosen large enough that it does not affect calculations, which can be tested by repeating the calculations with cutoff $M + 1$ and checking that the results are converged.

With a set of basis states defined we can then express any operator or state as a matrix by considering its action on these basis states. The eigenvalues of the Hamiltonian can then be found by diagonalizing it, completely or partially, as a matrix. We can also obtain the expectation value of any operator in matrix form by sandwiching it between the row and column vector corresponding to a given state. For the diagonalization of the Hamiltonian, and for the calculation of expectation values in systems larger than two sites, we use the QuTiP package for python [41, 42].

2.2 Ground-State DFT Calculations

The aim of our ground-state DFT calculations is to reverse-engineer the exact xc potential. That is to find the xc potential corresponding to a given density.

In order to do this we need the exact density $\mathbf{n}$. This we can calculate by first finding the ground-state via exact diagonalization, as described in the last section, and then applying the density operator. With the exact density known the goal is to find $\mathbf{v}_{KS}$ such that the non-interacting KS system has the same density. In order to find this potential we use an iterative procedure, starting with an initial guess $\mathbf{v}^0_{KS}$. This potential is then used to construct the non-interacting auxiliary system

$$H^0_{KS} = -J \sum_{\langle ij \rangle, \sigma} \left(c^\dagger_{i,\sigma} c_{j,\sigma} + \text{h.c.} \right) + \sum_{i,\sigma} v^0_{KS,i} \hat{n}_{i,\sigma}. \tag{2.5}$$

which is solved in order to calculate the density $\mathbf{n}^0_{KS} = \mathbf{n}_{KS}(\mathbf{v}^0_{KS})$. Since we know that the KS eigenstate will be a Slater determinant we can solve this as a single-electron system for the N lowest spin-orbitals, and sum their respective densities to get $\mathbf{n}^0_{KS}$. We then update the initial guess according to

$$\mathbf{v}^{k+1}_{KS} = \mathbf{v}^k_{KS} - \eta \left(\mathbf{n} - \mathbf{n}^k_{KS} \right), \tag{2.6}$$

with η being a small positive real number, and $\mathbf{n}$ being the density of the interacting system obtained via exact diagonalization. This is repeated until each $|n_i - n^k_{KS,i}|$ is smaller than some fix tolerance, whereupon $\mathbf{v}^k_{KS}$ is taken to be the real KS potential. Once $\mathbf{v}_{KS}$ has been found we subtract the external and the Hartree potential in order to obtain $\mathbf{v}_{xc}$. A slightly more rigorous iteration scheme is derived in [43], which amounts to having a specific choice of η, and letting it change between iterations. However, for our purposes the scheme mentioned above is sufficient.

In the case of a Hubbard dimer at half-filling we can find the exact ground-state KS potential analytically. We can choose to consider either spin for the auxiliary system, and then obtain the Hamiltonian

$$H^{\text{Dimer}}_{KS} = -\left(c^\dagger_1 c_2 + c^\dagger_2 c_1 \right) + \Delta v_{KS} \hat{n}_1 \tag{2.7}$$

in matrix form. Here we have $\Delta v_{KS} = v_{KS,1} - v_{KS,2}$, and we set $J = 1$ for convenience. Using the basis defined in Eq. (2.3) we obtain

$$H^{\text{Dimer}}_{KS} = \begin{pmatrix} \Delta v_{KS} & -1 \\ -1 & 0 \end{pmatrix}, \tag{2.8}$$

which can be straightforwardly solved for the ground-state

$$\phi_0 = \begin{pmatrix} \frac{1}{2}\left(-\Delta v_{KS} + \sqrt{4 + \Delta v^2_{KS}} \right) \\ 1 \end{pmatrix} \tag{2.9}$$

with energy $\frac{1}{2}\left(\Delta v_{KS} - \sqrt{4 + \Delta v_{KS}^2}\right)$. We define $a = \sqrt{1 + \left(\frac{\Delta v_{KS}}{2}\right)^2} - \frac{\Delta v_{KS}}{2}$, and obtain the density as

$$\frac{n_1}{2} = \frac{a^2}{a^2 + 1}. \tag{2.10}$$

The factor 1/2 on the left-hand side is included so that n_1 is the combined density of both spins. Solving for a gives

$$a = \sqrt{1 + \left(\frac{\Delta v_{KS}}{2}\right)^2} - \frac{\Delta v_{KS}}{2} = \pm\sqrt{\frac{n_1}{2 - n_1}}. \tag{2.11}$$

Adding $\Delta v_{KS}/2$ to both sides, squaring them, and solving for Δv_{KS} then gives us

$$\Delta v_{KS} = \pm\frac{1 - n_1}{\sqrt{n_1(2 - n_1)}}. \tag{2.12}$$

Finally we choose the negative solution to ensure that $n_1 < 1$ corresponds to larger Δv_{KS} than $n_1 > 1$.

Apart from the xc potential we also use the energy functional F in (Eq. 1.10) as an ML target. This is also what is done in [5], the results of which we reproduce in order to test our approach. We also use this energy functional to put some restrictions on the data. Specifically we reject all potentials for which this functional differs from the homogeneous case by more than $0.15J$. This is done so as to prevent large energy fluctuations in the dataset, and is in line with [5]. Putting this constraint on the data made training quite a bit more effective. Furthermore we also apply symmetry operations to the data in order to increase the number of samples we can train on. In this thesis we use both periodic and open boundary conditions. For periodic boundary conditions electrons can hop from the last site to the first one, meaning that the first and the final lattice points are treated as adjacent. In this case we have access to both translation and mirror symmetry. For open boundary conditions we no longer have access to the translational symmetry, but we do still have access to the mirror symmetry

$$v_i \to v_{L-i}, \qquad\qquad v_{xc,i} \to v_{xc,L-i}, \tag{2.13}$$

allowing us to effectively double the number of available samples in our dataset.

2.3 Green's Function Calculations

The Green's function calculations can be divided into two parts. First is the calculation of the Green's functions and the self-energy, then the extraction

of the poles and residues of the self-energy. In these calculations we only consider the half-filled Hubbard dimer. The calculation of the many-body GF consists in solving for the ground-state of the half-filled dimer, as well as fully solving the system with one electron added, and with one electron removed. Since the GF is diagonal in spin variables we can limit ourselves to adding or removing just a spin-up electron. The resulting states $|\Psi_0(N=2)\rangle$, $|\Psi_n(N=1)\rangle$, and $|\Psi_n(N=3)\rangle$, can be inserted into the Källén-Lehmann representation (Eq. 1.22) along with their respective energies in order to calculate $G(\omega)$ for any given frequency. Similarly we can solve the single-particle system for the single-particle orbitals $|\phi_n\rangle$ which we can then insert into (Eq. 1.23) in order to calculate $g(\omega)$ for any given frequency. With $G(\omega)$ and $g(\omega)$ found we can then proceed to find the self-energy $\Sigma(\omega)$. Both $G(\omega)$ and $g(\omega)$ are 2×2 matrices (recall that we only consider the spin-up block) and can be straightforwardly inverted. The inverses can be inserted into the Dyson equation (Eq. 1.27) to obtain $\Sigma(\omega)$. We must repeat this process for each ω, but since the states and energies are ω independent we do not need to recalculate these unless we change the external potential. During all these calculations we use the retarded GF (Eq. 1.24) to avoid issues arising from the time-ordered GF being singular in both halves of the complex plane.

Now that we have a way to calculate the self-energy for any given ω we describe how to find the poles and the residues. We consider the imaginary part of the self-energy, which will be a sum of Lorentzians

$$\mathrm{Im}\Sigma(\omega) = \sum_p \frac{A_p \delta}{(\omega + \epsilon_p)^2}, \tag{2.14}$$

where the sum runs over the poles, located at ϵ_p with residue A_p. Since the Lorentzians will be centered around the poles of the self-energy we can find these poles by locating these peaks. We start by calculating $Im\Sigma(\omega)$ for a set of widely spaced frequencies. This allows us to find the rough location of the poles without having to perform too many calculations. We can then repeat this search on a smaller interval around each of these points in order to improve upon these estimates. In this way we can iteratively find the poles with good accuracy.

After the poles have been found the task remains to obtain the residues. If there were just a single pole we could simply take the height of the Lorentzian multiplied by the convergence factor $\pm\delta$, but in our case the height of each peak also includes the tails of the other peaks. Given an estimate for the residues we can insert them into the spectral representation of the self-energy to get $\Sigma_{rep.}(\omega)$. We can then obtain the exact residues by minimizing the

MSE

$$\frac{1}{N_p}\sum_p \left(\Sigma(\omega_p) - \Sigma_{rep.}(\omega_p)\right)^2, \tag{2.15}$$

with p indexing the N_p poles.

2.4 TDDFT Calculations

In our work on TDDFT we focus on the half-filled Hubbard Dimer, i.e. with one electron of each spin. At time 0 we apply a time-dependent perturbing potential acting on the first site, with this potential being given by some function $v(t)$, and we will also take the two sites to have the same potential apart from this perturbation. The initial state is taken to be the ground-state of the unperturbed Hamiltonian.

The goal of our TDDFT work is to develop a machine-learning functional of the time-dependent xc-potential for the Hubbard dimer, which fully takes into account memory effects and the history of the system. In order to train a neural network for this task we need data in the form of time-series corresponding to the perturbation, the time-dependent density, and the time-dependent xc-potential. The perturbation at all times is given from the outset, but the other two quantities must be calculated. In order to calculate the time-dependent density we must time-evolve the system by application of the time-evolution operator

$$|\Psi(t+\Delta)\rangle = e^{-iH(t+\Delta/2)\Delta}\,|\Psi(t)\rangle\,. \tag{2.16}$$

While this can be calculated by inserting a complete set of eigenstates of the Hamiltonian, we will instead utilize the structure of the Hubbard dimer Hamiltonian to make things simpler. Specifically we will use the split-operator approximation

$$e^{-i(T+V)\Delta} = e^{-iT\Delta/2}e^{-iV\Delta}e^{-iT\Delta/2} + \mathcal{O}(\Delta^3), \tag{2.17}$$

which holds for T and V being operators, e.g. different terms of the Hamiltonian.

Using the same basis as for the interacting dimer in the ground-state we can write the time-dependent Hamiltonian in matrix form as a sum of two terms:

$$T = -J\begin{pmatrix}\sigma_x & \mathbb{1}_2\\ \mathbb{1}_2 & \sigma_x\end{pmatrix} \qquad V = \begin{pmatrix}U+2v(t) & 0 & 0 & 0\\ 0 & v(t) & 0 & 0\\ 0 & 0 & v(t) & 0\\ 0 & 0 & 0 & U\end{pmatrix}, \tag{2.18}$$

with σ_x being the Pauli x matrix, and $\mathbb{1}_n$ denoting the $n \times n$ identity matrix. The non-interacting single-particle Hamiltonian, for either spin, can likewise be written as

$$H = -J\sigma_x + \begin{pmatrix} v_{KS}(t) & 0 \\ 0 & 0 \end{pmatrix}. \tag{2.19}$$

In both cases we can use the property $\sigma_x^2 = \mathbb{1}_2$ to calculate the exponential of the kinetic term via Taylor expansion. Explicitly, for the 4×4 case

$$e^{-iT\Delta} = \mathbb{1}_4 + \frac{\cos(2J\Delta) - 1}{2} \begin{pmatrix} \mathbb{1}_2 & \sigma_x \\ \sigma_x & \mathbb{1}_2 \end{pmatrix} + \frac{i\sin(2J\Delta)}{2} \begin{pmatrix} \sigma_x & \mathbb{1}_2 \\ \mathbb{1}_2 & \sigma_x \end{pmatrix}, \tag{2.20}$$

and for the 2×2 case,

$$e^{-iT\Delta} = \cos(J\Delta)\mathbb{1}_2 + i\sin(J\Delta)\sigma_x. \tag{2.21}$$

We will be using these, along with the exponential of the diagonal parts of the Hamiltonian, in (Eq. 2.17) to do all our time-evolution calculations.

Using the above method we can find the many-body state and the time-dependent density, but for the KS system we can also use this result to obtain the KS potential without having to actually perform the time-evolution of the KS system. The approach we use is a lattice version of the one used in [34]. We know that the single-particle state can be written as

$$\phi(t) = \begin{pmatrix} \sqrt{n_1(t)/2} \\ \sqrt{n_2(t)/2}e^{i\theta(t)} \end{pmatrix} \tag{2.22}$$

where θ is the relative phase between the two states. This relative phase can be determined by considering the current, which like the density must be the same for both the interacting and the auxiliary system. Applying the current operator

$$\hat{j}_{1,\sigma} = -iJ(c_{1,\sigma}^\dagger c_{2,\sigma} - \text{h.c.}) \tag{2.23}$$

to $\phi(t)$ and solving for $\theta(t)$ we get

$$\theta(t) = \arcsin\left(\frac{j_{1,\sigma}(t)}{2h\sqrt{n_{1,\sigma}(t)n_{2,\sigma}(t)}}\right). \tag{2.24}$$

We can then apply the time-evolution operator as

$$e^{i\xi(t+\Delta)}\phi(t+\Delta) = e^{-iT\Delta/2}e^{-iV(t+\Delta/2)\Delta}e^{-iT\Delta/2}\phi(t), \tag{2.25}$$

where $\xi(t+\Delta)$ is the global phase added during the this time-step. This is a system of two equations, with two unknowns namely $e^{i\xi}$ and $v_{KS}(t+\Delta/2)$, which can be straightforwardly solved for the KS potential.

In order to test the xc potential obtained via machine-learning it is useful to be able to calculate the density from it. One problem that then arises is that in order to time-evolve the KS system we need the KS potential at times $t + \Delta/2$, but the densities are only known at times t. We will therefore use a Predictor-Corrector scheme in order to time-evolve the KS system. Since the ML v_{xc} is known at the intermediate time-steps we will formulate this predictor-corrector scheme only for the Hartree potential. We plan in follow-up work to test the robustness of this prescription, but do note that it reproduces the correct TD density when used with the reverse-engineered v_{xc}. The process is as follows:

1. Approximate the Hamiltonian by evaluating the external potential at time $t + \Delta/2$ and the Hartree potential at the initial time t. If the xc potential is known at time $t + \Delta/2$, as is the case with our ML v_{xc}, we use it at that time, otherwise at time t. Call this Hamiltonian H_P.

2. Using H_P, time-evolve the KS state, obtaining the density n_P.

3. Using this approximate n_p we obtain the approximate Hamiltonian at time $t + \Delta$, which we call H_C. Again, we evaluate as much of v_{KS} as possible at the intermediate time $t + \Delta/2$.

4. Time-evolve the state from time t to $t + \Delta$ using the average of the two Hamiltonians, $\frac{H_P + H_C}{2}$.

Using this scheme we can calculate the density given a TD xc-potential.

During our machine-learning calculations with TDDFT we take the external potentials to be a superposition of two sine functions,

$$v(t) = A\sin(t) + B\sin(2t). \tag{2.26}$$

For the different samples in the dataset we vary the two amplitudes, A and B. Specifically we have three samples for each A and B pair, as we also include samples with each of them switched off. We sample the amplitudes A and B uniformly on the interval $(0.1, 1)$. The goal is for the network to be able to predict the xc-potential given an arbitrary linear combination (Eq. 2.26), with amplitudes in the same range used during training.

In order to assess the quality of the ML xc potential we compare it to two approximations. The first such approximation is the linear-regime approximation. In the limit of small perturbations the response of the system is in the linear regime and, writing schematically $v_{xc} = v_{xc}(v)$, we then expect that $v_{xc}(v + v') \simeq v_{xc}(v) + v_{xc}(v')$. With the amplitudes we use, the perturbations are not small enough to be in the linear regime, but we can still compare the

ML v_{xc} to the sum of v_{xc} for the two individual sine functions. Another way we will test the ML v_{xc} is by comparing it to an adiabatic approximation. Here we approximate the time-dependent v_{xc} at time t by the ground-state v_{xc} corresponding to the density at time t. For this ground-state v_{xc} we can use the exact one, calculated as in section 2.2, making this is a purely adiabatic approximation which incorporates all ground-state effects but ignores the history-dependence.

2.5 Machine-Learning

Apart from generating data, much of the calculations performed during this thesis consisted of training different neural networks on such data. We will now go through the methodology of these ML calculations. The neural networks and the training procedures we use are all implemented using the Keras package in python [44].

Ground-State DFT - For ground-state DFT we use two different networks, namely with convolutional layers, and without. The convolutional networks were used for the longer chains of 8 sites. For this model we already have an architecture in [5], but hyperparameters such as the learning-rate and the batch-size are unknown. For smaller chains we do not use convolutional layers since the shorter input vector means we would only apply the convolutional filter once or twice. In the case of open boundary conditions we also no longer have translational symmetry, which was the main reason for using convolutional filters on the input. For these models we instead use fully dense networks. The input to the models consists of the ground-state density n. In order to make training easier we apply input normalization. This is done by calculating for each input n_i the average μ_i and standard deviation σ_i over the training dataset, and then applying the following transformation to all input

$$\tilde{n}_i = \frac{n_i - \mu_i}{\sigma_i}. \tag{2.27}$$

As targets we use either the density functional F, consisting of a single value for each input vector, or the vector containing the xc-potential , $\mathbf{v}_{xc}$. During the model search we train a variety of networks with different depth, and different number of nodes, using different values for the batch-size and learning-rate. For the training itself we use the ADAM optimizer, and the MSE as loss function. In all these networks we as activation function the Rectified Linear Unit (RELU), defined as

$$\varphi(x) = \max(0, x), \tag{2.28}$$

with the exception of the output activation which is taken to be linear. Each of these networks is trained with the same seed, and using early stopping. After we have trained a variety of such networks we select the best performing one and train a few networks with this set of hyperparameters, but without a predefined seed, and with a higher patience. Out of these we select the best performing model and proceed to evaluate it on the testing dataset. As such we are using one-fold cross validation for model selection.

Self-Energy - Machine-learning for the self-energy was performed in much the same way as for the ground-state DFT functionals. The input for the self-energy functionals is the external potential, as a two-component vector $\mathbf{v}$. Like for the ground-state DFT we normalize the input such that the mean and standard deviation calculated over the training dataset are 0 and 1 respectively. The targets now consists of a vector containing the poles and the residues of the self-energy. Each of the residues have three independent components, and there are two poles, giving a total of 8 outputs. The model search is conducted in the same way as for the ground-state DFT case. We use dense networks, and we also include convolutional layers as hidden layers, with the number of filters and their width being varied alongside the other hyperparameters. Again the activation function we use is RELU, with linear activation for the output, and we use the ADAM optimizer with the MSE as loss function.

TDDFT - For the machine-learning of the time-dependent xc-potential we use LSTM networks. The datasets consist of multiple time-series organized into pairs $(f_i, v_{xc,i})$, with f_i being the time-dependent external potential and $v_{xc,i}$ being the corresponding xc-potential. In order to use this data for LSTM network we need to organize it into a sequence of overlapping time windows. We then have the input to the network at time-step i consisting of $(f_{i-k}, \dots, f_i, \dots f_{i+k})$, with k being a new hyperparameter. The corresponding output will then be $(v_{xc,i-k}, \dots, v_{xc,i}, \dots, v_{xc,i+k})$. For the next time-step we have inputs $(f_{i-k+1}, \dots, f_{i+k+1})$, likewise for the output, and so on. In order to convert the output from such overlapping time windows back to a single time series we take the output at time-step i to be the average of $v_{xc,i}$ over the different windows. We take the activation function to be RELU, and the output activation function to be linear. The optimization is performed using the ADAM algorithm, with the Root Mean Square Error (RMSE) being used as loss function. To avoid overfitting we use $L2$ regularization.

Apart from LSTM networks we also use dense networks, similar to the ones used in [7]. There the goal is, as in our case, to recreate the xc-potential of TDDFT, however they are working with a different physical system and in continuum TDDFT. Nonetheless the same network architecture can be straightforwardly applied to the system we study. Their network is a dense

neural network without any feedback connections, taking as input the instantaneous density. In our case, however, we take the input to be the external potential f. Since it does not know anything other than the instantaneous input the resulting functional is not a history-dependent one.

The network used consists of two fully connected layers of 1200 nodes each, with the hidden activation function taken to be RELU, and the output activation taken to be linear. For regularization we use L2 weight decay, and as loss function we use the RMSE. As previously we train the networks using the ADAM optimizer.

3 Results

In this chapter we shall go through the results of this thesis. We will divide this into three parts. The first part covers ground-state DFT for the Hubbard and HH models. We discuss the neural networks we have used, and evaluate their performance. Throughout this chapter we will be setting the hopping parameter $J = 1$, and using this as our energy unit.

In the second part we cover the self-energy calculations for the half-filled Hubbard dimer. We present a machine-learning functional for the poles and the residues. This functional is then discussed and evaluated by how well it represents the exact self-energy.

Finally, in the third part, we cover the results for TDDFT. The goal is to be able to predict the time-dependent xc-potential for a time-dependent perturbation consisting of an arbitrary linear combination of two sine functions, with predefined frequencies. We present networks using two different architectures, and discuss their performance on the given task.

3.1 Ground-State DFT

8 site Hubbard model - The first model we will present for the ground-state DFT is an 8 site Hubbard model at quarter-filling ($N_\uparrow = N_\downarrow = 2$) with periodic boundary conditions. We have hopping parameter $J = 1$, and interaction $U = 4$, and have 52500 samples for training, along with 26250 samples each for validation and testing. Furthermore, we apply both mirror and translation symmetry to increase the number of samples further. This Hubbard model is the same as in [5]. We only consider as target the energy functional $F[n]$, defined in Eq. (1.10). In figure 3.1 we plot the error of the machine-learning functional against its output in the form of a scatter plot. Each point in this plot corresponds to one sample $(\mathbf{n}, F[\mathbf{n}])$ in the test dataset, with the horizontal position being $F^{ML}[\mathbf{n}]$, and the vertical position being the difference $F^{ML}[\mathbf{n}] - F[\mathbf{n}]$, i.e. the error made on this sample. The neural network is structured like in [5], consisting of eight convolutional filters of

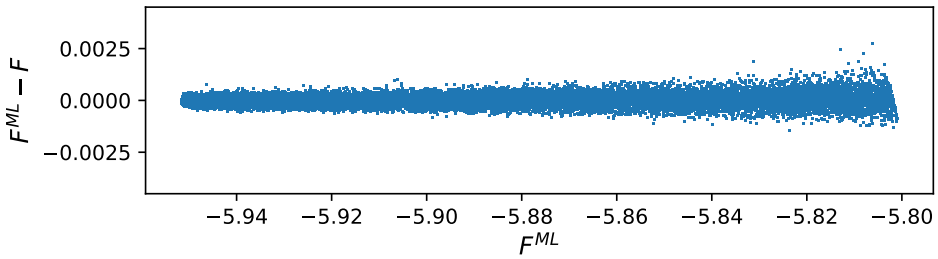

Fig. 3.1: Accuracy of the ML energy functional $F^{ML}[n]$ for an 8 site Hubbard model with periodic boundary at quarter-filling, $N_\uparrow = N_\downarrow = 2$. The horizontal axis shows the output of the model, with the vertical axis showing the error. Each point in the scatter plot corresponds to one input in the testing dataset.The RMSE is 2.63×10^{-4}.

width 3, and two fully connected layers with 128 nodes each. For the network we chose in the end we used a learning-rate of 1×10^{-4}, batch-size of 50, and early stopping with a patience of 250. The Root Mean Square Error (RMSE) evaluated on the test data is 2.63×10^{-4}, comparable to the error obtained in [5] (note that there the performance is measured in mean absolute error, for which we have 1.92×10^{-4}). We note that the error increases, on average, for larger outputs.

4 site Hubbard model - We now move on from the 8 site model, to a 4 site one, again at half-filling ($N_\uparrow = N_\downarrow = 2$), but with open boundary conditions. The reason for this change in system size is so that we can compare with the results for the HH model below, for which the addition of phonon degrees of freedom makes the calculations more difficult. Since we are interested in the xc-potential we also need to perform the reverse-engineering procedure, a further justification for studying a smaller system so as to lower the computational cost. As in the 8 site model we use hopping parameter $J = 1$, and interaction strength $U = 4$. For this model we have a machine-learning representation of the energy functional F, as well as one for the xc-potential. In both cases we use the same dataset, in other words the same densities are used as input, only the targets are different. We use 10000 samples for training, 5000 for validation, and 4354 samples for testing, all effectively doubled by the application of mirror symmetry. The network for the F^{ML} functional had four layers of 256 nodes each, and was trained with a learning-rate of 5×10^{-5}, a batch-size of 100, and patience of 300. We plot the error of this ML functional in the top of figure 3.2 in the same way as for the 8 site functional. When evaluated on the test dataset it gave an RMSE of 1.68×10^{-4}. Although this is smaller than for the 8 site model in

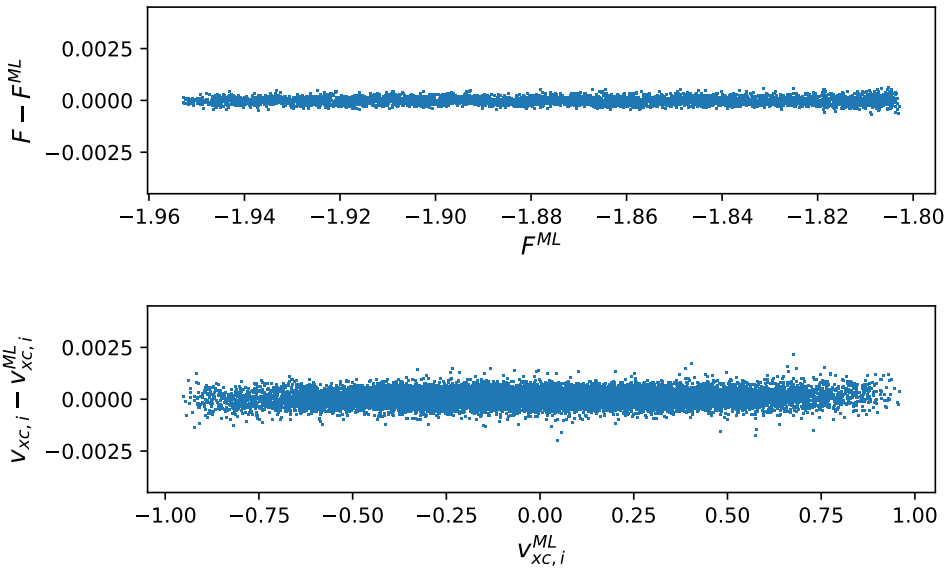

Fig. 3.2: Accuracy of the ML functionals for a 4 site Hubbard chain with open boundary at half-filling. Axes are as in figure 3.1. The top plot shows the error in the $F^{ML}[n]$ functional, with an RMSE of 1.68×10^{-4}. The bottom plot shows the error of the $\mathbf{v}_{xc}^{ML}[n]$ functional, with an RMSE of 3.22×10^{-4}.

absolute terms, we note that the output is itself smaller in magnitude, and so in relative terms the error is larger for the current model.We must however also consider the fact that we have used significantly fewer training samples, and that we still reach a quite impressive accuracy.

The network used in the end for the $\mathbf{v}_{xc}^{ML}$ functional consisted of three layers with 512 nodes each. We trained it with the same learning-rate and patience as for the F^{ML} functional, but with a different batch-size of 50. The error in the output of this network is plotted in the bottom of figure 3.2.For this network the RMSE, evaluated on the test dataset, was 3.22×10^{-4}. We see that the v_{xc} functional performs at the same order of magnitude as the F one, but the relative error is larger, especially near the homogeneous case $v_{xc} = 0$.

4 site Hubbard-Holstein model - Finally we turn to the half-filled 4 site HH model, based on the 4 site Hubbard model previously discussed. The boundary-conditions are again open, with a single phonon mode attached to each site. We use hopping parameter $J = 1$, electron-electron interaction $U = 4$, electron-phonon coupling $g = 0.1$, and phonon frequence $\omega = 1$ for all sites. The phonon Hilbert space is truncated at 10 phonons, for which the calculations were seen to be converged. Our dataset consists of 10000 samples

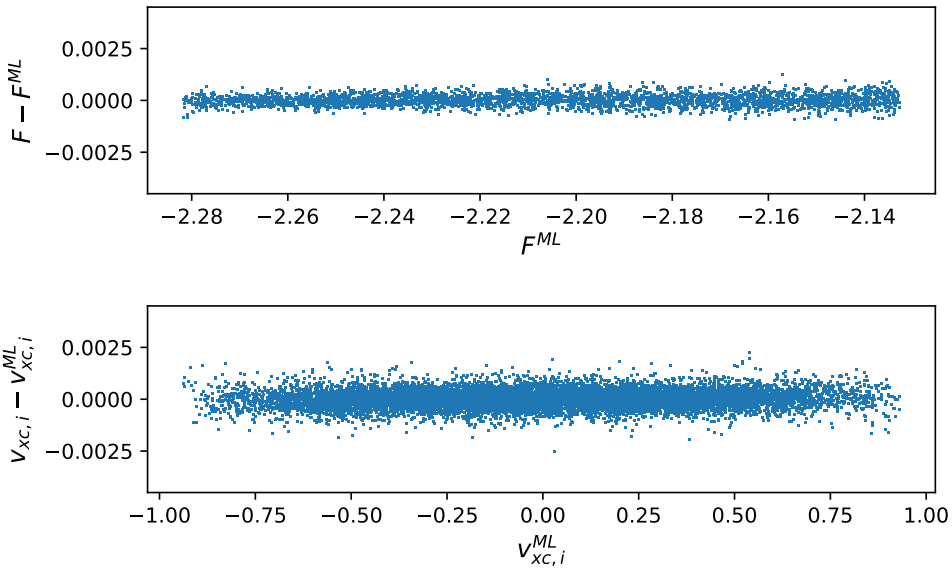

Fig. 3.3: Accuracy of the ML functional for a 4 site HH chain with open boundary at half-filling. Axes are as in figure 3.1. The top plot shows the error in the $F^{ML}[n]$ functional, with an RMSE of 2.71×10^{-4}. The bottom plot shows the error of the $\mathbf{v}_{xc}^{ML}[n]$ functional, with an RMSE of 4.03×10^{-4}.

for training, 5000 samples for validation, and 3163 samples for testing. As for the 4 site Hubbard model we want both the energy functional F^{ML},and the xc-potential v_{xc}^{ML}. For both of these we ended up using three layers with 512 nodes each, same as for the xc-potential of the Hubbard model. The network used for F^{ML} was trained with learning-rate 1×10^{-4}, batch-size 50, and patience 300. For the v_{xc}^{ML} functional we used a learning-rate of 1×10^{-5}, a batch-size of 100, and patience 300. Like for the Hubbard model we plot the error in the output of these two networks, calculated on the test data, in figure 3.3.Evaluation of the F^{ML} network on the test dataset gave an RMSE of 2.71×10^{-4}. The v_{xc}^{ML} network on the other hand gave an RMSE of 4.03×10^{-4}. Although these two networks performed slightly worse compared to the corresponding ones for the Hubbard model, their performance is still of a comparable level. We conclude that these ML functionals do indeed manage to represent the exact functionals also for the HH model, in the range of parameters considered.

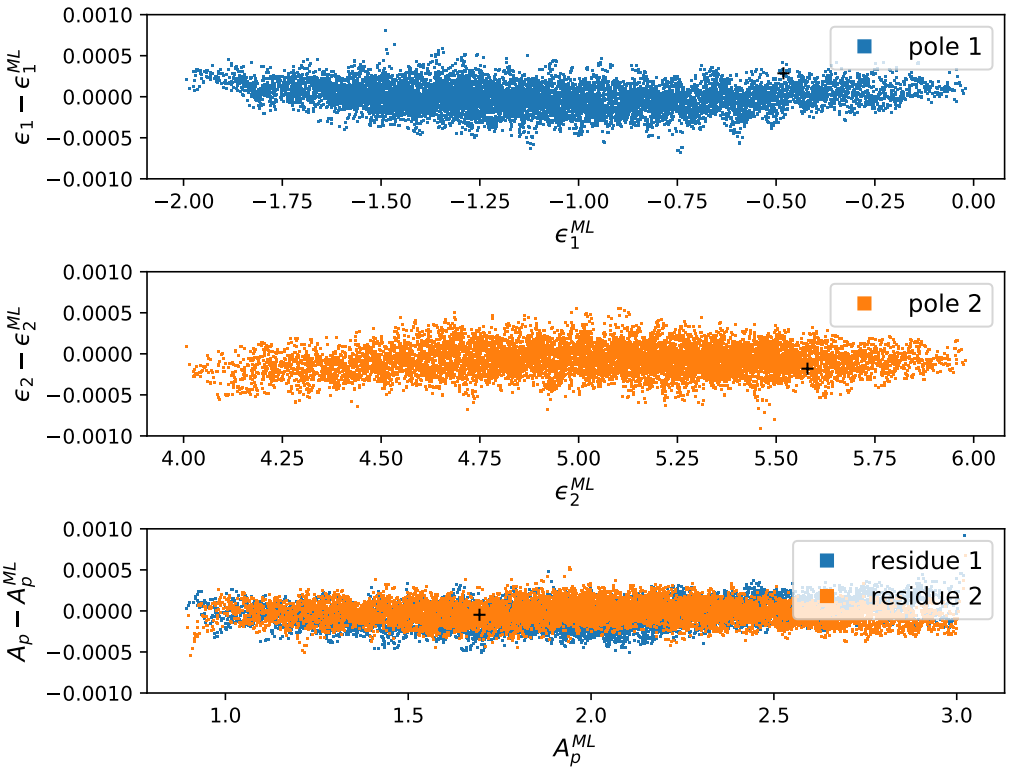

Fig. 3.4: Accuracy of the predictions for the poles and residues of the self-energy of a Hubbard dimer, evaluated on the test set. The top two plots show the error in the prediction for the poles, plotted against the prediction itself. The lowest plot shows the error in the prediction for the residues of the 11*-component of the self-energy, again plotted against the prediction. We use a color scheme where blue denotes the first pole, orange the second pole, likewise for their respective residues. The RMSE is* 1.47×10^{-4}*. We have marked with a black + a single sample, which we plot in figure 3.6.*

3.2 Many-Body Self-Energies

We will now present our results for the many-body self-energy of the Hubbard dimer at half-filling ($N_\uparrow = N_\downarrow = 1$). For this model we use hopping parameter $J = 1$, and interaction strength $U = 4$, same as for the ground-state DFT calculations. The on-site energies v_i, used as ML input, are randomly picked following a uniform distribution on the range $(-1, 1)$. Our dataset consists of 30000 samples used for training, and 10000 samples each for validation and testing.

The network we used in the end consists of a fully connected layer of width 512. We follow this by a convolutional layer with three filters of kernel-size

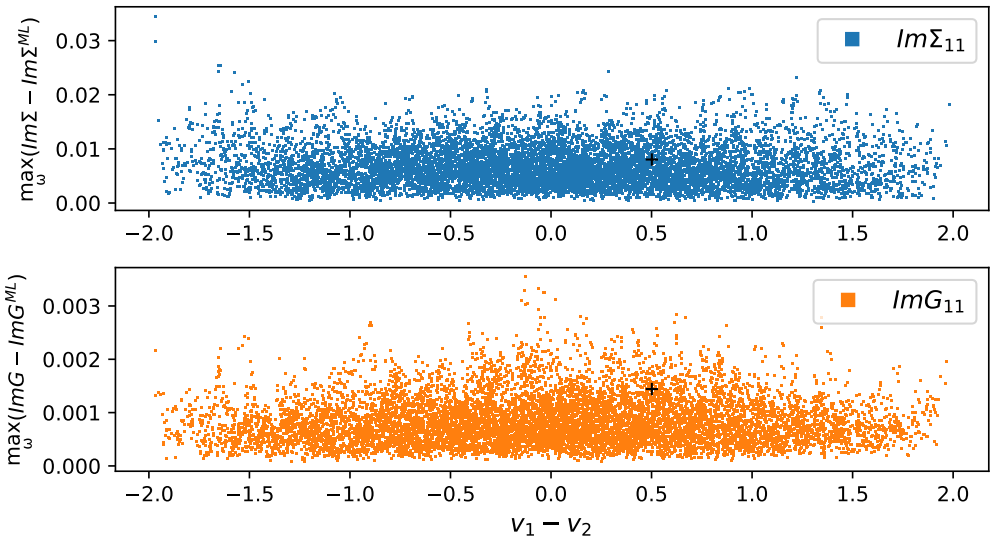

Fig. 3.5: Evaluation of the ML self-energy and GF of a Hubbard dimer, obtained using our neural network. Each colored point corresponds to one sample in the testing dataset, with the horizontal axis giving the potential difference between the two sites, and the vertical axis giving the maximum absolute error evaluated over a range of frequencies $-20 < \omega < 20$. The upper plot (blue) shows the error for the imaginary part of the 11*-component of the self-energy. The lower plot (orange) shows the error for the imaginary part of the* 11*-component of the GF. We have marked with a black + a single sample, which we plot in figure 3.6.*

3, applied to the output of the initial dense layer. The convolutional filters are then followed by a width 64 fully connected layer, followed by output. For the hidden layers we use RELU as activation function, with the output activation being linear. We the network with ADAM and early stopping, using a batch-size of 25, a learning-rate of 5×10^{-5}, and a patience of 200. Figure 3.4 shows the error of the machine-learning functional as evaluated on the test dataset. In the first of these plots we have the error in the prediction of the first pole of the self-energy, plotted against the predicted value, while the second plot shows the error in the prediction of the second pole. The third plot we show the error made in the prediction of the residues, with the same color scheme as their respective poles. In this figure, we have only included the errors in the residue of Σ_{11}, but the error in the other components behaved similarly.The RMSE, evaluated on the test dataset, is 1.47×10^{-4}.

While we have a good prediction of the poles and the residues of the self-energy, these are of interest primarily as a means of representing the

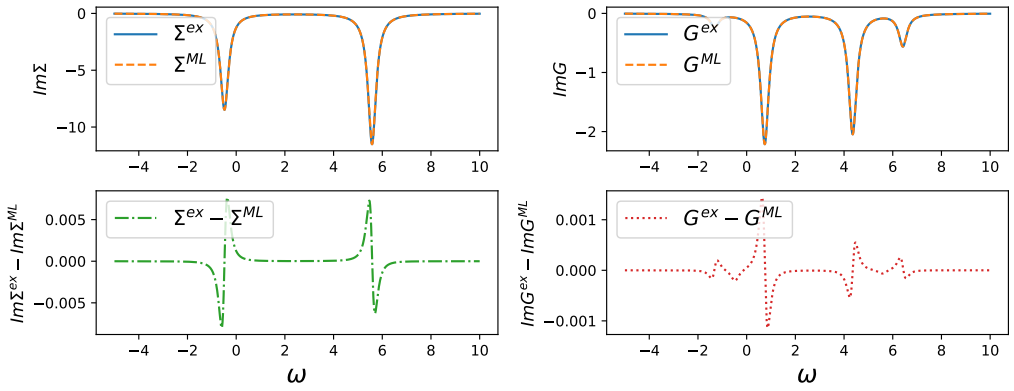

Fig. 3.6: The exact and ML self-energy and GF for a single test sample, marked in figures 3.4 and 3.5. The external potential for this sample is $(v_1, v_2) = (0.7996, 0.2981)$. *Upper plots show the* 11*-component of the exact and ML* Σ *(left plot), and the corresponding GFs (right plot). The bottom two plots show the difference between the exact function and the ML one.*

self-energy itself. In order to test how good this ML representation of the self-energy is we reconstructed the self-energy, using the output of the neural network, over an interval of frequencies $-20 < \omega < 20$, and compare to the exact self-energy. We then calculate the maximum absolute error on this interval. The top plot of figure 3.5 shows this error in the 11-component of the self-energy, plotted against the external potential difference between the two sites. We have chosen the maximum error as a measure of the accuracy since the error will vanish away from the poles. Thus the mean absolute error, RMSE, or the mean square error, would overestimate the performance; by increasing the range of frequencies considered, each of these error measures can be made smaller. This is seen in the left-hand side of figure 3.6, where we have plotted the imaginary part of the exact and ML Σ_{11} for a single sample in the test dataset (marked with a black + in figures 3.4 and 3.5), along with the difference between these. For this test sample the external potential was $(0.7996, 0.2981)$. The error is indeed largest near the poles. In order to further evaluate our network output we use the ML self-energy to calculate the interacting many-body GF. We compare to the exact GF in the same way as we did for the self-energy, and plot the maximum absolute error of the imaginary part of the 11-component against the external potential difference in the bottom plot of figure 3.5. The error is in general smaller than for Σ, which is again seen directly in the right-hand side of figure 3.6 where we have plotted the imaginary part of the exact and ML GF (11-component), along with the error, for the same test sample as earlier. Again the error

Model	v_{xc}^{ML}	n^{ML}	n^{linear}	$n^{adiabatic}$
LSTM, 10000 steps	0.104	0.0315	0.0115	0.0252
Dense,10000 steps	0.0395	0.00788	0.0237	0.0247
Dense, 10000 steps, 3out	0.0390	0.00773	0.0237	0.0247
Dense, 30000 steps	0.00386	0.000904	0.0118	0.0169
Dense, 30000 steps, 3out	0.00178	0.000379	0.0118	0.0169

Table 3.1: Errors of the TDDFT functionals. The rows denote different ML functionals First column shows the network, second row the error in v_{xc}. The final three columns show the error in density calculated with the ML functional, linear-regime approximation, and adiabatic approximation, respectively. The networks labeled 3out were trained on a dataset without the two potentials of which the test data is a linear combination, as described in the main text.

is largest near the peaks, vanishing as we go further away from them. We point out that the height of the peaks is larger than for Σ, by roughly the same factor as for the error. It is clear from our results that this network indeed makes for an accurate representation of the self-energy, at least for this simple two-site model.

3.3 TDDFT

We will now present our results for the time-dependent xc potential. The system under study is the half-filled Hubbard dimer with hopping parameter $J = 1$, interaction strength $U = 4$, and potential $v = 0$. We take the system to initially be in the ground-state, and at time $t = 0$ we apply a time-dependent external potential, coupled to the first site. This external potential is taken to be a linear combination of two sine waves of frequency $\omega_1 = 1$ and $\omega_2 = 2$ respectively, as well as these sine waves taken individually.Our data consists of time-series of this TD potential with random amplitudes $0.1 < A < 1$, and the corresponding xc-potential. In all our TD calculations we use a time-step $\Delta = 0.001$ (in units of J^{-1}).

Our first, natural choice is to use LSTM networks since, as explained in section 1.6.4, they are in general able to capture the history-dependence in a problem. The dataset we use consists of 30 samples, each consisting of 10000 time-steps, setting aside the final sample for testing the network. In the end we use a network with two layers of 200 LSTM nodes each, with RELU activation, and linear output activation, using a size 3 input window. We train the network for 2000 epochs using the ADAM optimizer and L2 regularization with both the learning-rate and the weight decay parameter

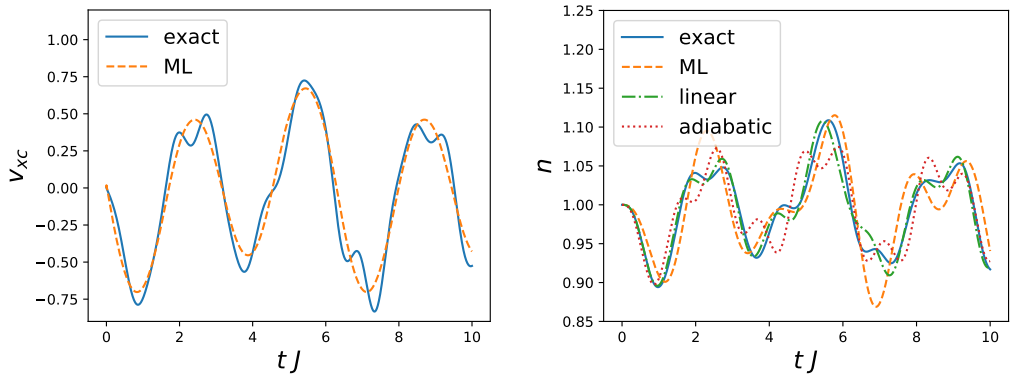

Fig. 3.7: Test of the LSTM model, trained on 29 *samples of length* 10000*. Left plot shows the exact xc potential (solid blue line), and the output of the LSTM network (dashed orange line). Right plot shows the exact density (blue), the density calculated with the ML v_{xc} (dashed orange line), with the linear-regime approximation (dash-dotted green line), and with our adiabatic approximation (dotted red line). Errors are found in table 3.1, top row (LSTM, 10000 steps).*

set to 1×10^{-4}. As loss function we use the RMSE.

The resulting v_{xc}^{ML} for the test sample is shown on the left-hand side of figure 3.7 (dashed orange line), along with the exact v_{xc} (solid blue line). We can see that the exact xc-potential has a structure with double peaks, something which the LSTM network does not capture. None of the LSTM networks we used managed to recreate this structure. In order to further test the v_{xc} from this ML architecture, we use it to calculate the electron density in the KS scheme. The result is shown in the right-hand side of figure 3.7. There we also plot the exact density, the density obtained with a linear-regime approximation, and the density obtained with an adiabatic approximation. We see that the exact density, like the xc-potential, has a double peak structure. This is captured by both the approximate potentials, but not the LSTM one, a clear shortcoming of the latter. We show the RMSE of v_{xc}^{ML}, and the densities, in the first row of table 3.1. From this we see that the RMSE of the v_{xc}^{ML} is of a comparable order to the v_{xc} itself, as seen in figure 3.7. It is also seen that the error in the calculated density is larger than for both of our other approximate potentials.

We will now turn to a different network architecture. Inspired by the good results of [7], we try to apply the same type of network used in that work, namely the MLP. The network in question consists of two fully connected layers, of 1200 nodes each. We use RELU as activation function for the hidden layers, and linear output activation. The model is again trained with

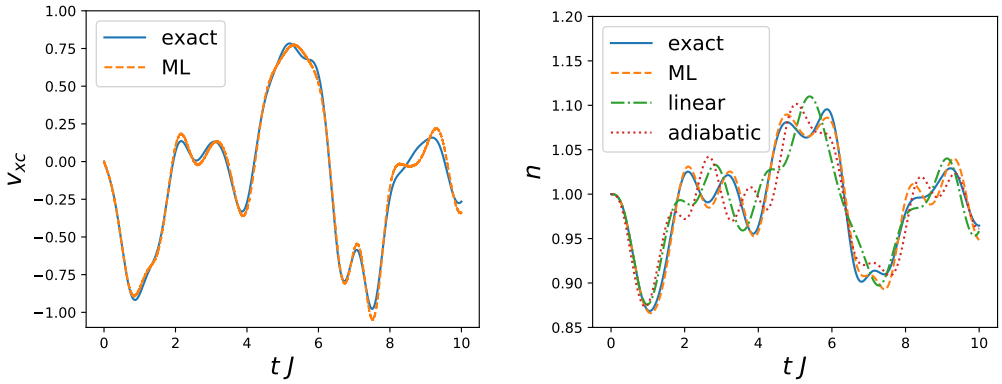

Fig. 3.8: Test of the dense network, trained on 29 *samples of length* 10000*. Left plot shows the exact xc potential (solid blue line), and the output of the dense network (dashed orange line). Right plot shows the exact density (blue), the density calculated with the ML v_{xc} (dashed orange line), with the linear-regime approximation (dash-dotted green line), and with our adiabatic approximation (dotted red line). Errors are found in table 3.1, second row (Dense, 10000 steps).*

ADAM and L2 regularization, using the same hyper-parameters as for the LSTM. Our loss function is RMSE, as this is the one used in [7]. We train the network for 20000 epochs on the same data as used for the LSTM. The output of this network on the training sample is plotted in the left-hand side of figure 3.8, alongside the exact v_{xc}, and we can immediately see an improvement compared to the LSTM. While there are notable differences between the ML and the exact potential, there is a very strong resemblance, and great progress has been made with respect to the LSTM results. The network has indeed managed to recreate, to a good extent, the double peak structure seen in the exact potential. Again we use the output of the network to calculate the density, plotting the result in the right-hand side of figure 3.8, along with the exact density and the approximate ones. The errors for this v_{xc}^{ML} and the densities are tabulated in the second row (Dense, 10000 steps) of table 3.1. Not only is the error in the potential a fair bit smaller, but in terms of the density we have surpassed both approximate potentials.

Encouraged by the success of this approach we generate a larger dataset, consisting of 300 samples, with a length of 30000 each. We train the same dense network as before on this larger dataset, again setting aside one sample for use in testing the network. The output of the fully trained network is plotted on the left-hand side of figure 3.9, along with the exact v_{xc}. We note that the two lines are almost indistinguishable. As before we use the output of the network to calculate the density, plotting it in the right-hand side

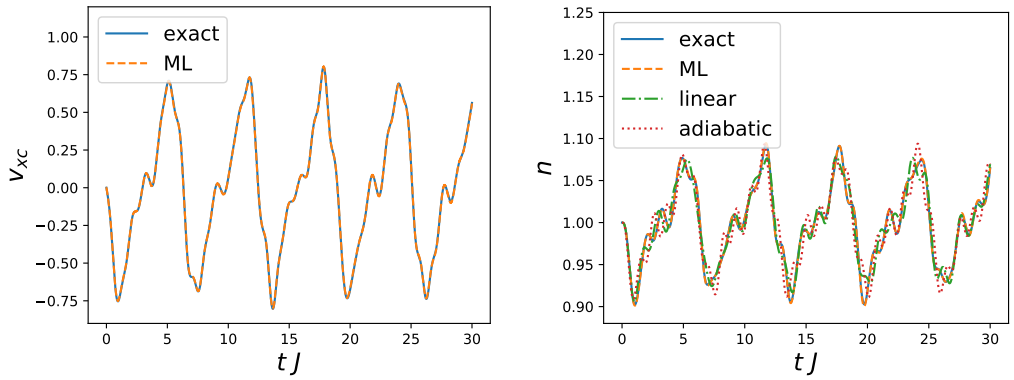

Fig. 3.9: Test of the dense network, trained on 299 *samples of length* 30000*. Left plot shows the exact xc potential (solid blue line), and the output of the dense network (dashed orange line). Right plot shows the exact density (blue), the density calculated with the ML v_{xc} (dashed orange line), with the linear-regime approximation (dash-dotted green line), and with our adiabatic approximation (dotted red line). Errors are found in table 3.1, fourth row (Dense, 30000 steps).*

of figure 3.9, along with the exact density, and densities calculated using a linear-regime and an adiabatic approximation. We calculate the RMSE of the v_{xc}^{ML} and the densities, tabulating them in the fourth row of table 3.1. The error in v_{xc}^{ML} is now a factor 10 better than for the smaller dataset, and the error in the density is likewise significantly better.

Finally we note one limitation in the previous procedure, namely that the TD potential we have tested on is a linear combination of two potentials that are included in the training dataset. Ideally one wants to know if the network will perform well on a completely new linear combination. To test this we train the dense network on the same data as before, but removing the two samples which combine into the test sample. Doing the same calculations as before we calculate the error in the predicted v_{xc}, and the density obtained with this v_{xc}. For the smaller dataset we put these errors in row three (Dense, 10000 steps, 3out) of table 3.1, and for the larger dataset in row five (Dense, 30000 steps, 3out). It is seen that the errors are still of the same magnitude, and might have even improved. The network has indeed managed to calculate the xc potential for a completely new external potential of the predefined form, and has managed to go well beyond the adiabatic approximation.

4 Conclusions and Outlook

In this thesis we have applied machine-learning methods (MLM) to the description of the equilibrium and non-equilibrium behavior of two simple but important lattice models of condensed matter theory, namely the Hubbard model and the Hubbard-Holstein (HH) model. The first model is a minimal template for the description of correlated electrons in narrow bands and ultracold atoms loaded in optical lattices, while the latter takes into account the additional role of localized (optical-like) phonons.

Our description of these models was primarily done within the frameworks of ground-state and time-dependent density functional theory (DFT and TDDFT, respectively), but also considering a description based on the many-body Green's function (GF) technique. More specifically, the aim of our study has been to explore, from a proof-of-concept perspective, ways to train artificial neural networks (ANN) to describe central quantities in the aforementioned theoretical approaches, namely the exchange-correlation (xc) potentials of (TD)DFT and the many-body self-energy of the GF technique.

The use of MLM-ANN in condensed matter research is not new [45, 46, 47, 48]; nevertheless, its application to the theoretical description of lattice models, especially out of equilibrium, remains to this date a largely unexplored area [6].

Density functional theory - Concerning DFT, and building upon existing work (to our knowledge, there are only very few papers in the literature devoted to this topic [5, 3, 49]), we have managed to train a neural network to recreate the exact energy functional $F[n]$ as well as the exact xc-potential $v_{xc}[n]$. We showed that MLM-ANN is applicable with periodic and open boundary conditions for the Hubbard model, as well as in the presence of phonons as in the HH model (it is perhaps worth noting here that an application of MLM-ANN to the HH model is performed for the first time in this thesis work). Unlike [5], we have also constructed an ML functional for the xc-potential using the same approach, achieving a similar good level of accuracy.

In both the energy functional and the xc-potential cases, there is a fun-

damental issue with the implementation of our approach, namely that the constructed functional is applicable only for a given system. In other words, if we wish to change the model parameters, the length of the chain, the number of electrons, or the boundary conditions, we then need to retrain the network on new data. Of these, perhaps the most significant issue is the length, with the difficulty of the exact diagonalization calculations being heavily dependent thereon. An alternative possibility would be to use the density-matrix renormalization group (DMRG) technique [50], but this method is limited to 1D samples and considerably expensive, especially in the MLM-ANN training phase.

A solution suggested in [5] is to use the ML functional to construct a semilocal functional. Indeed, there can be a definite advantage in using semi-local or near-local functionals for lattice models [51]. Following this route, it would be interesting to ascertain if it would be more efficient to implement MLM-ANN for the case of v_{xc}, or for open rather than periodic boundary conditions.

Many-body self-energies - Continuing with the ground-state regime, we have also constructed an ANN representation for the poles and the residues of the many-body self-energy in a Hubbard dimer (two-site system), as functions of the on-site (external) potential. In this way, an ML representation for the exact many-body self-energy becomes available, which is applicable for any given external potential within the range we have trained the network on. By comparison with exact benchmarks, we have verified that this ML self-energy can be used to obtain the correct GF via the Dyson equation. A possible drawback of our method is that as the size of the system grows, so does the number of poles which we must predict. In principle, this problem can be alleviated to some extent by using spatial/energy/time identities/symmetries of the self-energy. While the network should be able to learn such identities from the dataset, changing the training procedure such that the network is forced to satisfy them could potentially allow for better performance, and let us get away with smaller training datasets. At the same time, it cannot be excluded that this could require a slightly more complicated training process. However, as for DFT, in the actual numerics we did not exploit such constraints.

A second, obvious issue when going to larger systems with our pole/residue scheme would be the generation of data, since our naive way of calculating the GF would become prohibitively time-consuming when going to larger systems. However, it is also true that the distinct pole/residue structure that appears in the GF of a small cluster would be largely superseded by a branch-cut-like and continuous density-of-states-like structure: clearly, more

sophisticated methods would then be in order [1].

Another natural extension of the present work is to consider the role of phonons, by applying our MLM-ANN method to the HH model. Within a many-body treatment of the HH model, one obtains two coupled Dyson's equations, one for the electron propagator, and the other for the phonon one. So the procedure would go along the lines discussed above for the purely electronic case, but at the cost of a possibly increased number of poles/residues and computational complexity due to the system of equations. During the thesis, some initial work in this direction was performed but, eventually, due to shortage of time, in the last part of the project our interest turned to the only-electron non-equilibrium case within a TDDFT description.

Regarding TDDFT - Due to the explorative character of this thesis, in our study of MLM-ANN for TDDFT we focused on a very simple, preliminary question: "given a time-dependent external potential which consists of an arbitrary linear combination of two sine functions with predefined frequencies, can one predict via MLM-ANN the corresponding xc-potential in a Hubbard dimer ? "

Our first solution attempt was made using Long Short-Term Memory (LSTM) networks, because of their built-in ability to capture memory effects and correlations over long time-scales. We did not find much success in this way: in fact, the corresponding ML functional was outperformed by two simple approximate functionals, respectively based on linear-response-type arguments, and on an adiabatic scheme. The LSTM has seen much success in other applications involving series data and correlations over long time-scales, and so it is a bit surprising that it fails so badly for the task in question. One possible cause of this could be the windowing procedure used to preprocess the input to the network, specifically the way we convert the output matrix into a time-series. Nevertheless, no matter the reason behind this bad performance, more work on LSTM should be warranted. At the very least, it would be interesting to know exactly why this method failed to give good results.

In addition to LSTMs we have tried to construct the time-dependent xc-potential using a dense neural network like used in [7]. This approach showed a lot more promise, despite the lack of any history-dependence in the resulting functional, managing to perform at a level well beyond the adiabatic approximation.Presumably, more numerical experimentation is needed to ascertain, the role of memory.

Our implementation has only involved two different frequencies. It should

[1]The problem of extended lattice models had already been considered in the literature [2]. Here, we rather focus on small finite systems.

however be possible to include more sine functions. If still good results are achieved while increasing the number of sine functions, then one could make predictions on any external potential which can be represented as a Fourier series. Alternatively, expansions in a different basis (e.g. Gaussian or wavelets) could be attempted.

As a second possibility for future work, we mention the inclusion of memory effects in a similar way to [7], that is to provide the network with an additional input consisting of a weighted average of previous inputs. Furthermore, as we did not explicitly force the network to learn any of the constraints the xc-potential must satisfy, work in this direction could offer another avenue to possibly improve on our results. Yet another obvious but interesting line of development could be to include phonon degrees of freedom, or to move to larger systems, provided that one does not encounter issues with non-interacting v-representability. It is known that such issues can exists for Hubbard clusters other than a dimer [29, 52, 53, 54, 55], and this suggests that another possible line of development would be to move to continuum systems.

To conclude this outlook, and taking some "distance" from our work, it is our impression that all the results we found, both "good" and "bad", point to a rather general message. It is not only important to use a given MLM-ANN within the stipulated domain of its applicability; rather, it is even more important to choose the right MLM-ANN architecture for the problem at hand. This we feel is possibly the greatest challenge to be faced when using MLM-ANN in condensed matter physics research.

Bibliography

[1] F. Aryasetiawan, O. Gunnarsson, *Rep. Prog. Phys.* **61**, 237 (1998)

[2] L.-F. Arsenault, A. Lopez-Bezanilla, O. A. von Lilienfeld, A. J. Millis, *Phys. Rev. B* **90**, 155136 (2014)

[3] C. A. Custódio, É. R. Filletti, V. V. França, *Sci. Rep.* **9**, 1886 (2019)

[4] F. Blockherde, L. Vogt., L. Li, M. E. Tuckerman, K. Burke, K.-R. Müller, *Nat. Commun.* **8**, 872 (2017)

[5] J. Nelson, R. Tiwari, S. Sanvito, *Phys. Rev. B* **99**, 075132 (2019)

[6] M. Schmitt, M. Heyl, *Phys. Rev. Lett.* **125**, 100503 (2020)

[7] Y. Suzuki, R. Nagai, J. Haruyama, *Phys. Rev. A* **101**, 050501 (2020)

[8] J. Hubbard, *Proc. R. Soc. A* **276**, 238 (1963)

[9] E. H. Lieb, F. Y. Wu, *Phys. Rev. Lett.* **20**, 1445 (1968)

[10] J. P. Coe, V. V. França, I. D'amico, *EPL* **93**, 10001 (2011)

[11] G. A. Canella, V. V. França, *Physica* **545**, 123646 (2020)

[12] D. Karlsson, C. Verdozzi, M. M. Odashima, K. Capelle, *EPL* **93**, 23003 (2011)

[13] K. Capelle, V. L. Campo, *Phys. Rep.* **528**, 91 (2013)

[14] T. Holstein, *Ann. Phys.* **8**, 118 (1959)

[15] A. Lanzara, et. al., *Nature* **412**, 6846 (2001)

[16] O. Rösch, J. E. Han, O. Gunnarsson, V. H. Crespi, *phys. stat. sol. (b)* **242**, 1 (2005)

[17] P. Hohenberg, W. Kohn, *Phys. Rev.* **136**, B864 (1964)

[18] W. Kohn, L. J. Sham, *Phys. Rev.* **140**, A1133 (1965)

[19] W. Kohn, *Rev. Mod. Phys.* **71**, 1253 (1999)

[20] N. D. Mermin, *Phys. Rev.* **137**, A1441 (1965)

[21] E. Runge, E. K. U. Gross, *Phys. Rev. Let.* **52**, 991 (1984)

[22] K. Capelle, *Braz. J. Phys.* **36**, 1318 (2006)

[23] J. T. Chayes, L. Chayes, M. B. Ruskai, *J. Stat. Phys.* **38**, 497 (1985)

[24] O. Gunnarsson, K. Schönhammer, *Phys. Rv. Lett.* **56**, 1968 (1986)

[25] A. Schindlmayr, R. W. Godby, *Phys. Rev. B* **51**, 10427 (1995)

[26] R. López-Sandoval, G. M. Pastor, *Phys. Rev. B* **61**, 1764 (2000)

[27] K. Shönhammer, O. Gunnarsson, R. M. Noack, *Phys. Rev. B* **52**, 2504 (1995)

[28] E. Viñas Boström, P. Helmer, P. Werner, and C. Verdozzi, *Phys. Rev. Res.* **1**, 013017 (2019)

[29] C. Verdozzi, *Phys. Rev. Lett.* **101**, 166401 (2008)

[30] N. Dittmann, J. Splettstoesser, N. Helbig, *Phys. Rev. Lett.* **120**, 157701 (2018)

[31] D. J. Carrascal, J. Ferrer, J. C. Smith, K. Burke, *J. Phys.: Condens. Matter* **27** 393001 (2015)

[32] J. I. Fuks, N. T. Maitra, *Phys. Rev. A* **89**, 062502 (2014)

[33] C. Verdozzi, D. Karlsson, M. Puig von Friesen, C.-O. Almbladh, U. von Barth, *Chem. Phys.* **391**, 37 (2011)

[34] M. Lein, S. Kümmel, *Phys. Rev. Lett.* **94**, 143003 (2005)

[35] A. L. Fetter, J. D. Walecka, *Quantum Theory of Many-Particle Physics* (2003), Courier Corporation

[36] C. M. Bishop, *Pattern Recognition and Machine Learning* (Springer, 2006)

[37] D. P. Kingma, J. Ba, *arXiv preprint arXiv:1412.6980*, (2014)

[38] A. Krizhevsky, I. Sutskever, G. E. Hinton, *Commun. ACM* **60**, 84 (2017)

[39] S. Hochreiter, J. Schmidhuber, *Neur. Comp.* **9**, 1735 (1997)

[40] Y. Mahfoud, *Optimizing the functionalities of a code for real time dynamics of finite systems* (LUP Student Papers, 2018), `http://lup.lub.lu.se/student-papers/record/8960218`

[41] J. R. Johansson, P. D. Nation, F. Nori, *Comp. Phys. Comm.* **183**, 1760 (2012)

[42] J. R. Johansson, P. D. Nation, F. Nori, *Comp. Phys. Comm.* **184**, 1234 (2013)

[43] J. P. Coe, I. D'Amico, V. V. França, *EPL* **110**, 63001 (2015)

[44] F. Chollet, et. al., *Keras*, `http://keras.io` (2015)

[45] J. Carrasuilla, R. G. Melko, *Nat. Phys.* **13**, 431 (2017)

[46] E. Bedolla, L. C. Padierna, R. Castañeda-Priego, *J. Phys.: Condens. Matter* **33**, 053001 (2021)

[47] J. Schmidt, M. R. G. Marques, S. Botti, M. A. L. Marques, *npj Comput. Mater.* **5**, 83 (2019)

[48] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, A. Walsh, *Nature* **559**, 547 (2018)

[49] J. R. Moreno, G. Carleo, A. Georges, *Phys. Rev. Lett.* **125**, 076402 (2020)

[50] U. Schollwöck, *Rev. Mod. Phys.* **77**, 259 (2005)

[51] J. Basheer, "Near-local density approximation approach to one-dimensional lattice systems", BsC Thesis, Lund (2018).

[52] M. Farzanehpour, I. V. Tokatly, *Phys. Rev. B* **86**, 125130 (2012)

[53] Y. Li, C. A. Ullrich, *J. Chem. Phys.* **129**, 044105 (2008)

[54] R. Baer, *J. Chem. Phys.* **128**, 044103 (2008)

[55] T. Rössler, C. Verdozzi, C.-O. Almbladh, *Eur. Phys. J. B* **91**, 219 (2018)