UNIVERSIDAD DE GUANAJUATO

MASTER'S THESIS

Using Computational Intelligence to solve the Ornstein-Zernike equation

Author:
Edwin Armando Bedolla Montiel

Supervisor:
Dr. Luis Carlos PADIERNA GARCÍA
Dr. Ramón CASTAÑEDA PRIEGO

A thesis submitted in fulfillment of the requirements for the degree of Master in Applied Science

in the

Science and Engineering Division
University of Guanajuato

October 15, 2021

UNIVERSIDAD DE GUANAJUATO

Abstract

Science and Engineering Division
University of Guanajuato

Master in Applied Science

Using Computational Intelligence to solve the Ornstein-Zernike equation

by Edwin Armando Bedolla Montiel

This thesis is an exploration of the use of *computational intelligence* techniques to study the numerical solution of the Ornstein-Zernike equation for simple liquids. In particular, a continuous model of the hard sphere fluid is studied. There are two main proposals in this thesis. First, the use of *neural networks* as a way to parametrize closure relation when solving the Ornstein-Zernike equation. It is shown that in the case of the hard sphere fluid, the neural network approach seems to reduce to the Hypernetted Chain closure. For the second proposal, we explore the fact that if more physics is incorporated to the closure relation, a better estimate can be obtained with the use of *evoluationary optimization* techniques. When choosing the modified Verlet closure relation, and leaving a couple of free parameters to be adjusted, the results are as good as those obtained from molecular simulations. The thesis is then closed with outlooks on different ways to improve the proposals presented in the current work, as well as new ways to solve the problem using other Machine Learning techniques.

Acknowledgements

Pendiente por redactar ...

Contents

Chapter 1

Introduction

Since the mainstream adoption of *Machine Learning* (ML) methods on common tasks such as object recognition, computer vision, and human-computer interactions [LBH15], scientists have tried to adopt most of these techniques to further research in their respective fields. From drug development [RKD20] to genetics and biotechnology [LN15], multiple applications of ML to current research problems have seen widespread interest for their generalization and automatic discovery attributes. It is with the inspiration from these applications that physicists have attempted to use such methods in diverse Physics fields [Car+19; DB18; CM17].

Most of the attempts and successes of using ML methods in Physical sciences come from the direct application of common ML pipelines and uses, such as *classification*, *regression*, and *unsupervised learning* [HTF09], just to name some. Such is the case of the determination of the critical point of the Ising model as means of a classification task [CM17]. Similar is the case of the use of *computer vision* and *deep learning* techniques in particle physics, which have seen great applications when dealing with experimental data [Rad+18]. In each of the previous examples, scientists have taken the most common applications of ML methods and have adjusted them for their respective research problems. This has the advantage that such ML techniques have been extensively researched and developed, so physicists know that these methods are robust and useful for the problems they have been developed for. However, it turns out that not all ML techniques can be readily applied to the problem at hand, and physicists should instead try to capitalize on the Physics of the problem and use it along with the ML method to boost its usefulness, flexibility and accuracy.

It is with this perspective that physicists have preferred to incorporate most of the Physics into the ML method, and thus create a new form of *physics-inspired machine learning* [Kar+21]. One such example is the Behler-Parrinello neural network approach for energy surfaces in the Density Functional Theory framework [BP07]. Within such proposal, Behler and Parrinello chose to use some functions whose definition and composition are based on the properties of the system studied, which were atoms and their components, and use such information as input to a *regression* scheme to approximate the energy surface of the studied system. At the moment of publication, this approach defined a new paradigm of ML application within the physical sciences. It was no longer the fact that simple learning tasks were used, but by including physical descriptors in the ML methods, new ways of obtaining the same results were found. Not only do ML methods provide mostly the same results as the physical framework they are modeling, they also provide solutions much faster and more efficiently [Zhu+19].

Of all the research fields within Physics, in this thesis we would like to focus our attention to the field of *condensed matter physics*, and in particular, to the field of Liquid State Theory and Soft Matter. Before we do that, however, we should mention briefly some of the precursors to the applications of ML to said fields. A great review by Jörg Behler [Beh16] describes in great detail

some of these precursors. Most applications have dealt with materials science, computational chemistry and chemistry, and the computational aspect of condensed matter physics. It is within these fields that new ways of using ML methods have been developed from various needs in research. Another prime example is the coupling of computer simulations and ML methods, such as the work by Li *et al* [LKDV15]. In this work, whenever the quantum-mechanical information is needed it is computed with first-principles calculations. These calculations are then added to a dataset to be used by ML methods, which in turn are used as approximators for the computer simulation. This approach not only efficiently uses Physics in its most pure form, but it also adopts the ML best attributes and uses them to its advantage. For a more complete overview of some of the most impactful applications, the review by Bedolla *et al* [BPC20] covers these and some other important aspects of ML methods applied to condensed matter physics.

Although these are a tiny subset of all the modified applications of ML techniques in physical sciences research, we should note that these show an important aspect in common between them. If we wish to assimilate the physics of the problem at hand, variations and modifications to the common ML methods and techniques are needed. Exploration and testing, trial and error, are an important part of the search and application of ML methods to Physics research. In the case of Liquid State Theory and Soft Matter, we can consider that most research is still in the exploration and testing stage, although some applications have seen great success in specific scenarios. One such successful application is the use of *Support Vector Machines*—an explicit method useful for classication and regression based on kernels and quadratic optimization [SC08]—in the description of the properties of glassy dynamics [Sch+16]. The reason for research in Soft Matter and ML still being part of the exploration step is that in Soft Matter and Liquid Theory it is hard to find suitable descriptors that actually tell useful information from the system or phenomena [DL21]. As such, most of the time a descriptor-based approach might not be feasible for every possible system. Instead, we need to explore a diverse range of possibilities when using ML methods within the context of Soft Matter and Liquid Theory.

Even though most research is still exploration and testing, there have been interesting amalgamations and developments in the fields of Soft Matter, with Liquid Theory dragging behind. In a sense, this is expected, due to the fact that Liquid Theory might be thought of as *solved*, although there is still research done within the field. Let us focus first on the developments of Soft Matter. Instead of referring to specific uses of ML within Soft Matter, it is more fruitful to mention some of the research groups that have delved deep into using ML methods in Soft Matter. The group of Marjolein Dijkstra at Utrecht University is an excellent example. Having done impactful research in the field of Colloidal Soft Matter [DvE99; Leu+05], the group is now focusing on using all the research and knowledge built and trying to understand the best way to enforce the physics of the systems into ML techniques. The group has explored with *evolutionary algorithms* and their uses in patchy colloids [Bia+12]. We refer to evolutionary algorithms as derivative-free optimization algorithms that are useful for nonlinear optimization problems [YG10]. Unsupervised methods, such as *principal component analysis* [HTF09], have been used for choosing the best descriptors in supercooled liquids [BSF21], as well as the detection of local structure in colloidal systems [BDF19]. All in all, these methods simplify the process of dealing with these research problems. ML methods make it simpler and easier to identify structure and attributes from a system. However, it is important to note that not only do these methods make it simpler, they can also contribute to finding new things that were previously not as obvious or easy to see. Another work from the group is the use of classification methods to identify different types of crystal phases using a mix of supervised and unsupervised methods [HTF09], such as in the work by van Damme *et al* [van+20]. This work is quite interesting because it is a great example of using

physical descriptors, such as bond order parameters [SNR83; LD08], along with ML methods that actively select and distinguish between the best descriptors for the system.

Another important group that has done several advances in the use of ML within Soft Matter is the one lead by Thomas Truskett from the University of Texas. Their work on the use of ML methods for inverse desing of soft materials [She+20] is in similar ways helping out the work by the group of Dijkstra in the same research problem [Aps]. However, the work by the Truskett group is in fact more focused on the definition and foundations of better descriptors for soft materials and off-lattice systems [JLT18]. An important topic that the group explores is the inverse design of self-assembly systems. *Self-assembly* is the property of soft materials to order their components, such as particles, atoms or cells, without any external interactions, into functional structures [Grz+09]. In this direction, their interest is particularly focused in self-assembly and how this phenomena can be dealt with ML methods. In a particular work by Lindquist *et al* with the Truskett group [LJT16], optimization-based methods were used along with standard molecular dynamics computer simulation methods to understand how certain materials can self-assemble. Instead of using optimization-based methods, another work from the Truskett group deals with probabilistic ML methods for some of the same phenomena [JLT17]. The difference between both these uses is that probabilistic methods are better at dealing with probability distributions that stem from the Statistical Mechanics description of the problem. With the use of specialized frameworks, handling these descriptors are easier and more flexible.

Up until this point, we have talked about Soft Matter and its interaction with machine learning methods, and most of all the way it has been disrupting this research field with novel ways to do things. Liquid State Theory was briefly mentioned before, but it is time to state the relation between both these research fields. Why talk about Liquid State Theory and Soft Matter as if these were related somehow? In a fascinating note by Evans, Frenkel and Dijkstra [EFD19] they talk about the precise relation between liquids, their theory, and how by studying such simple, but also complicated systems, soft matter research arises quite naturally. Liquids are non-trivial systems, that are also dense, disordered and spatially correlated but exhibit peculiar characteristics that make them different from crystals, solids, and gases. The first attempt to understand and apply liquid theory, along with computer simulations of liquids, was to study the classic Lennard-Jones system to model Argon [MS67; Ver67]. Computer simulation methods, theoretical frameworks, and experimental setups helped understand the true nature of liquids, given them a special place in Physics research. Physicists were attempting to construct a rigorous framework which will later be used to further create new research fields. Such fields were then created by the outstanding work of Nobel Laureate Pierre Gilles-de Gennes. In his Nobel Lecture [de 92] he mentioned that by studying simple systems it was possible to extend what was known of such systems to new, different systems that exhibit similar properties. His work on liquid crystals and polymer physics opened up a whole new research field, which we now call Soft Matter. Then again, behind all the bewildering phenomena found in Soft Matter, and all the innovative research done to understand it, there exists a solid theory of the most simple systems known as liquids. And yet, these systems are not *simple* at all, they have evolved into active research fields such as ionic liquids [RS03], complex fluids [GB96], and many others.

Despite all the research done in Liquid State Theory, most research has been focused on understanding it through computational and theoretical frameworks. There is not much interaction between Liquid State Theory and ML research at this point. Yet, there seems to be an area of opportunity where Liquid State Theory can greatly benefit from all the current research in ML methods. Arguably, the first work to delve deep into the intersection between Liquid State Theory

and ML theory was the work done by Goodall and Lee. In this work, a dataset is built from Molecular Dynamics computer simulations using several interaction potentials. Then, a ML method was used to predict properties from a new, unknown system, using most of the information available from the dataset. The method worked great, but the precision was not the best in some cases. Granted, this is just the first attempt to build a solution to the problem of closure relations for the theoretical framework of the Ornstein-Zernike equation [HM13]. It was then extended, in the same work, to solve an inverse design problem of liquids, which indeed solved the problem but was lacking in accuracy, and was not tested in more challenging problems, which would be interesting to see the generalization of such methods. After all, the attempt is a novel way to look at things, and most of all, a new way to look at Liquid State Theory and its intersection with ML theory. Exploring this area is the sole purpose of this thesis, and in particular, in the same line of understanding how can ML methods be used in theoretical frameworks that model simple liquids. It seems like a worthwile task to test various approaches and look for a way to streamline the methods used so far in Liquid State Theory. This is the basis for this work, to explore the use of ML methods in the theoretical formalism of the Ornstein-Zernike equation, a rigorous framework that provides all the information from a simple liquid system. We wish to understand if some of the most common ML methods can be used, how can they be used, if they can even provide physically-relevant solutions or not.

To achieve this goal, this thesis is organized as follows. In chapter 2, the next chapter, the theory of simple liquids will be outlined, focusing heavily on the Ornstein-Zernike formalism, as well as the theory of integral equations. A description of the reference system—the hard sphere fluid—will be discussed. The thermodynamical properties of simple liquids will also be described in full. Computer simulation methods will also be discussed as they are the standard tool for current research when a baseline result is needed. Computer simulation methods can provide similar results to those obtained from experimental data, which is why they are used as a reference. Then, in chapter 3, an overview of ML theory is presented. In particular, those methods used in the current thesis, namely neural networks and optimization methods. These methods have their mathematical frameworks, which will be outlined as well, without going into much mathematical rigor. The most common learning tasks such as supervised and unsupervised learning will be touched only slightly, in a manner that is consistent with the current presentation of the topics. After the theorical background, a discussion is carried out on the first proposal of using ML methods in Liquid State Theory in chapter 4. The proposal is to use a neural network as a parametrization for the closure relation within the Ornstein-Zernike formalism. It is shown that the proposal works, the method proposed to make it work, a training scheme, and the results obtained. It is found that the proposal works, but if the neural network is not given too much information or description about the system, the neural network training dynamics will reduce to a common closure relation. In chapter 5, an exploration of including physical information to the problem gives a more direct solution. Instead of using a neural network to parametrize the closure relation, a known approximation is used, and let an evoluationary algorithm find the best parameters for it. When using an approximation that is already shown to give accurate results, the ML method is better suited to work correctly, but not without its drawbacks, which are discussed in full. The thesis the closes in chapter 6, with important conclusions and future ideas to amend the proposals presented here. A few other proposal are outlined, although not with results, or rigorous theoretical frameworks. These can be thought of suggestions that might somehow become new ways to use ML methods within Liquid State Theory.

Chapter 2

Liquid State Theory

In this chapter, a brief description of Liquid State Theory is carried out. In particular, the focus of the chapter is to state what a liquid is, its thermodynamical properties and how equilibrium statistical mechanics is used to understand them. Then, a description of the hard-sphere fluid is mentioned, which is the fundamental system studied in this work. After this, a brief description of computer simulation methods is outlined, both Molecular Dynamics and Monte Carlo methods are mentioned, giving more importance to Monte Carlo methods which are extensively used in this work. The chapter is closed with a presentation of the important formalism that is the Ornstein-Zernike integral equation. Its important aspects are mentioned, along with a discussion of sveeral closure relations and the important role of the *bridge function*.

2.1 Equilibrium Statistical Mechanics

Consider a system of N spherical particles in three dimensions where each particle is characterized by its position $\boldsymbol{r}$ and momentum $\boldsymbol{p}$. The *Hamiltonian* of the system is given by

$$\mathscr{H}\left(\boldsymbol{r},\boldsymbol{p}\right) = K\left(\boldsymbol{p}\right) + U\left(\boldsymbol{r}\right) \tag{2.1}$$

with K the kinetic energy and U the potential energy of the system. All together, the $6N$ variables define a *phase point* in a $6N$-dimensional *phase space*. The state point of the system is then described by a *phase space vector* $\Gamma\left(\boldsymbol{r},\boldsymbol{p}\right)$, however, considering that all N particles move according to Newton's equations of motion, Γ is a function of time, or $\Gamma(t)$. Using this phase space vector, *time averages* can be obtained for a given observable A by means of the following expression

$$\langle A\rangle = \lim_{t\to\infty}\frac{1}{t}\int_0^t A\,\Gamma(t')\,dt'\,. \tag{2.2}$$

If instead the complete set of state points, also known as the *ensemble* of state points, is considered, then the average $\langle A\rangle$ can be rewritten in terms of this ensemble. This ensemble of state points is distributed in phase space according to a probability distribution that is specified by the *thermodynamic ensemble*. Then, if the time evolution of $\Gamma(t)$ is such that all states are visited eventually, irrespective of its initial conditions, the system satisfies the weak ergodic theorem [Kit04], and the time average in Equation 2.2 can then be rewritten with an *ensamble average* that reads

$$\langle A\rangle = \sum_{\Gamma} A(\Gamma)\,\rho_{ens}(\Gamma)\,, \tag{2.3}$$

where the sum is for all state point vectors Γ and $\rho_{ens}(\Gamma)$ is the probability density function for the ensemble. This probability function is a weight function for the averaging procedure and

should be normalized,

$$\sum_{\Gamma} \rho_{ens}(\Gamma) = 1\,. \tag{2.4}$$

With this information, it is now time to introduce the *canonical ensemble*, which is the main ensemble used throughout this work.

2.1.1 Canonical ensemble

The canonical ensemble is established as a system of N particles in a fixed volume V at temperature T that can exchange energy with a heat bath. This ensemble has a probability density function associated with it,

$$\rho_{NVT} = \frac{e^{-\beta\mathscr{H}(\Gamma)}}{\sum_{\Gamma} e^{-\beta\mathscr{H}(\Gamma)}}\,, \tag{2.5}$$

where $\beta = 1/k_B T$, and with k_B being Boltzmann's constant. In the classical limit of continuous distribution functions the denominator from Equation 2.5 transforms into

$$Q(N,V,T) = \frac{1}{N!\,h^{3N}} \int d\boldsymbol{p}^N\, d\boldsymbol{r}^N\, e^{-\beta\mathscr{H}(\boldsymbol{r}^N,\boldsymbol{p}^N)}\,, \tag{2.6}$$

which is known as the *canonical partition function* [Hua87]. Here, h is an arbitrary but predetermined constant with the units of energy × time. As a side note, in the original formulation by Gibbs, the value of h was set to $h = 1$ [Gib14], however, since the advent of quantum mechanics, it is now taken to be Planck's constant [Tol79] in order to show a correspondence between the classical and quantum formulations. Indeed, the integrals from Equation 2.6 can be separated, and the integral over the momentum coordinates can be carried out analytically giving,

$$Q(N,V,T) = \frac{1}{N!\,\Lambda^{3N}} \int d\boldsymbol{r}^N\, e^{-\beta U(\boldsymbol{r}^N)}\,, \tag{2.7}$$

with $\Lambda = \sqrt{h^2/2\pi m k_B T}$ the thermal wavelength, also known as the *de Broglie* wavelength. This again shows a correspondence between classical and quantum formulations. The remaining integral over the positions is called the *configuration integral*,

$$Z(N,V,T) = \int d\boldsymbol{r}^N\, e^{-\beta U(\boldsymbol{r}^N)}\,. \tag{2.8}$$

Finally, we arrive at the canonical ensemble density function in the continuum limit which is

$$\rho_{NVT} = \frac{e^{-\beta U(\boldsymbol{r}^N)}}{Z(N,V,T)}\,. \tag{2.9}$$

Using this probability density function, the ensemble average for an observable A is computed as

$$\langle A \rangle = \frac{\int e^{-\beta U(\boldsymbol{r}^N)}\, A(\boldsymbol{r}^N)\, d\boldsymbol{r}^N}{Z(N,V,T)}\,. \tag{2.10}$$

2.2 Distribution functions

Still, the probability density function in Equation 2.9 provides far more information from the system than necessary for the calculation of thermodynamical functions and structure properties.

Instead, a focus on a small set of particles $n \ll N$ is preferred, in which case a *reduced probability density* is defined as

$$\rho_N^{(n)}(\boldsymbol{r}^n) := \frac{N!}{(N-n)!}\frac{1}{Z(N,V,T)}\int e^{-\beta U(\boldsymbol{r}^N)}\, d\boldsymbol{r}^{(N-n)}\,, \tag{2.11}$$

where $\rho_N^{(n)}(\boldsymbol{r}^n)$ is also known as the equilibrium *n-particle density*. The quantity $\rho_N^{(n)}(\boldsymbol{r}^n)\, d\boldsymbol{r}^n$ defines the probability of finding n particles of the system with coordinates in a volume element $d\boldsymbol{r}^n$ from the phase space, regardless of the positions and momenta of the remaining particles. As a result of this contraction of the probability density function, it is now possible to provide a complete description of the *structure* of a fluid, while the knowledge of low-order particle distribution functions is sufficient to calculate thermodynamic quantities [McQ00].

From the definition of the n-particle density in Equation 2.11, the normalization condition is

$$\int \rho_N^{(n)}(\boldsymbol{r}^n)\, d\boldsymbol{r}^n = \frac{N!}{(N-n)!}\,, \tag{2.12}$$

and in particular, the *single-particle* density is

$$\int \rho_N^{(1)}(\boldsymbol{r})\, d\boldsymbol{r} = N\,, \tag{2.13}$$

For a homogeneous fluid, Equation 2.13 is then simplified to the following expression

$$\rho_N^{(1)} = N/V \equiv \rho \tag{2.14}$$

where ρ is the *particle number density*.

The n-particle distribution function $g_N^{(n)}(\boldsymbol{r}^{(n)})$ is defined in terms of the particle densities by

$$g_N^{(n)}(\boldsymbol{r}^{(n)}) := \frac{\rho_N^{(n)}(\boldsymbol{r}_1,\ldots,\boldsymbol{r}_n)}{\prod_{i=1}^n \rho_N^{(1)}(\boldsymbol{r}_i)}\,, \tag{2.15}$$

which again for a homogeneous fluid it reduces to

$$\rho^n g_N^{(n)}(\boldsymbol{r}^{(n)}) = \rho_N^{(n)}(\boldsymbol{r}^n)\,. \tag{2.16}$$

Particle distribution functions measure the extent to which the structure of a fluid deviates from complete randomness [HM13]. If the system is also isotropic, which it shall be the consideration henceforth, the pair distribution function $g_N^{(2)}(\boldsymbol{r}_1,\boldsymbol{r}_2)$ is a function only of the separation $r = r_{12} = |\boldsymbol{r}_2 - \boldsymbol{r}_1|$ between particles in positions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$; here $|\cdot|$ is the Euclidean distance or norm between two n-dimensional vectors. The function $g_N^{(2)}(\boldsymbol{r}_1,\boldsymbol{r}_2)$ is the *radial distribution function*, and it is referred to as $g(r)$ for the rest of this work. This radial distribution function is of crucial importance in Liquid State Theory for several reasons. First, the radial distribution function can be obtained experimentally for liquids using X-ray diffraction, digital videomicroscopy, and light scattering experiments [McQ00]. Second, thermodynamic properties of liquids can be determined using integrals that contain the radial distribution function. Third, the radial distribution function can be easily computed in computer simulations [AT17], which is the standard way to obtain these quantities. Finally, the radial distribution function can be obtained analytically using integral equations such as the Ornstein-Zernike equation, which shall be discussed in a later section.

2.3 Thermodynamic properties

As mentioned in the previous section, thermodynamic properties can be defined in terms of distribution functions, and in particular, in terms of the radial distribution function [HM13]. It is now time to discuss such expressions, and the relation between thermodynamical quantities and the radial distribution function.

The *total energy* E of a system of N particles can be defined in terms of the radial distribution function as

$$E = E_{ideal} + E_{excess} = \frac{3Nk_BT}{2} + \frac{N}{2}\int_0^\infty \rho\, u(r)\, g(r)\, 4\pi r^2\, dr\,, \tag{2.17}$$

where E_{ideal} is the ideal gas contribution, whose result comes directly from the energy equipartition theorem [McQ00]; E_{excess} is an interaction or excess contribution that can be understood as the interaction energy between a central particle for all the N particles, and all the neighbors located in a spherical shell of radius r and thickness dr. The total number of neighbors is given by $4\pi r^2 \rho g(r)\, dr$. The integration from 0 to ∞ gives all the interaction energy, and the factor of $1/2$ accounts for double counting of particle pairs. The function $u(r)$ is the *pairwise interaction potential* which comes from the fact that the total potential energy $U(\boldsymbol{r}^N)$ can be expressed in terms of the individual particle interactions as follows

$$U(\boldsymbol{r}^N) = \sum_i u_1(\boldsymbol{r}_i^N) + \sum_i \sum_{j>i} u_2(\boldsymbol{r}_i^N, \boldsymbol{r}_j^N) + \dots \tag{2.18}$$

This kind of interactions are the most commonly researched and studied, given that most systems can be modeled after such interaction potentials, such as the hard-sphere, Lennard-Jones, Yukawa [Hua87] and several others. The hard-sphere interaction potential will be discussed in a later section. In the current work, the two-particle interaction potential $u_2(\boldsymbol{r}_i^N, \boldsymbol{r}_j^N)$ is referred to simply as $u(r)$. It is important to note that this is not the norm, and interaction potentials that are not pairwise additive are also studied. These many-body potentials are notoriously hard to study, but recent advances in Machine Learning have made it possible to do so [Boa+20].

The *pressure equation* is a relation between the thermodynamic pressure P and the radial distribution function, defined as

$$P = P_{ideal} + P_{excess} = \frac{\rho}{\beta} - \frac{2\pi\beta\rho}{3}\int_0^\infty u'(r)\, g(r)\, r^3\, dr\,, \tag{2.19}$$

where P_{ideal} is the kinetic pressure of an ideal gas, and P_{excess} is the excess pressure that can be derived using classical mechanics through the virial theorem [GPS02].

Another important quantity that must be mentioned is the *isothermal compressibility*. Its thermodynamic definition is

$$\chi_T = -\frac{1}{V}\left(\frac{\partial V}{\partial P}\right)_T = \frac{1}{\rho}\left(\frac{\partial \rho}{\partial P}\right)_T\,, \tag{2.20}$$

and it is a physical measure of the relative change in volume due to a change in pressure or stress. It can also be computed using the radial distribution function using the following expression [HM13]

$$\frac{\chi_T \beta}{\rho} = 1 + \rho \int d\boldsymbol{r}\, \left[g(r) - 1\right]\,. \tag{2.21}$$

We call this the *compressibility equation*, and it is a standard way of computing the isothermal compressibility.

2.4 The hard-sphere model

In order to describe the behavior of materials, pairwise interaction potentials are chosen according to specific physical properties. These interaction potentials provide a particular functional form for the potential energy U of the system. With this information, all the theory described so far can be used to understand the properties of the system, and compute thermodynamic quantities.

Out of all the possible interaction potentials, there is one that stands out for its simplicity and surprising ability to generalize to complex systems. This model is called the *hard sphere model*, and shall be the topic of discussion for this section. The *hard sphere model* is a simple pairwise interaction potential, defined as

$$u_{hs} = \begin{cases} \infty, & r \leq \sigma \\ 0, & r > \sigma, \end{cases} \tag{2.22}$$

where σ is the radius of the particles in the system, and r is the distance between two particles, as previously stated.

This interaction is a particular model; it is an approximation of the intrinsic behavior of particles. It attempts to describe the excluded volume interaction of particles, similar to what happens with billiards balls in three dimensions. These particles seem to hit each other only to be separated at the contact point, which is the value of σ. Hence, this model is by definition a *repulsive model*. Apart from being a reference model system for the liquid state [HM13], hard spheres are extremely useful in colloid science, as mentioned in the introduction section. This is due to the fact that is has been demonstrated that the hard sphere interaction gives rise to a fluid-crystal phase transition around a volume fraction of 50% of hard spheres [HR68; GR98; RLS14]. This phase transition was the subject of an extensive discussion in the early 1950's and was for the first time discovered in computer simulations [AW57]. The relationship between this simple model and colloid science was discovered experimentally by Pusey and van Megen [Pv86], in dense colloidal suspensions of sterically stabilized particles in a solvent.

In this work a similar potential is used, though defined differently. Looking closely at Equation 2.22, it is straightforward to see that the potential is a discontinuous function. In mathematical proofs and statistical mechanics frameworks, this poses no problem. It might be hard to manipulate, mathematically speaking, but it is possible to do so. Yet, the problem lies with the use of computer simulations. The technical issues arise mostly in Molecular and Brownian Dynamics computer simulations [AT17], so a different formulation must be employed. Computer simulations of liquids are presented in the next section. In 2018, Báez *et al* [Báe+18] used the Extended Law of Corresponding States [VLC18] to map the second virial coefficient of the hard sphere model to a continuous model. By doing so, most of the dynamic and thermodynamic properties of the hard sphere model are "passed along" to the continuous function, which in turn is suited for all kinds of computer simulation algorithms. This new continuous function is used throughout this work, and it is defined as follows

$$u_{CP} = \begin{cases} A\epsilon\left[\left(\frac{\sigma}{r}\right)^{\lambda} - \left(\frac{\sigma}{r}\right)^{\lambda-1}\right] + \epsilon, & r < \sigma B, \\ 0, & r \geq \sigma B, \end{cases} \tag{2.23}$$

with

$$A = \lambda\left(\frac{\lambda}{\lambda-1}\right)^{\lambda-1}, \quad B = \left(\frac{\lambda}{\lambda-1}\right). \tag{2.24}$$

The value of λ is fixed to $\lambda = 50$, following the findings in the work of Báez *et al* [Báe+18]; and the value of ϵ is not fixed explicitly, instead the reduced potential is employed, i.e., $u^* = u_{CP}/\epsilon$.

2.5 Computer simulations of liquids

Liquid State Theory is always supplemented with results from computer simulations because these methods come from first-principles Physics and yield accurate estimations of the behavior of the physical systems. Also, given the availability of computational resources nowadays, these methods have become a standard tool for research. There has been an extensive amount of research in these computer simulation methods, following the pioneering work of Alder and Wainwright [AW57]. There are two main computer simulations methods used in Liquid State Theory and Soft Matter research: *Molecular Dynamics* and *Monte Carlo methods* [AT17; FS01]. These methods serve different purposes, and because of that, a brief description of each one will be outlined here. However, the main focus of this work is the use of the Monte Carlo simulation technique. *Brownian dynamics* methods are equally as important to Soft Matter research, but they are beyond the scope of this work. The book by Jan Dhont [Dho96] provides a rigorous explanation of the Brownian dynamics framework.

2.5.1 Molecular dynamics

Molecular Dynamics simulations use the basic laws of Physics to understand the behavior of liquids. In particular, Newton's laws are used to evolve a system of particles through time and space. For a system of N particles, each with mass m_i, $i = 1, 2, \ldots, N$, the equations of motion are

$$\boldsymbol{F}_i(t) = m_i \frac{d^2 \boldsymbol{r}_i}{dt^2}\,, \tag{2.25}$$

with

$$\boldsymbol{F}_i(t) = \sum_j -\nabla_{\boldsymbol{r}_i} U(|\boldsymbol{r}_i - \boldsymbol{r}_j|)\,, \tag{2.26}$$

the force on particle i with position r_i exerted by particle j with position $\boldsymbol{r}_j$, and U the potential energy, which in the context of liquids this is the interaction potential between the particles.

To integrate these equations of motion, different integration schemes have been developed. At first, Alder and Wainwright [AW57] used the simple Euler method to integrate these equations. It was later understood that such scheme does not yield *stable* results, i.e., the integration scheme gives different results if used with different initial conditions each time. Integration methods must conserve important physical quantities, such as energy, momentum and phase space properties [RHC18]. The Verlet method was later introduced in 1967 by Loup Verlet [Ver67], in order to alleviate the drawbacks of the Euler method. These methods are crucial to obtain correct results from molecular dynamics computer simulations.

The main appeal of molecular dynamics computer simulations is the fact that *dynamical quantities* can be computed effortlessly, given the temporal nature of the formulation. Physical observables such as the *mean square displacement* and the *dynamic structure factor* [Dho96; HM13] are of great importance to the understanding of transport properties in liquids and more complex systems in Soft Matter.

2.5.2 Monte Carlo methods

Monte Carlo methods make use of randomness to explore a possible solution to a given problem. The basic idea is to exploit random sampling to obtain numerical results. Monte Carlo methods have successfully been applied to three main problems: optimization, numerical integration, and probability density function estimation [Kro+14]. It might seem odd that Statistical Physics is not mentioned here, but the reason for that is that it actually belongs to the wider range of probability density function estimation techniques. It is the purpose of this section to explore Monte Carlo methods and their application to the simulation of liquids, mainly because this is the method used in this work, and because in this work there is no dynamical physical quantity involved, only statical ones, i.e., the radial distribution function.

The basic idea of Monte Carlo methods in the computer simulation of liquids is to generate a sequence of configurations for a given system. The idea is to arrive at a probability distribution function close to the ensemble probability distribution function chosen to model the fluid. For instance, if a system is under the NVT ensemble, the sequence of Monte Carlo steps are expected to converge to the distribution function in Equation 2.9. Having arrived at this estimation, observables can the be computed, as in Equation 2.10. There is also other possibilities of ensemble distribution functions that can be used with Monte Carlo computer simulations, such as the isobaric-isothermal ensemble and the Gibbs ensemble [FS01].

In general, Monte Carlo methods are completely random. However, liquids cannot be subject to complete randomness due to their intermolecular interactions. In the case of the hard sphere fluid, the denominator of Equation 2.9 would be zero at least 50% of the time, which would incur in numerical instabilities and observables would not get measured properly. So, a *bias* must be introduced in order to *guide* the sequence of configurations to a path which could avoid unnecessary configurations. This bias is called *importance sampling*, and the most common rule, as well as the one used here, is the Metropolis rule [LB21]. Nevertheless, when a bias is introduced, there is one important rule that must *always* be obeyed: the condition of *detailed balance*. Detailed balance is a strong condition stating that if there is a transition from a state o to a state n, this should be balanced with the number of accepted trial moves that go from state n to state o. This can also be summarized in mathematical form,

$$p(o)\,T(o \Longrightarrow n) = p(n)\,T(n \Longrightarrow o)\,, \tag{2.27}$$

where $p(\alpha)$ is the probability to be at state α, and $T(\alpha \Longrightarrow \beta)$ denotes the probability to go from a state α to a state β. It might look like a simple rule, but this condition is what guarantees the ergodicity of the computer simulation [LB21], i.e., that the phase space is explored correctly and with physical significance. If this condition is not met, the results do not represent the physical properties of the system, and the observables cannot be computed properly.

In the NVT ensemble, the Metropolis rule that satisfies detailed balance is

$$\frac{p(n)}{p(o)} = \frac{A(o \Longrightarrow n)}{A(n \Longrightarrow o)} = \exp\left\{-\beta\left[U(n) - U(o)\right]\right\}, \tag{2.28}$$

with

$$A(o \Longrightarrow n) = \min\left(1, \exp\left\{-\beta\left[U(n) - U(o)\right]\right\}\right), \tag{2.29}$$

the proof that this condition satisfies detailed balance is omitted here, but it can be found in the book by Frenkel [FS01]. With this acceptance rule, it is guaranteed that the configurations that have the larger Boltzmann factor, i.e. $\exp\left\{-\beta\left[U(n) - U(o)\right]\right\}$, will be visited more frequently

rather than the configurations with a smaller Boltzmann factor, which will be avoided. In practice, the core of the Monte Carlo method for the computer simulation of liquids, which corresponds to the Metropolis rule, is implemented as follows:

- Select a particle i at random.
- Calculate the energy $U_i(o)$ of particle i.
- Move particle i randomly.
- Calculate the new energy $U_i(n)$ of particle i.
- Accept or reject the move according to Equation 2.28 and Equation 2.29.

Monte Carlo methods are efficient when static quantities, such as the radial distribution function, are computed. This is because time is not part of the computer simulation method, unlike the previously discussed Molecular Dynamics method. Monte Carlo methods can efficiently explore a large portion of the phase space due to their stochastic nature. Although the phase space might be large, it is often the case that a simple exploration of one or two replicas of the system are enough to obtain good estimations for a given observable. In this work, Monte Carlo computer simulations were used for all the observables computed throughout.

2.5.3 On the computation of the radial distribution function

The procedure to obtain the radial distribution function from computer simulations is as follows. For each Monte Carlo step a particle i is chosen at random. Then, all the particles that are a distance δr from the particle i are counted as a neighbor particle. After that, all the particles that are now at a distance $2\delta r$ from the particle i are counted. This is done after all the particles have been accounted for. If the interaction potential between the particles is truncated to a cutoff radius r_c, then this is the last value of the distance used to count for neighboring particles. When every particle has been counted, a histogram is built, which corresponds to a discrete distribution function of the neighboring particles. Now, this distribution function is not normalized, and the procedure to do so is as follows. A normalization constant is obtained by dividing the total number of particles at position r with the product of the total number of particles in the simulation, multiplied by the total number of configurations visited throughout the simulation, multiplied by the total volume of the spherical shell that lies between r and $r+\delta r$. In mathematical form, this can be summarized as follows

$$g(r) = \frac{1}{\rho}\left\langle \frac{1}{N}\sum_{i=1}^{N}\sum_{i\neq j}^{N}\frac{1}{2\pi r}\delta(r-\boldsymbol{r}_{ij})\right\rangle, \quad \boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j\,. \tag{2.30}$$

2.6 The Ornstein-Zernike Integral Equation

Up until this point, the presentation on Liquid State Theory has been driven by the possibility of calculating a given observable for a liquid. The method to compute such observables is to obtain a main quantity, the radial distribution function, and make use of any of the thermodynamic equations that relate $g(r)$ with the needed thermodynamic observable. There has been a discussion of how the $g(r)$ can be obtained through computer simulations. And yet, there has not been a discussion of how the $g(r)$ can be obtained analytically. The Ornstein-Zernike formalism

provides a way to obtain such analytical results, and it is the focus of the present section, as well as the main focus of this thesis. Despite its importance, the formal derivation of the equation is too rigorous to be presented here, so most of the details will be omitted here but left to specific references on the subject [HM13]. Nonetheless, the main results and physical interpretation shall be outlined here.

To understand the Ornstein-Zernike equation, a physical interpretation is defined as follows. If there is a system of N interacting particles with an interaction potential defined by $u(r)$, then it so happens that each of them are related to each other. It might seem like an obvious statement, but this relation has in fact two contributions. There exists a *direct* relation between particles and their neighbors, and an *indirect* relation between particles and the rest that may be far away from them. This interaction begs the question, *is there a way to relate the indirect and direct correlations between particles?* The answer is given recursively through the following definition. Let $h(r)$ be the *direct correlation function* that "measures" the direct correlations between particles; and let $c(r)$ be the *indirect correlation function* that "measures" the indirect correlations between particles, then both $c(r)$ and $h(r)$ are related with each other through the *Ornstein-Zernike equation* as follows

$$\begin{aligned} h(r) = c(r) &+ \rho \int_V c(\boldsymbol{r}')\, c(|\boldsymbol{r}-\boldsymbol{r}'|)\, d\boldsymbol{r}' \\ &+ \rho^2 \int_V c(\boldsymbol{r}'') c(|\boldsymbol{r}-\boldsymbol{r}'|) c(|\boldsymbol{r}'-\boldsymbol{r}''|)\, d\boldsymbol{r}'\, d\boldsymbol{r}'' \\ &+ \rho^3 \int_V c(\boldsymbol{r}''') c(|\boldsymbol{r}-\boldsymbol{r}'|) c(|\boldsymbol{r}'-\boldsymbol{r}''|) c(|\boldsymbol{r}''-\boldsymbol{r}'''|)\, d\boldsymbol{r}'\, d\boldsymbol{r}''\, d\boldsymbol{r}''' \\ &+ \dots \end{aligned} \tag{2.31}$$

Equation 2.31 can be rewritten recursively by noting the following factorization

$$h(r) = c(r) + \rho \int_V c(\boldsymbol{r}') \begin{bmatrix} c(|\boldsymbol{r}-\boldsymbol{r}'|) \\ + \rho \int_V c(|\boldsymbol{r}-\boldsymbol{r}'|) c(|\boldsymbol{r}'-\boldsymbol{r}''|)\, d\boldsymbol{r}'\, d\boldsymbol{r}'' \\ + \rho^2 \int_V c(|\boldsymbol{r}-\boldsymbol{r}'|) c(|\boldsymbol{r}'-\boldsymbol{r}''|) c(|\boldsymbol{r}''-\boldsymbol{r}'''|)\, d\boldsymbol{r}'\, d\boldsymbol{r}''\, d\boldsymbol{r}''' \\ + \quad \dots \end{bmatrix} d\boldsymbol{r}' \tag{2.32}$$

and thus arrive at the most common form of the Ornstein-Zernike equation, which for isotropic and uniform liquids reads

$$h(r) = c(r) + \rho \int_V c(r')\, h(|\boldsymbol{r}-\boldsymbol{r}'|)\, d\boldsymbol{r}' \,. \tag{2.33}$$

The Ornstein-Zernike integral equation provides an analytical way to obtain the $g(r)$ provided an interaction potential. Yet, in Equation 2.33 there is no mention or sign of any of such quantities. So how is it possible to obtain an estimate of $g(r)$? It turns out that the definition of $h(r)$ is key to answering this question, for the actual mathematical definition is $h(r) \coloneqq g(r) - 1$. As for the interaction potential, the information lies within $c(r)$ and $g(r)$. In the *low density limit*, the radial distribution function has the following asymptotic behavior

$$g(r) \to e^{-\beta u(r)}, \quad \rho \to 0 \,. \tag{2.34}$$

Also, when $r \to \infty$ then $g(r) \to 1$. Now, again in the low density limit, the asymptotic behavior for $c(r)$ that follows from Equation 2.33 is that $c(r) \to h(r)$. This also means that, with Equation 2.34, $c(r)$ has the following limiting definition

$$c(r) \to h(r) = g(r) - 1 = e^{-\beta u(r)} - 1\,, \quad \rho \to 0\,, \tag{2.35}$$

where the function $f := e^{-\beta u(r)} - 1$ is called the f-Mayer function, and it is an important quantity for theoretical analysis in Liquid State Theory. Thus, it follows from Equation 2.35 that

$$c(r) = -\beta u(r)\,, \quad r \to \infty\,. \tag{2.36}$$

An aditional important fact is that the compressibility equation, Equation 2.21, can be defined in terms of the $c(r)$ functions as follows

$$\frac{\beta}{\rho \chi_T} = 1 - \rho \int_0^\infty 4\pi r^2 c(r)\, dr\,, \tag{2.37}$$

this definition follows from the relation between $c(r)$ and the *structure factor* [HM13], but the details are omitted here.

2.6.1 Closure relations

Now that it has been stated the relation between the three quantities, $c(r)$, $g(r)$, and $u(r)$, the solution to the Ornstein-Zernike equation must be discussed. Until now, the discussion of the functions $c(r)$ and $h(r)$ has been focused on their low density limit. Still, liquids are *dense* systems and most of the time the purpose of studying them is to understand their properties for a wide range of density values, low and high density regimes. Furthermore, it is expected to study the phenomena that arises when the density varies in ways that *phase transitions* are induced. To provide a solution to this, Equation 2.33 must be solved for all possible values of the density. However, if looked at closely, Equation 2.33 has two unknowns, $c(r)$ and $h(r)$, thus cannot be solved as it is. For this reason, approximations must be introduced in the form of *closure relations*.

Closure relations come from diagrammatic expansions and density function theory [HM13], and are not simple to summarize in this section. The most common closure relations will be stated, along those that are used in this work. One of the most important closure relations is the *Percus-Yevick* [PY58] approximation. The most important fact from this closure is that it provides an analytical solution to the hard sphere model. The closure relation reads,

$$c(r) = g(r)\left[1 - e^{\beta u(r)}\right]\,, \tag{2.38}$$

and when used along the hard sphere potential found in Equation 2.22, it provides an exact solution for $g(r)$. Another common closure relation is the *Hypernetted Chain* approximation, which reads,

$$c(r) = h(r) - \beta u(r) - \ln g(r)\,. \tag{2.39}$$

The appeal of this closure relation is that it works quite well for interaction potentials with long-range repulsive behavior, such as the Yukawa or Lennard-Jones potentials [HM13]. Despite this, both closure relations have deficiencies when attempting to predict a phase transition. Moreover, these closure relations suffer from *thermodynamic inconsistency*. This is a phenomena that happens from the approximation nature of closure relations. Both these closure relations

give different results for a given thermodynamic quantity if computed from different routes. For instance, if the thermodynamic pressure P is to be computed using the Ornstein-Zernike and the Percus-Yevick closure relation, then, if Equation 2.19 is used, a particular result will be obtained. But if Equation 2.21 is used, and then Equation 2.20 to obtain the thermodynamic pressure, a different result will be obtained with respect to the one computed before. Furthermore, if the energy equation is used —Equation 2.17 —, and then using thermodynamic relations to obtain the pressure, a new result will be obtained, different from the previous two results. This constitutes a serious drawback of the Ornstein-Zernike formalism.

To alleviate this main drawback, Rogers and Young [RY84] proposed a new version of the closure relation, which in fact combines two closure relations, the Hypernetted Chain and the Percus-Yevick. With great success, this new and improved closure relation can reproduce the thermodynamics of the hard sphere model and related short-range interaction potentials. Another attempt at providing thermodynamic consistent closure relations were provided by Zerah and Hansen [ZH86], which combined different closure relations into one, in the same fashion as the Rogers-Young closure relation. Both of these approaches work as follows. Two main routes to compute a given thermodynamic quantity are chosen, and are forced to yield the same results through the use of a fixed parameter, called the *mixing parameter*. This value is obtained is an optimization problem by enforcing equality of the thermodynamic routes.

Another important closure relation is the one provided by Verlet and then modified by Kinoshita [Kin03]. It turns out that when hard spheres are studied, there are better closure relations than the Percus-Yevick approximations for larger density values. The Kinoshita modification reads,

$$B(r) = -0.5\frac{\gamma^2(r)}{1+0.8|\gamma(r)|}\,, \tag{2.40}$$

where $B(r)$ is the bridge function (see next section), and $\gamma(r) = h(r) - c(r)$. This closure relation is not thermodynamically consistent, but in this work this consistency will be explored in detail.

2.6.2 The role of the bridge function

All the possible solutions to the Ornstein-Zernike equation discussed so far have been just approximations. But the Ornstein-Zernike formalism is an exact integral equation, thus there should be an exact closure relation. Indeed, an exact relation exists which comes from diagrammatic expansions [HM13] and reads,

$$\ln\left[h(r)-1\right] = \beta u(r) + B(r) + \gamma(r)\,. \tag{2.41}$$

Still, it introduces a new function and quantity, the so-called *bridge function*. What role does the bridge function play in the exact solution of the Ornstein-Zernike equation? It turns out that if chosen correctly, the bridge function can make Equation 2.41 reduce to all previous closure relations. For instance, if $B(r) = 0$, the Hypernetted Chain closure relation is recovered. If now the bridge function takes the value of $B(r) = \ln\{\gamma(r)+1\} - \gamma(r)$, then the Percus-Yevick approximation is obtained. Given the true nature of the bridge function, physicists were drawn to understand the bridge function and proposed approximations for the bridge function, instead of a simple closure relation for the Ornstein-Zernike equation itself. Nevertheless, by approximating the bridge function, the same problems arise, i.e., thermodynamic consistency and defficient results for every possible interaction potential. So the most common way to solve the Ornstein-Zernike is through experience: knowing the properties of $u(r)$, a given bridge function is chosen, then

with this approximation the Ornstein-Zernike equation is solved, and a solution is obtained. Most of the time, the solution is extremely precise. But if there is not much understanding of the interaction potential, then several bridge function approximations must be tested. However, most of the important interaction potentials have been extensively studied, and it is already known how these behave and their properties. But it is still cumbersome to attempt to use all possible bridge function approximations. It seems like a task that could be *discovered* or *automated* in a way, and fortunately this is what Machine Learning excels at.

Chapter 3

Computational Intelligence and Machine Learning

In this chapter, the fundamentals of Computational Intelligence and Machine Learning are developed. Particularly, the focus of the chapter is to present the main tools used in this thesis, namely *neural networks* and *evolutionary algorithms*. To reach a general understanding of these tools, a brief description of learning mechanisms and numerical optimization is carried out.

3.1 Computational Intelligence

The first thing to address is the meaning and scope of *Computational Intelligence* (CI). With recent advancements in fundamental research in this area and its sub-fields, there seems to be a blurry definition of what exactly is CI, and there is no concrete one until now. For this reason, in this work the definition of CI is an umbrella term for several other applications. However, these applications are related to each other for the same reason that CI exists: to provide a computational solution to a problem using as inspiration the paradigms of nature-inspired intelligence. For instance, following the handbook by Kacprzyk and Pedrycz [KP15], the definition of CI is to be a collection of nature-inspired computational methods that provide solutions to problems where *hard computing* is inefficient or it not even suited to provide a solution to a given problem. Here, there is an important distinction that should be carried throughout the remaining of the work: that there are problems for which traditional tools are insufficient for the traditional problems. In this work, this is the philosophy used to provide solutions: that the traditional methods might seem hard and unfitting to provide solutions to the problems presented, and therefore new ways of approaching these solutions should be used.

In general, CI methods do not provide exact and accurate results, but this is expected. It does not mean that CI is providing the ultimate, best solution to a problem. Rather, it is providing an approximate solution that can be used later with more robust algorithms and methods. CI is not meant to be used as the sole method to solve a problem, but instead to help find *some* solution to a difficult problem. That is why CI is considered an umbrella term, because it comprises diverse fields that can be used to find such an approximate solution. Indeed, in this work, two main field of CI are used: *neural networks*, which is a part of ML, a sub-field of CI; and *evolutionary algorithms*, which are stochastic optimization methods inspired by the evolution mechanisms found in nature.

3.2 Machine Learning

In modern times, data is constantly being created and used to model the reality around us. However, traditional methods (hard computing) have not been enough to handle the large data sets created. For this reason, new ways of dealing with this information are needed. Most importantly, the need for automated discovery and *pattern recognition* within the data. Following Murphy [Mur12], ML is defined as the set of techniques that can automatically detect patterns in data, and use these patterns to create new predictions, or to perform other kinds of decision making. In other words, *the data creates the ML algorithm*, not the other way around. It is with data that ML methods work, however, these methods are essentially *function approximation* methods, which shall be discussed in a later section. For now, a brief overview of the different kinds of ML tasks and problems will be presented, focusing primarily on *supervised learning*. Nonetheless, there exist other types of learning, such as *unsupervised learning* [GBC16; HTF09] and *reinforcement learning* [SB18; KLM96], which are out of scope of this work, but remain an important part of modern ML theory.

3.3 Supervised Learning

The most common kind of ML methods is that of *predictive* learning. For a set $\mathscr{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with inputs $\boldsymbol{x}$ and outputs y, the goal of *supervised learning* is to learn a map between inputs and outputs. Here, $\mathscr{D}$ is the so-called *training set* and N is the number of *training samples*.

In most common cases, the inputs $\boldsymbol{x}$ are D-dimensional vectors that represent information about something. For instance, the height and weight of a person, or the evolution of the stock market throughout the years. These vectors are colloquially referred to as *features* or *attributes*. The general form of $\boldsymbol{x}$ is not defined, it can be anything from an image, a time series, sentences from a text, graphs, molecules, and so on.

In a similar fashion, the *response* variables y can be, in general, anything. However, there is a clear distinction given the forms that these variables can take. For instance, if the variable y has *categorical* values, the supervised learning task is considered a *classification* problem. Categorical values come from a finite set of possible values, $y_i \in \{1, \dots, C\}$, and might represent any type of discrete or nominal value. For example, it might represent the colors in a clothing line, the gender between people, and so on. On the other hand, if the values of y are real-valued, such that $y_i \in \mathbb{R}$, then the learning task is dubbed a *regression* problem.

3.3.1 Classification

In this section, the problem of classification is looked at with more detail. Although classification is not used at all in this thesis, it is helpful to discuss it as it makes it easier to understand the importance of supervised learning. It also creates a basis on which the problem of regression can then be generalized from.

In the problem of classification, the computer algorithm is given a data set $\mathscr{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, and is asked to specify to which category does the input belong to. Here, $\boldsymbol{x}_i$ is an n-dimensional feature vector. As mentioned before, the idea is for the algorithm to learn a map, or more precisely, a *function* such that $f : \mathbb{R}^n \mapsto \{1, \dots, k\}$, with k the total number of categories, or *classes*, to which the input can be assigned to. More specifically, when $y = f(\boldsymbol{x})$, the model assigns an input described by the n-dimensional feature vector $\boldsymbol{x}$ to a particular categorical value of y.

One of the most common uses of classification is object recognition. The goal of the object recognition problem is to decode a particular image with a specific object in it, and label it accordingly. An extremely popular data set for this kind of task is the MNIST handwritten digits data set [Lec+98], and its most common variation, the Fashion-MNIST [XRV17]. These data sets are comprised of several images and categories, and the goal is to *classify* each of the several categories. In the case of the handwritten digits, the goal is to specify which digit is represented in the image; in the case of the Fashion-MNIST data set, the goal is to specify the type of clothing. These data sets have become the standard benchmarks in the ML community for a long time. They are used primarily to test whether a ML algorithm is working properly, and if it is *accurate* enough. The word *accuracy* means something quite specific in ML theory, and will be discussed in a later section.

3.3.2 Regression

In *regression*, the goal is for the computer algorithm to learn a map, or equivalently a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, that predicts a numerical or real-valued output. The resemblance to the classification task is quite obvious: categories or classes can now be represented as a continuous value within the reals and there is no restriction for the number of classes. In a sense, this is the most general form of classification problem. So then, why make a distinction between the two problems? Most of the time, regression tasks are created to *predict*, *forecast* , or even *generate* outputs for a given data set $\mathscr{D}$.

One of the major applications of regression is *time series forecast*. This can be found mostly in economical and financial contexts [BBTLB13; SGO20], but there are also applications in bioengineering and medical situations [MPP18]. The goal here is to obtain a good approximation of the function f in order to *extrapolate* its domain to obtain new, and unseen, results. It is expected that ML methods, with their ability to find undiscovered patterns, can effectively predict and forecast outputs that are not in the data set $\mathscr{D}$. This is an extremely hard task, but one that has been finding a lot of applications and many groups have done extensive research on the matter.

Regression is an important task, and in chapter 4 it is the primary learning mechanism used to solve a particular problem in Liquid State Theory. The problem of regression, as stated before, is to learn a *good* approximation of the function f. Again, the measure of *good* is not clear enough, as was the case with *accuracy*, and both these concepts shall be discussed next.

3.3.3 Performance Measure

ML algorithms must be assessed on their abilities to perform a certain task T, either a classification or regression problem in the current discussion. It is common practice to extract a subset of the training data set $\mathscr{D}$ as the *testing set*, $\mathscr{T}$, to evaluate the algorithm in the task T. The measure depends on the task T, as well as the type of data used.

In classification tasks, the *accuracy* is one of the most general performance measures. The accuracy is simply defined as the proportion of data points in $\mathscr{T}$ for which the model produces the correct output. This measure is quite strict in the sense that if there are examples that are *mislabeled*, this is carried on to the ML model.

Another common performance measure for classification tasks is the receiver operating characteristic, also known as the ROC [HTF09]. More specifically, the area under the ROC curve. This measure of performance is created by plotting the *true positive rate* against the *false positive rate*. This measure is used because these rates are more permissive, since they stem from the well

established type I and II errors from statistical theory [Ric06]. Depending on the data set, other measures can be used, such as the F1 score, the Jaccard index, Akaike information criterion, and others [Mur12].

Regression tasks are different when measuring performance since these methods attempt to approximate continuous, real-valued functions. So in a sense, one can simply use *metrics* in the mathematical analysis sense of the word. For instance, the use of the L^2 norm, also known as the so-called Euclidean norm, defined as

$$\|y - \hat{y}\|_2 = \sqrt{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}, \tag{3.1}$$

is extremely common in regression tasks. Here, the training examples y are measured against the output value from the ML model, $\hat{y}$. Another extremely common, and profoundly useful metric, is the L^1 norm,

$$\|y - \hat{y}\|_1 = \sum_{i=1}^{N} |y_i - \hat{y}_i| . \tag{3.2}$$

The L^1 has the amazing property that it can create *sparse* representations of the learned function f. This is the basis of the LASSO method [HTF09], a very useful and common ML method to do *variable selection*, the discrimination of variables that are useful or not to the prediction of the model; and *regularization*, which is adding information in order to solve a problem that might seem hard or impossible to solve, also known as an *ill-posed* problem [GBC16]. When generalizing these metrics to the full data set, they take on different names. For instance, the squared L^2 norm takes the name *mean squared error*, defined as

$$MSE(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{3.3}$$

where N is the total number of examples used to compute the MSE.

Similarly, the L^1 norm can be generalized to the *mean absolute error*, defined as

$$MAE(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| . \tag{3.4}$$

These generalizations might seem trivial, but they allow for more general formulations, as they are in fact *estimators* that measure the average of the outputs. When measuring the performance of regression models, the goal is to *minimize* these errors, with zero being the optimal value.

3.3.4 Approximation Theory

Before closing this brief overview on ML theory, a note on approximation theory is in order. The link to ML has to do with the striking resemblance between both classification and regression tasks. Both problems seem to be, in a sense, the same exact problem which is to learn a *map* or *function* that relates an input with an output. However, this problem is not new, and in fact, it has been extensively studied before, so much so that is has created a branch within mathematics, called *approximation theory*.

The primary goal of approximation theory is simple: to understand how functions can be *best* approximated with even simpler functions [Tre13]. This sounds analogous to the learning

tasks, however, there was no mention of simpler functions in the previous discussion. This is because to see this more clearly, there has to be a detailed explanation of each of the models used. For instance, *Gaussian Processes* [RWB06] can approximate an arbitrary function with Gaussian functions, or more precisely, with normal probability distributions. In any case, there is not enough space in this work to talk about all the possible methods and how these are formulated, even if they are formulated based on simpler functions or not.

However, the discussion can be simplified by noting the following. If looked at closely, Equation 3.1 and Equation 3.2 look quite similar. Indeed, there is a generalization of this norm, called the L^p norm,

$$\left\| y - \hat{y} \right\|_p = \left(\sum_{i=1}^{N} \left(y_i - \hat{y}_i \right)^p \right)^{1/p}, \tag{3.5}$$

which generalizes the Euclidean norm to a more general norm, defined now in function spaces called L^p spaces [Rud13]. These spaces are, roughly speaking, a generalization of vector spaces where the basis that span the linear spaces is comprised of functions instead of vectors. There is, however, a particular form of the L^p norm called the *supreme norm*, or equivalently, the *infinity norm* defined as

$$\left\| y - \hat{y} \right\|_\infty = \max_i \left| y_i - \hat{y}_i \right|, \tag{3.6}$$

also known as L^∞, and it is a function space that contains all the essentially bounded measurable functionss [Tao11].

Why is L^∞ so important in this context? The answer is one simple, but outstandingly powerful theorem: the Stone-Weierstrass theorem [Sto37; Sto48]. In 1885, Karl Weierstrass proved that any function can be approximated by a polynomial of a given degree. This result is so powerful that it created the field of approximation theory. And it is so compelling, that it might arguably be the fundamental theorem of ML theory. This is the reason for the L^∞, and its relation to the learning tasks should become clear once the main theorem is stated.

Theorem 1 (**Weierstrass approximation theorem**)**.** *If f is a continuous real-valued function on $[a, b]$, and if any $\epsilon > 0$ is given, then there exists a polynomial P on $[a, b]$ such that*

$$\left\| f(x) - P(x) \right\|_\infty < \epsilon$$

for all $x \in [a, b]$.

The proof is left for a specialized text on the subject, for instance the book by Richard Beals [Bea04]. Nonetheless, it is important to see the striking resemblance between the kinds of learning discussed so far, and Theorem 1. On one hand, classification and regression both deal with finding the best mapping between inputs and outputs. On the other hand, Theorem 1 states that every function can be approximated by another, simpler function. So, in a sense, these problems are related. One seeks to find the best mapping, while the other guarantees that there is such a mapping. Theorem 1 helps visualize that the main problem of ML theory is to approximate the underlying function between inputs and outputs in a data set. However, the question of how to find such polynomial P is still open.

With the presentation of the Weierstrass approximation theorem it is now time to move on to the case of neural networks, which are in fact a way to build a polynomial P with the help of data. If looked at closely, the theorem only states that there is such polynomial, but does not specify how it might be constructed or formulated. There are roughly two ways: with pure mathematics or with the help of data. To use mathematics is to turn to the help of approximation theory; in

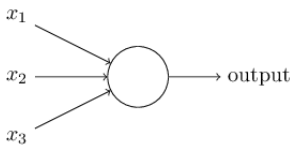

FIGURE 3.1: A representation of the original McCulloch-Pitts perceptron. From [Nie15].

this case orthogonal polynomials are used, such as Tchebyshev polynomials, to build such a polynomial P. Of course, many other approximation methods can be used [Tre13]. On the other hand, turn to statistics and data, and seek the help of ML theory, such that the data is used to construct such a polynomial. Before moving on, it should be pointed out that this is not a unique or novel way of seeing the problem of learning. Indeed, a more general formulation is based on probability theory, which uses much more rigorous arguments to link the relation between ML theory and approximation theory. The book by Murphy [Mur12] is a great reference for that.

3.4 Neural Networks

In this section, neural networks (NN), their architectures, training and mathematical formulations are discussed. Although, this is just a short overview of the full theory of NN. In recent years, NN have been the most used, researched and applied method in modern ML. Up to this day, there are unmeasurable number of research articles and books devoted to NN. So this space is obviously quite small to fit all the information about them. It is for this reason that this section will focus primarily on how NN are trained and how they learn a mapping given a data set. For a more thorough understanding of NN, these references [Meh21; GBC16; HTF09; Ber+21] should suffice.

3.4.1 Motivation

The basic idea of NN is to mimic the way the brain works, and more specifically, the way neurons interact with each other. In the modern form of NN, this interaction is highly idealized in the sense that the actual human brain does not follow the same form of interaction, but it is somewhat a rough approximation. In 1947, Pitts and McCulloch [PM47] devised the *perceptron*, an idealized model of a neuron. This perceptron can be observed in Figure 3.1. Here, the middle circle is the perceptron or *unit*, and it accepts three inputs x_1, x_2, x_3, or equivalently $\boldsymbol{x} = (x_1, x_2, x_3)$ with $\boldsymbol{x}$ a 3-dimensional input vector. The output of the McCulloch-Pitts perceptron is a single binary output, it can either return a 0 or a 1. Then, in 1958, Rosenblatt [Ros58] extended this idea and introduced a way to compute the output of the perceptron. He introduced what he called *weights*, and gave a weight to each of the three inputs, w_1, w_2, w_3. Later, he proposed to compute the output as a weighted sum of each of the three inputs, such that the output would have the following form,

$$\text{output} = \begin{cases} 0 & \text{if} \quad \sum_i w_i x_i \leq \varepsilon \\ 1 & \text{if} \quad \sum_i w_i x_i > \varepsilon \end{cases}, \tag{3.7}$$

where ε is a *threshold* value. This model is simple, yet effective. If one input would be more important than the other, the weight could be adjusted to reflect this. As simple as the model

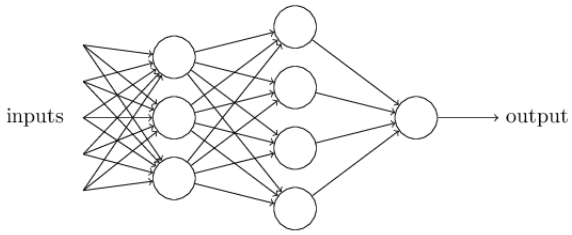

FIGURE 3.2: A representation of several perceptrons connected with each other.
From [Nie15].

is, it is a good approximation of how decision-making models can be constructed efficiently. Furthermore, a more complex decision-making system can be built if several perceptrons are connected with each other. After all, the inputs and outputs of a single perceptron do not have limitations, and they can well be defined to be the inputs and outputs of other perceptrons as well. Instead of using a simple perceptron, a more complex structure can be built upon using several of them. If *layers* of perceptrons are stacked between each other, a more complete model of decision-making is created. This is the case of the *multilayer perceptron*, or MLP, shown in Figure 3.2.

Despite this, the model has serious drawbacks. For one, there is no clear understanding of the threshold value and what it is. On the other hand, having the perceptron output just two values might make it inaccesible in the most general case. After all, the idea of NN is to learn a mapping, a function that in general is real-valued. Finally, how are the weights adjusted? Are these values picked randomly? Does the user choose them? If in fact NN are learning models, then there is no real reason why the user would likely pick the weights for each problem they need to solve. All of this questions will be addressed in a later section, when training is discussed in more detail. For now, architectures and activations functions will be presented next. These topics will provide answers as to how the output of perceptrons can be modified to return other values, rather than just a 0 or 1.

3.4.2 Architectures and Activation Functions

Now that a complete network of perceptrons has been defined, it is time to discuss one important issue with them. First, it might be easier to deal with a different notation than just using threshold values. It is simpler to use the language of Linear Algebra and define the output expression shown in Equation 3.7 using the dot product as follows,

$$\text{output} = \begin{cases} 0 & \text{if} \quad \boldsymbol{w}\cdot\boldsymbol{x}+b \leq 0 \\ 1 & \text{if} \quad \boldsymbol{w}\cdot\boldsymbol{x}+b > 0 \end{cases}, \tag{3.8}$$

where $\boldsymbol{c}\cdot\boldsymbol{d} = \sum_i^N c_i\, d_i$ is the dot product between the N-dimensional vectors $\boldsymbol{c}$ and $\boldsymbol{d}$. Now, there is a new value, called the *bias* term, defined as b in Equation 3.8. This new term is the value that is added to each perceptron and it can be interpreted as a measure of how easy it is to get the perceptron to output a 1.

There is now the following issue to address, and that is that the output is only 0 or 1. The issue here is that NN are expected to learn complex mappings between inputs and outputs, and if the weights are adjusted a minimal amount, for instance a value of Δw, then the output should

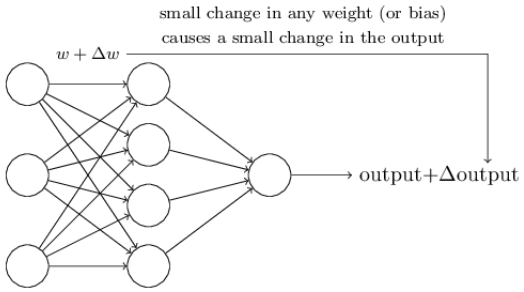

FIGURE 3.3: Visualization of the basic training mechanism in a neural network. By inducing a small change to the weights, a small change in the output is expected. From [Nie15].

change just a small amount, proportional to the change in the weights. However, if the output is either a 0 or 1, then a small change will trigger a drastic change, and learning will not be entirely possible. A representation of this idea can be seen clearly in Figure 3.3. The pertaining question here is, is there a way to change this behavior? The answer is yes, but to do so, a new type of perceptron, as well as a NN *architecture* must be introduced.

The *sigmoid* perceptron is a modification to the McCulloch-Pitts original perceptron which can provide a continuous value to the output by using a continuous function. This function, the *sigmoid* function, is defined as

$$\sigma(z) := \frac{1}{1+\exp(-z)} . \tag{3.9}$$

Instead of the output being just 0 or 1, Equation 3.8 is now replaced by the following equation, using now Equation 3.9,

$$\text{output} = \frac{1}{1+\exp\left(-\sum_i w_i x_i + b\right)} . \tag{3.10}$$

This function might seem to drastically change the behavior and original intention of the McCulloch-Pitts perceptron. However, in reality, the sigmoid function is just a smoothed version of the original perceptron. With this modification, when a small change is done in the weights of the network, a small change will be expected in the output. The amount of change, and how this change happens, will be discussed in the next section when training is presented.

If different functions are used instead of the sigmoid function, the network is expected to behave differently with each one. The choice of the function is so important, that not only have these function taken in a particular name—*activation functions*—, but there has been extensive research in this area [CC95; Nwa+18; Ago+15; RZL17]. In practice, the choice of activation function can make a NN perform better in certain classification tasks, e.g. object recognition, segmentation, and others; as well as in regression tasks. Activation functions can also help the NN perform faster, which is an important aspect of the training mechanisms and practicalities of NN [FL19; Bis+21].

Finally, it is important to mention that the *topology*, or *architecture*, of the network has been subject to significant research in recent times. For instance, if more layers are added in between layers, the NN takes in a different name, a *deep neural network*. Deep neural networks have been the quintessential model of the revolutionary and extremely popular field of *deep learning* [GBC16]. In deep learning, NN architectures are extended to be larger, deeper, and more complex than simple multi-layer perceptrons. These new models can be thought of being several

multi-layer perceptrons, each stacked upon each other. Furthermore, deep learning attempts to generalize the simple percpetron model and introduce better models that can perform with greater accuracy in certain tasks. The explosion of deep learning came when *convolutional neural networks* were applied successfuly in the AlexNet architecture [KSH12]. There other kinds of important variations in the topology of the network, such as the U-shaped networks use for medical image segementation [RFB15]; V-shaped networks used for remote sensing [APA20]; and very deep topologies that can track images in real time [RF18].

3.4.3 Training

Choosing the correct activation function and the appropiate topology is hard, and a lot of experimentation is needed. But even if these could be chosen automatically, there is still a latent issue here: how weights can be adjusted. Up until this section, this issue has been left aside and it is now time to address it. To do so, it is imperative to remember the sole purpose of NN and ML methods for classification and regression tasks: *to learn a mapping or function*. Assuming it is possible to have a training set $\mathscr{D}$ with enough examples, containing both inputs and outputs, the function that determines the relationship between them can be learned by a NN. In general, this is referred to as the *universal approximation theorem*, which shall be the topic of the next section.

The topic for this section is to answer the question of how this learning is carried out in the first place. And to do so, it is useful to recall the information from Figure 3.3 in that, if the weights are modified then the output is expected to change. Now, there are several questions that need an answer in that figure. For instance, what is the proportion between the change in the weights to the change in the output? If the weights are changed by half, is the output expected to change in the same amount? For the case of classification, there are only categorical outputs, so how are changes to the weights passed on the output? It seems that there needs to be a quantity that must inform the *learning* mechanism how much should the weights change in order to get the correct output from the network.

In order to address those questions, there needs to be a relationship between the output, the weights and the bias. Fortunately enough, calculus provides such information in the following way,

$$\Delta \text{ output} \approx \sum_j \left[\frac{\partial \text{ output}}{\partial w_j} \Delta w_j \right] + \frac{\partial \text{ output}}{\partial b} \Delta b \,, \tag{3.11}$$

where the sum is over all the weights in the network, w_j, and ∂ output $/ \partial w_j$ and ∂ output $/ \partial b$ denote the partial derivatives of the output with respect to the weights and bias, respectively. In other words, these derivatives represent *by how much* the output changes if the weights and bias are modified. Yet, there is still even more information from Equation 3.11 that is extremely useful. If looked at closely, Equation 3.11 actually represents a *linear combination* of terms that, together, form the change to the output. This means that both, the weights and the bias, must contribute to the change in the output, and the amount is provided by the partial derivatives. So with just this much, the answer to the question of *how much* is now answered. But now there is the burning question of how to compute the partial derivatives. Well, this question is easy to answer, because assuming that the NN is using sigmoid neurons, then the derivative can be reduced to the following expression,

$$\frac{d}{dz}\sigma(z) = \sigma'(z) = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right) = \sigma(z)\,(1-\sigma(z))\,, \tag{3.12}$$

where some steps were skipped to avoid taking too much space. Recalling that the ouput from the simoid perceptron is given by Equation 3.10, together with Equation 3.12, the partial derivatives from Equation 3.11 can be obtained without much effort. In more general cases when the perceptron uses a different activation function than the sigmoid, the computation of derivatives is carried out using modern tooling known as *automatic differentiation* [Bay+18].

Now, the issue of addressing how the learning mechanism is informed can be solved by introducing the *cost function*. The cost function, also referred to sometimes as *loss* or *objective*, is a function that measures how the NN is approximating the underlying function. This is, in fact, the same concept as the performance measures encountered in subsection 3.3.3. In fact, cost functions are metrics that address how good the approximation of the NN is. And so this issue has also been addressed. However, how is this cost function, and Equation 3.11 used to make the NN learn the mapping? For this, we turn to optimization and the powerful method of *gradient descent*. To help illustrate the learning mechanism, the mean squared error in Equation 3.3 will be used. However, this is not the only metric that can be used, and in fact several research advancements have been carried out in this direction [Fon+10; Li+16].

Numerical optimization is a rigorous mathematical subject, and the use of gradient descent techniques can not be summarized in this section. For that, the great book by Nocedal and Wright is an excellent reference for this topic [NW06]. Here, only the main results will be mentioned and used. In optimization, specifically *unconstrained optimization*, the task is to find the value of an input vector $\boldsymbol{x}$ that *minimizes the function* f, such that $f(\boldsymbol{x}) = 0$. The fact that it is unconstrained is because there is no restriction in the bounds of $\boldsymbol{x}$, or in the form of f, which in general is a non-linear function. In the context of optimization, f is referred to as a cost function, or objective function. Now, a relation between optimization and NN can be made by noting the following fact. The purpose of NN is to learn the function that relates inputs and outputs in a given data set. Or in the context of optimization: *to minimze the error between the true values and the approximation values*. The *true values* correspond to the outputs in the data set, and the *approximation values* to the outputs from the NN. So, in essence, the task is to find the best weights w_j, and the best bias b, that minimize the following cost function,

$$C(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{i=1}^{N} \left[y_i - \hat{y}_i(\boldsymbol{w}, b) \right]^2 . \tag{3.13}$$

Here the sum goes over all N training samples in the training data set, and the output from the NN $\hat{y}_i(\boldsymbol{w}, b)$ depends on the values of the weights $\boldsymbol{w}$ and bias b.

There is now one step left for the training procedure to be completely characterized, and that is to define a way to *update* the weights and bias accordingly so that Equation 3.13 is minimized. For this purpose, *gradient descent* is used. Gradient descent is an iterative optimization method based on first-order derivative information. The idea is to use the information from the gradient of the objective function to drive the minimization to minimum, which in general corresponds to a *local minimum*. The derivation of the method is left for a specialized reference on the matter [NW06], and the main results will be used here. In the context of NN, the idea to update the weights and bias so as to minimize the cost function. Thus, the gradient descent method is introduced, which follows the expression,

$$\begin{aligned} \boldsymbol{w}_{j+1} &= \boldsymbol{w}_j - \eta \frac{\partial C}{\partial \boldsymbol{w}_j} \\ b_{j+1} &= b_j - \eta \frac{\partial C}{\partial b} \end{aligned} \tag{3.14}$$

where the index j represents each of the steps taken to update each value. These steps are also sometimes referred to as iterations. Hopefully, by doing several updates of Equation 3.14, then Equation 3.13 can be minimized and the appropiate value of the weights and bias would have been found.

In practice, this simple rule is not actually followed for one simple reason. If looked at closely, Equation 3.14 actually takes in all the information gathered so far, and this includes the training examples, which are encoded in the cost function. But if the data set contains thousands of examples, then the optimization can not be carried out due to current limitations in modern hardware. So a simple modification is used, where instead of choosing all the training examples at once, a small sub-set is chosen randomly and then the gradient descent is used on this sub-set. This procedure is repeated until all the training examples have been selected. This is referred to as a *pass*. At the end of this pass, all the obtained values are averaged, and this average is used to update the weights and bias. This can be repeated for as many *passes* as needed, or wanted. This simple modification is the actual method use, and goes by very popular name of *stochastic gradient descent*. Extensive research has been conducted in this area as well, from introducing simple accelerating terms to the gradient descent rule. For instance, the Nesterov momentum gradient descent method [Rud17], or the Adam method [KB17], which is in fact the method used in this work and will be presented at a later part.

To summarize, the training mechanism of NN is as follows. Assuming a training data set $\mathscr{D}$ is available, then a cost function is chosen. In practice, the most common has the form of Equation 3.13 but it can be any other. Then, the weights and biases are initialized at random. After that, the main loop of the algorithm is to repeat the steps shown in Equation 3.14 while keeping track of the minimum. When a certain minimum has been achieved, a performance measure can be used to check wheter the NN has a good performance for the task given. This concludes, roughly speaking, the common practice of modern ML.

3.4.4 Universal Approximation Theorem

To close this section on ML and NN, an overly important fact of NN must be stated. Before, it was mentioned that the goal of tasks such as classification and regression is for the model to learn or approximate the function that maps the input to the output for a given data set. For this reason, NN were presented, along with their learning mechanism. However, what *guarantee* is there that NN *will* approximate the function? Further, *can NN, in fact, approximate a function*? In a previous section, it was briefly mentioned that NN can in fact *learn* any function, and that will be the topic of this section.

Indeed, just as mentioned before in the section for Approximation Theory, there exists a similar result for NN. This result, which are actually a series of results, come from the mathematical research of NN. The results are known altogether as the *universal approximation theorem*. To really understand these results, advanced mathematical results should be employed. However, there is no need to extend the current section to such lengths. Instead, following the research from Hornik and Cybenko [HSW89; Hor91; Cyb89], the main results will be stated here to be interpreted.

In short, Hornik and Cybenko showed that, given the sigmoid function and a given topology, there exists an arbitrary number of nodes that are needed to *approximate* any *continuous* function. The topology is a three layer topology, similar to the one in Figure 3.4, where there is an input, a hidden and an ouput layers. This results is also similar to Theorem 1; however, in this case there is no polynomial but a NN instead. The arguments that follow this result is beyond

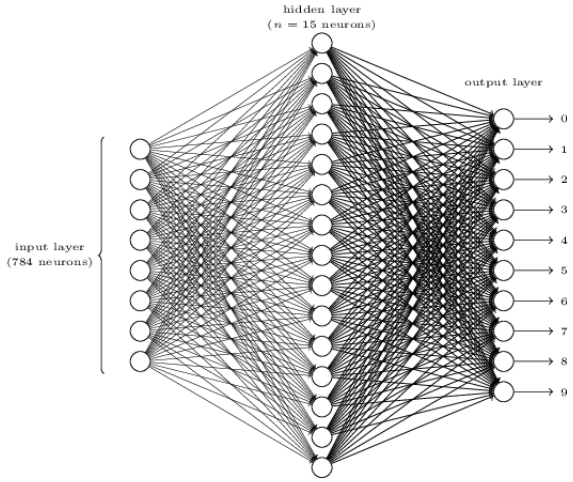

FIGURE 3.4: A multi-layer perceptron with three layers: the first layer is the *input layer*; the middle layer is known as the *hidden layer*; and the last layer is known as the *output layer*. From [Nie15].

the scope of this work, but the result remains valid. Then, as the research for the mathematical framework of NN advanced, newer results have seen the light that extend these results even more. Now, there is a clear understanding that not only the sigmoid function can be used, but any non-affine continuous function that it continuously differentiable [Par+20]. Furthermore, there is also no need to restrict the case to just width, but also depth, from which the generalization to deep learning follows [Zho20; Ber+21]. In this work, these guidelines are used to create simple topologies for NN that can actually satisfy the conditions of the theorem. A toplogy similar to Figure 3.4 with a large amount of nodes for each layer, which should suffice to satisfy the conditions of the theorem. However, in practice, it has been shown that *overparametrization*—when there are way more weights and bias to adjust than needed —is the key to generalization in NN [Ney+18; CMR21]. This means that most of the time, the common practice is to just use deep topologies, which mean a large number of layers and a large number of nodes for each layer, and use this overparametrization to leverage the generalization that this provides. So, in a sense, one might say that the universal approximation theorem is always satisfied in modern ML practice.

3.5 Evolutionary Computation

In this section, Evolutionary Computation (EC) is presented and discussed. Much like the driving force of perceptron is to mimic the brain and its intelligence, the inspiration behind EC is the way nature adapts itself through evolution [Fog00]. Although, in reality, evolution takes millions of years but it eventually creates variations that adapt and *converge* to a particular goal. This particular goal, as described by Darwin, is to be fit to its surroundings and survive. The idea behind EC is to use this powerful mechanim of evolution and apply it to complex problems, with the use of modern computational and mathematical resources to facilitate and empower a simplified version of evolution. In the most general case, evolution is roughly a two-step process: first comes *variation* and then comes *selection*. Algorithms based of EC use these properties and, depending on how these variations and selections occur, the algorithms are named and used differently. In particular, for this work the focus will be *evolution strategies*, which will be

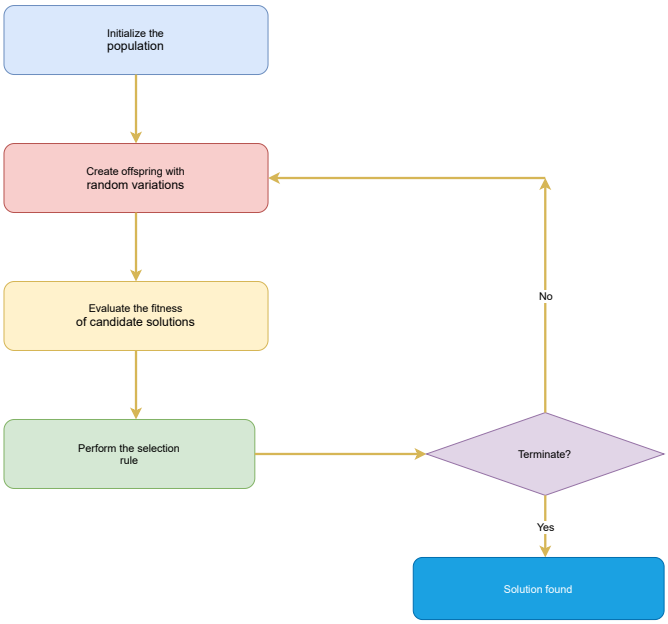

FIGURE 3.5: Evolutionary algorithms always start by creating a population of solutions. New solutions are then created by varying those from the original population. Then, the solutions are measured with respect to how well they address the task. Finally, a selection criterion is applied extract the best solutions so far. The process is iterated using the selected set of solutions until a specific criterion is met.

address in a later section. However, there is also *genetic algorithms*, *evolutionary algorithms*, and other [KP15].

As mentioned before, there is a common procedure in evolution, and this has been translated to computer science and its application in EC. The basic procedure is as follows: for a given problem, an initial population of possible *solution* is created; then, those solutions that are *fit* or *approximate* the solution are kept, those that not are discarded; finally, the successful candidate are mixed, creating offsprings with *variations* of all the other possible solutions. With these steps, a common EC algorithm can be constructed, and in general this is case. The advantage of using EC is that of *adaptability*. Many of these algorithms are suited for hard problems, some of which are impossible to solve with traditional methods, or at least take a long time to find a solution, such as the famous traveling salesman problem [DG97]. Flexibility, adaptability and generalization are the advantages of EC algorithms. When used in the popular StarCraft computer videogame, the AlphaStar system [ACT19] performed surprisingly better than the top players in the world.

However, the lack the general ease of use in some cases, as well as the difficulty to implement them might be troublesome in some other cases. Also, EC algorithms tend to have many parameters to adjust. For instance the total number of solutions in the population, the number of offsprings, and so on. In the case of EC algorithms, there is no *one size fits all*, so this can make them hard to deal with. In the present work, EC algorithms will be used for non-linear unconstrained optimization, and in particular, in the sub-field of *derivative-free optimization*. This will be discussed in the next section.

EC algorithms have been used for optimization in cases where the objective function is not

smooth, does not posses a *continuous derivative*, or when it is too *costly* to search for a solution in finite time [KP15]. The basic functionality of EC algorithms for optimization follow the diagram from Figure 3.5. With EC algorithms, the idea is to form an initial population of solutions, then to vary those solutions while keeping track of the fitness of the objective function. By varying the possible solutions, it is expected that these will *follow an evolutionary process*, and eventually reach a minimum. In particular, EC algorithms are best suited for *global optimization*, which is the search for minima in a large search space, without reaching any local minima along the way. The purpose behind this is to *explore* the search space and look for the values that best minimize the function. After all, EC algorithms are *approximation algorithms*, i.e. this methods do not attempt to find the best solutions, but to *find a possible solution* to the problem.

3.5.1 Derivative-free and Black-box Optimization

In optimization tasks, the best algorithms are those that include some kind of information about *derivatives*. For instance, the Newton method, and Newton-like methods such as BFGS [NW06], are some of the best for their speed, accuracy and convergence properties. Mathematically rigorous frameworks can be used to understand these algorithms, and can also be used to understand their convergence properties. Another similar optimization method was encountered before, the gradient descent method, which by leveraging the gradient information of the objective function a local minimum could be found by iterative steps.

However, the derivative information is not always available. When such cases are encountered, the only methods capable of providing solutions are known as *derivative-free* methods. These methods try to search the best solution, using different kinds of strategies, such as the trust-region [BSS87], quadratic polynomial approximation [Pow02], and the most recent and interesting approach is that of surrogate model optimization [FK09]. In each case, the objective function is approximated using some form of interpolation, or surrogate model, and the original search space is projected onto a new search space where the approximation can be optimized efficiently. With this approach, derivatives are no longer needed and optimization can be performed, but not without its drawbacks. First and foremost, the method relies entirely on function evaluations. If the function is *costly* to evaluate, then this methods will take a long time to find a suitable solution. Further, the search space will be explored, and this can considerably slow down the optimization procedure. After all, these methods try to explore and map the whole search space, but if the search space is quite large, the optimization procedure will suffer greatly in terms of performance. Finally, the *no free lunch* theorem states that, averaged over all optimization problems, all optimization methods perform equally well [Ada+19]. This means that if a particular derivative-free method is not providing sufficiently good results, then there must exist another optimization method that should give sufficiently good solutions. In other words, several algorithms must be tested against each other, given that there are enough computational resources to do so.

As mentioned before, sometimes there is not derivative information about the objective function, but more than that, objective function evaluations are *costly*. This means that, for a given input $\boldsymbol{x}$ and an objective function $f\colon \mathbb{R}^n \mapsto \mathbb{R}$, the evaluation $f(\boldsymbol{x})$ will take a long time. Further, in most cases when this is true, the *analytical or functional form* of the objective function *is not known*. When a particular objective function satisfies these conditions, it is referred to as a *black-box function*. When optimizing costly black-box function, the main goal is to *minimize* the number of evaluations of the function. This is done by sampling the search space *intelligently* so as to not evaluate the objective function where there is no sign of a minimum.

Black-box functions can, in general, be any type of function, either mathematical or computational. When the function is computational, this can represent code that performs several tasks, each of which might take considerable amount of time and computing resources. In such cases, black-box optimization algorithms are the key to solving the problem. Or maybe the black-box function is not too costly, but instead the true form of the function is not known. Again, black-box optimization algorithms are the best choice. In the next section, a particular EC, black-box optimization algorithm will be reviewed and presented, which will be extensively in a later chapter of this thesis.

3.5.2 Natural Evolution Strategies

One algorithm that stands out for its robustness and flexibility in dealing with black-box optimization tasks is the *natural evolution strategy* algorithm (NES), presented in 2014 by Wierstra *et al* [Wie+14]. This algorithm is a derivative of the Evolution Strategies (ES) family of algorithms, which follow closely the principles of EC. In ES algorithms, the principal characteristics are the ease of dealing with high-dimensional optimization problems, and low number of function evaluations. Also, unlike other EC algorithms, ES methods evaluate the fitness of the functions in batch instead of individually, allowing for more efficient computation and solution space search in general. The most prominent algorithm in the ES family of methods is the CMA-ES, or Covariance Matrix Adaptation-Evolution Strategy [Han06]. In general, ES methods do two main things. To perform *variation*, multivariate normal random vectors are used. To perform *mutation*, these algorithms modify different aspects of such multivariate normal distribution. The case of CMA-ES is to modify and adapt the covariance matrix in order to perform better at each step.

However, the method of NES is different. Instead of modifying the covariance directly, the information on the *natural gradient* is exploited. To understand this method more clearly, let $f\colon M \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ be a black-box function, and the problem of optimization is to search for a vector $f(\hat{x}) \in M$ such that a *local minimum* is found, where the local minimum is defined as,

$$\exists\, \epsilon > 0 \quad \forall x \in M : \ \|x - \hat{x}\| < \epsilon \Rightarrow f(\hat{x}) \leq f(x)\,. \tag{3.15}$$

Now, the goal is not to minimize the objective function directly, but to minimize an *expected fitnesss* under a particular search distribution. This means that the parameters searched will define a continuous probability distribution. This cost function is now expressed as,

$$J(\theta) = \mathscr{E}_\theta \left[f(\boldsymbol{x}) \right] = \int f(\boldsymbol{x})\, \pi(\boldsymbol{x}\,|\,\theta)\, d\boldsymbol{x}\,, \tag{3.16}$$

and using this cost function, together with a gradient descent-like rule, the purpose of NES is to find the best *probability distribution* that will eventually be sampled to obtain a minimizer for the original objective function. In other words, the NES method is not looking for the minimizer itself in a search space, but it is looking for the probability distribution that, when sampled, will give in average the minimizer for a particular local minimum of the objective function.

Still, the gradient information is unknown, as well as the true form of the search distribution, $\pi(\boldsymbol{x}\,|\,\theta)$. But it is also impossible to obtain a gradient estimation without explicit information about the probability distribution. Here is where the NES provides a novel way of dealing with

this problem. First, the search distribution is fixed to be a multivariate normal distribution,

$$\pi(\boldsymbol{x}\,|\,\theta) = \frac{1}{(2\pi)^{d/2}\det\boldsymbol{\Sigma}} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{3.17}$$

with $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$ the covariance matrix and $\boldsymbol{\mu} \in \mathbb{R}^{d}$ the mean vector of a multivariate normal distribution. The parametrization $\theta := \langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle$ is the link between these probability distributions. With this information, the gradient is obtained with some important mathematical manipulations, which is approximately,

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda}\sum_{i=1}^{\lambda} f(x_k)\,\nabla_\theta \log\pi(x_k\,|\,\theta)\,, \tag{3.18}$$

and it is also known as the *search gradient*, and the gradient descent rule is just

$$\theta_{j+1} = \theta_j + \eta\nabla_\theta J(\theta)\,.$$

The parameter λ in Equation 3.18 is the population size of the method, which in general is a small number, $\lambda \approx 20-35$.

However, this is just the search gradient, and it has some serious disadvantages. One of which is the fact that the gradient descent update rule is not independent of the distribution, so the parameters can take different proportions. Another important aspect of the gradient descent rule is that it might be slow in terms of convergence for black-box objective functions. For this reason, the *natural gradient* [Ama98; AD98] is introduced, which modifies the gradient descent rule to be,

$$\theta_{j+1} = \theta_j + \eta\cdot\boldsymbol{F}^{-1}\nabla_\theta J(\theta)\,. \tag{3.19}$$

Here, $\boldsymbol{F}$ is the *Fisher information matrix* defined as,

$$\boldsymbol{F} = \int \pi(\boldsymbol{z}\,|\,\theta)\,\nabla_\theta \log\pi(\boldsymbol{z}\,|\,\theta)\,\nabla_\theta \log\pi(\boldsymbol{z}\,|\,\theta)^{\top}\,d\boldsymbol{z}\,, \tag{3.20}$$

but in practice it is estimated as,

$$\boldsymbol{F} \approx \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log\pi(x_k\,|\,\theta)\,\nabla_\theta \log\pi(x_k\,|\,\theta)^{\top}\,. \tag{3.21}$$

In summary, NES attempts to estimate the search distribution using the natural gradient such that, when sampled, the search distribution will in average output the minimizer to the objective function. Here, most of the details have been omitted for the sake of brevity, but the original work [Wie+14] provides all the information needed to implement the method. In this work, though, a variant of the NES method is used, the *distance-weighted exponentital* (DXNES) NES [Fuk+11; Nom+21]. This method is a variation on the way the natural gradient and Fisher information matrix are obtained, but the core of the method remains the same.

Chapter 4

Neural networks as an approximation for the bridge function

Neural networks can be used as *universal approximators*, i.e., they can take the form of any continuous function if and only if the conditions of the universal approximation theorem hold (see subsection 3.4.4). The aim of this chapter is to explore the hypothesis that a neural network might be useful as a bridge function parametrization in the closure expression for the Ornstein-Zernike equation [HM13]. If this is true, then choosing a particular approximation can be avoided for a given interaction potential, and leave the choice of the bridge function to the neural network itself, while simultaneously solving for the Ornstein-Zernike equation. It is intended to explore the implications of two inquiries:

(a) Is it possible to solve the Ornstein-Zernike equation using a neural network?

(b) If it is indeed possible, how good is the quality of the approximation for the pseudo hard sphere potential [Báe+18]?

In this chapter, we show in detail the methodology created to answer these questions, and the mathematical structure with which a neural network can be used to solve the Ornstein-Zernike equation. The results obtained are compared to those from Monte Carlo computer simulations to assess the quality of the solution. In the appendices (Appendix A, Appendix B), the numerical algorithm used to solve the Ornstein-Zernike equation is presented, along with a detailed computation of the gradients used for the training scheme. Here, we shall focus only on the main results and the algorithm structure in general.

4.1 Parametrization of the bridge function

The Ornstein-Zernike formalism is given by the following coupled equations [HM13]

$$\begin{aligned} h(\boldsymbol{r}) &= c(\boldsymbol{r}) + n\int_V c(\boldsymbol{r}')h(|\boldsymbol{r}-\boldsymbol{r}'|)d\boldsymbol{r}' \\ c(\boldsymbol{r}) &= \exp\left[-\beta u(\boldsymbol{r}) + \gamma(\boldsymbol{r}) + B(\boldsymbol{r})\right] - \gamma(\boldsymbol{r}) - 1, \end{aligned} \tag{4.1}$$

with the already known notation for each quantity (Ref a marco teórico).

Let $N_\theta(\boldsymbol{r})$ be a neural network with weights θ. The main hypothesis of this chapter is that $N_\theta(\boldsymbol{r})$ can replace the bridge function $B(\boldsymbol{r})$ in the previous equation, which will yield the following expression for the closure relation

$$c(\boldsymbol{r}) = \exp\left[-\beta u(\boldsymbol{r}) + \gamma(\boldsymbol{r}) + N_\theta(\boldsymbol{r})\right] - \gamma(\boldsymbol{r}) - 1. \tag{4.2}$$

With this new expression, the main problem to solve is to find the weights of $N_\theta(\boldsymbol{r})$ that can successfully solve the Ornstein-Zernike equation for a given interaction potential, $\beta u(\boldsymbol{r})$.

4.2 Training scheme

Now that a parametrization is defined, a way to fit the weights of the neural network must be devised. This new numerical scheme must also be able to solve the OZ equation, while simultaneously finding the appropiate weights for $N_\theta(\boldsymbol{r})$.

4.2.1 Cost function

It was mentioned previously that the main problem is to find the weights of $N_\theta(\boldsymbol{r})$ that can successfully solve the Ornstein-Zernike equation for a given interaction potential. To solve such a problem, a **cost function** must be defined, and be used as part of a *minimization* problem.

To define such a function, we consider the successive approximations obtained from the iterative Piccard scheme to solve the OZ equation, $\{\gamma_1(\boldsymbol{r}), \gamma_2(\boldsymbol{r}), \ldots, \gamma_n(\boldsymbol{r})\}$. From this, we expect to have found a solution when each approximation is *close enough* to the previous one. This can be translated into the following cost function

$$J(\theta) = \left[\gamma_n(\boldsymbol{r},\theta) - \gamma_{n-1}(\boldsymbol{r},\theta)\right]^2, \tag{4.3}$$

where $\gamma_n(\boldsymbol{r},\theta)$ is the n-th approximation of the indirect correlation function, $\gamma(\boldsymbol{r})$. The notation $\gamma(\boldsymbol{r},\theta)$ indicates that the function now depends on the weights of the neural network, as seen in Equation 4.2. This means that, if the weights of $N_\theta(\boldsymbol{r})$ change, we should expect a change in the output from the γ function.

Another way of looking at Equation 4.3 is that we require that the last two approximations of the γ function in each iteration from the numerical scheme to be equal within a tolerance value, or upper bound. This will enforce a change on the weights every time both approximations deviate between them.

4.2.2 Optimization problem

With a cost function at hand, an optimization problem can be defined such that the weights of $N_\theta(\boldsymbol{r})$ will be adjusted properly.

This optimization problem is in fact an *unconstrained optimization problem*, and it is defined simply as

$$\min_{\theta\in\mathbb{R}^n} \quad J(\theta). \tag{4.4}$$

This formulation is just a search for the best values of the weights that minimize the squared difference between successive approximations. We denote these optimal values as the *minimizer*, or θ^*, such that $J(\theta^*)$ is a minimum. This optimization problem can be solved iteratively, along with the solution of the OZ equation, whose procedure to get a solution is also through an iterative process.

4.2.3 Weight updates

The iterative method employed to adjust the weights of $N_\theta(\boldsymbol{r})$ is based on the *gradient descent* method [NW06]. The most general update rule for a method based on gradient descent

reads [GBC16]

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta), \tag{4.5}$$

where η is known as the *learning rate*, and it is a hyperparameter that controls the step size at each iteration while moving toward the minimum of a cost function. This value needs to be *tuned* accordingly, so that the method converges properly. By tuning the parameter we mean that this value should change until the numerical scheme is stable enough, or provides the best possible answer to the optimization problem.

Regardless the particular expression for the weight updates, every method based on the gradient descent method *requires* the gradient information from the cost function with respect to the weights, $\nabla_\theta J(\theta)$. In this particular case, the detailed computation of the gradient is described in the Appendix A. Once this information is obtained, all that is left is to build an algorithm that can correctly use this training scheme and solve the OZ equation.

4.2.4 Solving the Ornstein-Zernike equation with neural networks

Having described all the necessary elements needed, a general layout for the solution of the Ornstein-Zernike using neural networks is now presented.

Thus, we propose the following steps to solve the OZ equation using the parametrization described by Equation 4.2:

1. Given a particular interaction potential $\beta u(\boldsymbol{r})$, Equation 4.2 is used to obtain the value of the direct correlation function $c(\boldsymbol{r})$. In this step, an initial value for $\gamma_n(\boldsymbol{r})$ is needed, which is initialized based on the five-point Ng methodology shown in Appendix B. The weights of the neural network $N_\theta(\boldsymbol{r})$ are initialized randomly. The initialization method is discussed in detail in the next section.

2. The newly found function $c(\boldsymbol{r})$ is transformed to a reciprocal space by means of the Fourier transform yielding the new function $\hat{c}(\boldsymbol{k})$.

3. Then, the full OZ equation(Ref a ec marco teórico) is Fourier transformed. Using the information from the previous step, a new estimation of the indirect correlation function is obtained, $\hat{\gamma}_{n+1}(\boldsymbol{k},\theta)$.

4. The Fourier transform is applied once again to return all the functions to real space. With this operation, a new estimation $\gamma_{n+1}(\boldsymbol{r},\theta)$ is computed from the transformed function, $\hat{\gamma}_{n+1}(\boldsymbol{k},\theta)$.

5. Both estimations, γ_n and γ_{n+1}, are used to evaluate Equation 4.3. In this step, the gradient $\nabla_\theta J(\theta)$ is computed as well.

6. The weights θ are updated using a gradient descent rule, similar to Equation 4.5, and the process is repeated from step 1. In the next iteration, the initial value for the indirect correlation function will be γ_{n+1}, and a new estimation γ_{n+2} will be obtained. This process is repeated until convergence.

4.2.5 Convergence criterion

The procedure described in the previous section is repeated indefinetely until convergence is achieved. This convergence criterion is defined as follows

$$\sum_{i=1}^{N} \left(\gamma_i^{n+1} - \gamma_i^n\right)^2 \leq \epsilon. \tag{4.6}$$

This expression is also known as the *mean squared error* [GBC16]. Here, we sum all the N elements of the squared difference between estimates γ_{n+1} and γ_n. The paramater ϵ is a tolerance value that indicates an upper bound for the error between estimations. When the computed error is below this tolerance value, we consider the algorithm to *have converged to a particular minimum*. This means that the weights are adjusted until the successive estimations of the γ functions are equal between them, up to the defined tolerance ϵ. Specifically, the numerical tolerance in all the experiments was fixed to be $\epsilon = 1 \times 10^{-5}$ to allow for a robust exploration of the search space, without being too restrictive. When using a lower value of ϵ, it was observed that the results were not improving at all (data not shown).

4.3 Implementation

In this section we detail the most important aspects about the implementation of the method described in the previous section. This includes the topology of the neural network, the optimization method, and the choice of activation function. The physical parameters as well as the computer simulations methods used to solve the OZ equation are also outlined.

4.3.1 Choice of optimization algorithm

The general rule for the weight update based on Equation 4.5 was implemented to solve the optimization problem, but numerical inconsistencies rendered this method unstable and convergence was almost never achieved.

To solve this issue, the *Adam* [KB17] optimization method was chosen. This optimization method is an excellent choice for the training of neural networks, even more when the gradient is expected to be *sparse*, i.e. most of the elements of the gradient itself are zeros. The *Adam* method uses several rules to adjust the descent direction of the gradient, as well as the hyperparameters related to the acceleration mechanism of the method. Notably, there are two important hyperparameters used in the method; β_1, which controls the moving average of the computed gradient; and β_2, which controls the value of the gradient squared [KB17]. Both parameters are necessary for the optimal convergence of the algorithm.

The equations that define the optimization method are the following

$$\begin{aligned}
m_t &= \beta_1 m_{t-1} - (1-\beta_1)\nabla_{\theta_{t-1}} J(\theta_{t-1}) \\
s_t &= \beta_2 s_{t-1} + (1-\beta_2)\nabla_{\theta_{t-1}} J(\theta_{t-1}) \odot \nabla_{\theta_{t-1}} J(\theta_{t-1}) \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{s}_t &= \frac{s_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} + \eta \hat{m}_t \oslash \sqrt{\hat{s}_t + \varepsilon}
\end{aligned} \tag{4.7}$$

where $\odot$ is the elementwise multiplication, or Hadamard product; $\oslash$ is the elementwise division, or Hadamard division; and ε is a smoothing value to prevent division by zero [HJ12]. The index t represents each of the updates, or iterations, done by the algorithm.

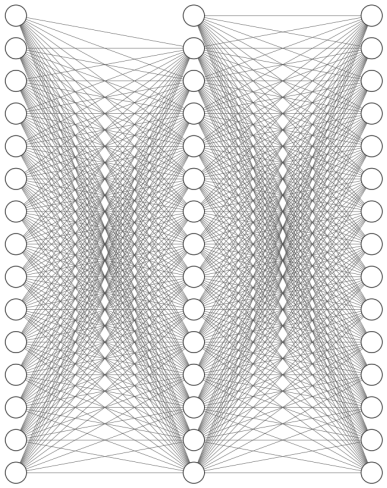

FIGURE 4.1: Cartoon of a fully connected multilayer neural network. Note that there is one *hidden layer*. The circles represent the *nodes* or *units* used to compute the final output. These nodes are being evaluated by an activation function to account for nonlinearities. The top-most nodes that seem different from the main nodes are known as the *bias* nodes. The real topology used in this chapter is larger, with many more nodes and connections, but the topology is the same.

In the results presented in this chapter, the parameters were fixed to the ones reported as optimal in the original work of the *Adam* method [KB17], which are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. It is important to note that this method has its own mechanisms to control and modify the gradients, as well as the hyperparameters. This makes it a *hands-off* method, without the need to tune the hyperparameters. The *learning rate*, η in Equation 4.5, was fixed to $\eta = 1 \times 10^{-4}$ for all the experiments because when a larger value was used the optimization method did not converge properly, and the results were not useful to be presented here. In a more practical situation, the best way to choose the value of η is to employ *grid search* and look for the value that minimizes the error the most [HTF09].

4.3.2 Neural network topology

The neural network topology used in all the experiments is identical to the one shown in Figure 4.1, with the exception of the number of nodes in each layer. In particular, the neural network is made of *three layers* fully connected between them. There is an *input* layer, one *hidden* layer, and a final *output* layer. All layers have the same number of nodes, which is 4096. Additional nodes are added to the final two layers that serve as the *bias* terms.

All the weights must be initialized appropiately, and in this case the Glorot uniform distribution was used [GB10], which has proven to be an excellent way to help the convergence of neural networks. When using the Glorot uniform distribution, the weights are initialized as $\theta_{ij} \sim \mathcal{U}\left[-\frac{6}{\sqrt{(in+out)}}, \frac{6}{\sqrt{(in+out)}}\right]$, where $\mathcal{U}$ is the uniform probability distribution; in represents the number of units in the input layer; and out the number of units in the output layer. All bias nodes were initialized to be zero.

The activation function used was the *ReLU* [GBB11] function, which has the form

$$\mathrm{ReLU}(x) = \max(0, x).$$

This activation function is applied to all the nodes in the layers, with the exception of the input layer. This function was chosen due to the fact that the other most common functions (tanh, softmax, etc.) were numerically unstable in the training process of the neural network (data not shown).

4.3.3 Physical parameters and simulations

To solve the OZ equation a cutoff radius of $r_c = 7\sigma$ was used, where σ is the particle diameter and it was fixed to be $\sigma = 1$. The interaction potential used was the pseudo hard sphere potential (Ref a ec.), both for the solution of the OZ equation, as well as the results obtained from computer simulations.

Seven different densities were explored in the range $\phi \in [0.15, 0.45]$, with $\Delta\phi = 0.05$. For each density value, a grid of 70 points was used to ensure convergence of the iterative algorithm when solving for the OZ equation. This was not the case for the computer simulations, where such partition is not needed.

Computer simulations results were obtained using the traditional Monte Carlo simulation method within the Metropolis scheme for the NVT ensemble (Ref a marco teórico). In every experiment, the total number of particles was 2197, the system was equilibrated for a total of 10 million Monte Carlo steps, and the radial distribution functions were obtained from the sampling of 7 million Monte Carlo steps, after the system was equilibrated. To reduce the number of computations, a cutoff radius of half the size of the simulation box was used for the evaluation of the interaction potential. Periodic boundary conditions in all spatial dimensions were used accordingly. The same pseudo hard sphere potential (Ref a ecuación) was used, instead of the true hard sphere potential, for a fair comparison with the results obtained from the OZ equation.

4.4 Results

It is now time to investigate the results obtained from the proposed methodology, using all the elements previously described. The main point of discussion will be the radial distribution function —$g(r^*)$ —for different values of densities, both in the low and high density regimes.

4.4.1 Low densities

In this section we will deal with the low density values ranging from $\phi = 0.15$ to 0.25, which are shown in Figure 4.2 and Figure 4.3, respectively. The results show that, at low densities, the HNC and neural network approximations are more precise than the modified Verlet approximation. Although at first glance, all approximations seem to fall short compared to computer simulations. This is particularly noticeable in the neighborhood around the second peak, which is shown in the insets of Figure 4.2 and Figure 4.3. Additionally, it is important to note that the neural network approximation is slightly more precise than the HNC approximation, which can be qualitatively appraised by observing the estimation of the main peak in the radial distribution function. This peak can be found in the vicinity of $r^* = 1$. Nevertheless, it is still overestimated, which is the same case for the HNC approximation. However, this is not the case for the modified Verlet approximation, which underestimates the main peak.

In like manner, the functional form of $g(r^*)$ is important to study closely. For the HNC and neural network approximations, it appears to have the same form between both approximations, and it might as well be the same. This would imply that, somehow, the weights of the neural

network were updated enough such that a minimum was found, and this minimum was very close to the HNC approximation. In other words, the results suggest that the weights are very close to zero, such that when the neural network is evaluated, the output is close to the result obtained from the HNC approximation. Another important aspect to observe is that this functional form is marginally different to the one seen from computer simulations, and that the modified Verlet approximation is closer to the form found in the computer simulations results.

4.4.2 High densities

We now turn our attention to the high density values, namely, $\phi = 0.35$ and 0.45, represented in Figure 4.4 and 4.5. In the same spirit as before with the low densities, the HNC and neural network approximations are not precise when compared to computer simulations. In this case, the modified Verlet bridge function approximation is even more precise, which was expected. This is because the HNC approximation is a very good approximation for long range interaction potentials (Ref faltante), whereas the modified Verlet is better suited for short range potentials, such as the one studied here. In this case, modified Verlet is the most precise of the approximations used, which can be inpsected in Figure 4.4 and Figure 4.5, where the main peak is accurately estimated by the approximation when compared to Monte Carlo computer simulation results. However, both HNC and neural network approximations overestimate this quantity.

Further, the functional form of $g(r^*)$ computed with the neural network approximation is substantially different to the one obtained with computer simulations. Indeed, the result obtained is similar to the one obtained with the HNC approximation, and both are imprecise approximations to the expected functional form; this was also the case for low densities. This result is important, backing the hypothesis that the neural network might reduce to the HNC approximation. This would imply that the neural network is in fact approximating the bridge function $B(\boldsymbol{r}) \approx 0$. If we now pay attention to the modified Verlet approximation, again in Figure 4.4 and Figure 4.5, we can see that the modified Verlet bridge function is the most precise out of all the set of bridge functions used. In other words, we observe that this estimation provides a precise prediction of the main peak, as can be seen when compared to the results obtained from computer simulations, which are almost identical.

4.5 Discussion

It would seem as though the neural network approximation reduces to the HNC approximation, as seen in the results from the previous section. In this section we shall investigate this matter in detail. We will also continue the discussion of the results presented and try to make sense of the training dynamics of the neural network. This is an important topic to address due to the clear results that the neural network provides almost the same result as the HNC approximation.

4.5.1 Weight evolution of the neural network

We shall now examine the evolution of the weights θ from $N_\theta(\boldsymbol{r})$, from the moment it was initialized to the moment its training finalized. A histogram of this for the density values $\phi = 0.15, 0.25, 0.35$ and 0.45 can be seen in Figure 4.6, Figure 4.7, Figure 4.8, and Figure 4.9, respectively. We can observe that the way the weights show a diagonal represent a linear relationship between the initial weights, θ_i, and the trained weights, θ_t. In other words, the weights follow the linear expression $\theta_t = \alpha\theta_i + \beta + \epsilon$, with $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$ a normal random variable with

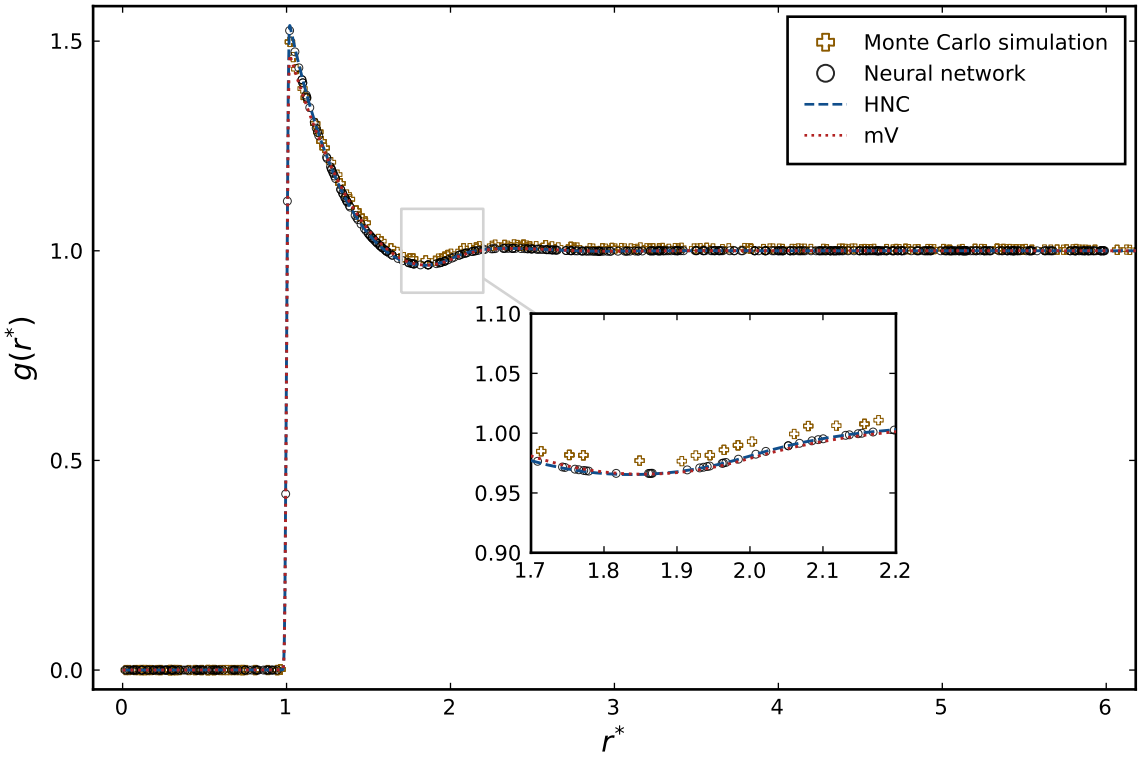

FIGURE 4.2: Radial distribution function for $\phi = 0.15$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

mean μ and variance σ^2. The noise term can be any other continuous probability distribution, but without loss of generality the normal distribution was chosen for our purposes. For now, we are not interested in the values of α or β, but merely on the linear relationship between them.

One thing to notice is the fact that the higher the density value is, the larger the variance turns out to be. If we observe the variance for the density $\phi = 0.15$ in Figure 4.6 we see that the variance is small due to the fact that the blue shaded region around the diagonal is close to it. If we now see the same Figure 4.9 for the density value of $\phi = 0.45$ we observe that this shaded region is significantly larger. This would mean that, at higher densities, the weights of $N_\theta(\boldsymbol{r})$ are more spread out from the mean, and the neural network might have adjusted its weights to account for different computations of the bridge function.

The most interesting part of this is the fact that the weights from initialization do not change much throughout the training scheme, which would imply that a local minimum has already been found. This might be the case, because HNC is actually a solution of the OZ equation, and solutions around this particular approximation might as well be solutions themselves. This, however, does not answer the question of why the spread is larger when higher densities are inspected.

4.5.2 The Hypernetted Chain approximation as a stable minimum

It would seem that the way the weights are updated, albeit with minimal change from its initial values, is due to the fact of already being near a minimum when the training starts. We must recall

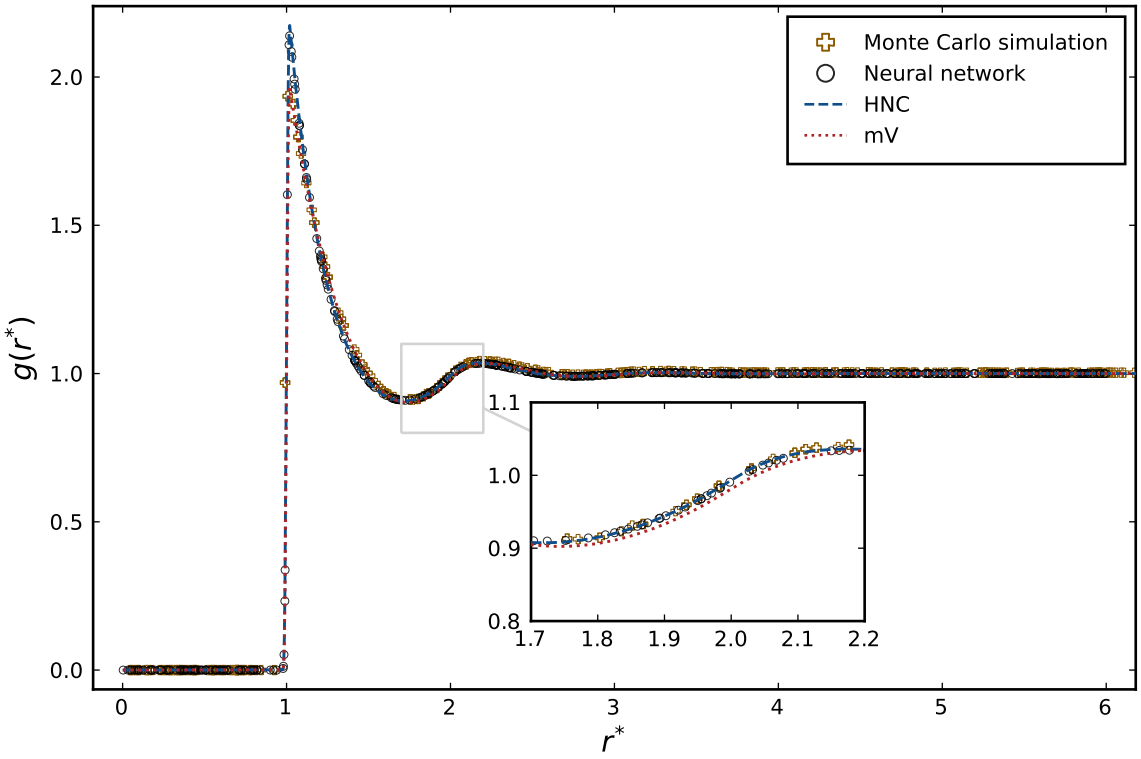

FIGURE 4.3: Radial distribution function for $\phi = 0.25$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

that the weight update and neural network training is essentially an optimization problem, and the main goal is to find a minimum of the cost function in Equation 4.3. With the results presented so far, it might be possible to postulate that the *HNC approximation is a stable minimum* for the neural network $N_\theta(\boldsymbol{r})$. This would answer the question of why the weights of the neural network during training explored in the previous section did not change very much throughout the numerical scheme. Because if we have already found a minimum, the optimization algorithm might end up oscillating in the proximity of this value.

On the other hand, this idea could also give answer to the question of why the spread is larger for higher density values. If we pay close attention to the neural network bridge approximation results for the *low density* values in Figure 4.2, we can see that although all the bridge functions give a low accuracy estimation of the second peak as shown in the inset within the figure. However, for the main peak the neural network approximation is accurate. If we now observe Figure 4.5, which refers to the *high density* value, we can see that the estimation is a poor one.

Let us now relate this to the weight evolution. For the *low density* regime, the weight evolution has a *lower variance*; for the *high density* regime, a *higher variance* is observed in the weight evolution. This suggests that, for *lower density* values, there was no need to adjust the weights more than shown in Figure 4.6 because the approximation is accurate enough. However, for the *higher density* values, the approximation is not good enough and the optimization method was trying to adjust the weights accordingly, even if unsuccessfully. Thus, by oscillating near the value of zero, which represents the HNC bridge function, the neural network does not need additional

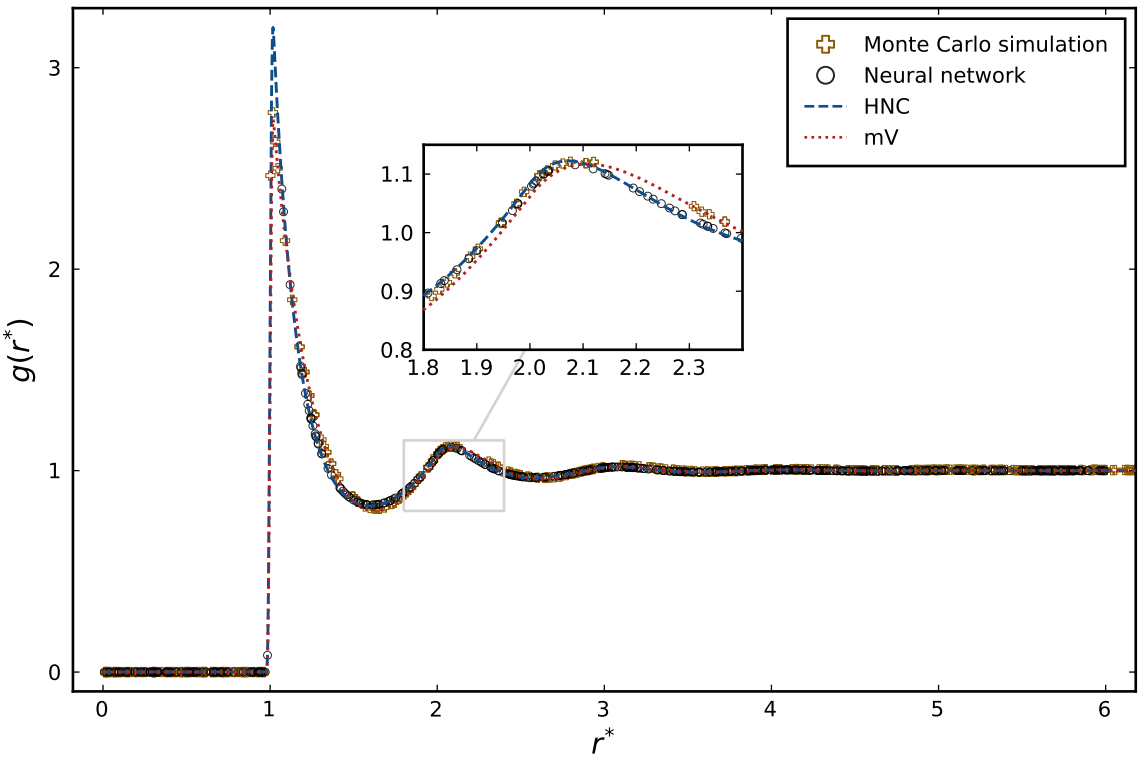

FIGURE 4.4: Radial distribution function for $\phi = 0.35$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

information and generates a bridge function approximation that reproduces the results from the HNC closure relation.

Having a stable minimum when training starts would mean that the neural network does not learn enough, and will alway keep its weights tightly centered about the mean of this minimum. Still, this implies that other minima are available for the neural network as long as the weights are correctly initialized, or a probability distribution centered about a particular minima is used.

4.5.3 Does the neural network reduce to HNC?

For the low density regimes, HNC is an accurate approximation for the interaction potential. Hence, the neural network is an accurate approximation. On the contrary, for high density regimes, both approximations fail to provide an accurate solution.

If the neural network is in indeed oscillating about zero (the HNC approximation), then it makes sense that both estimations give the results observed. Yet, we cannot guarantee by any means possible that the neural network reduces to the HNC approximation. We only possess *statistical evidence* from the training dynamics that the neural network weights do not change much throughout its training.

This observation might shed light into possibilities of changing the way the neural network propagates its values and return an output. For example, a modification to the neural network topology might be in order, such that introducing important nonlinearities that are consistent with the physical properties of the system can yield better results. For the case of hard spheres,

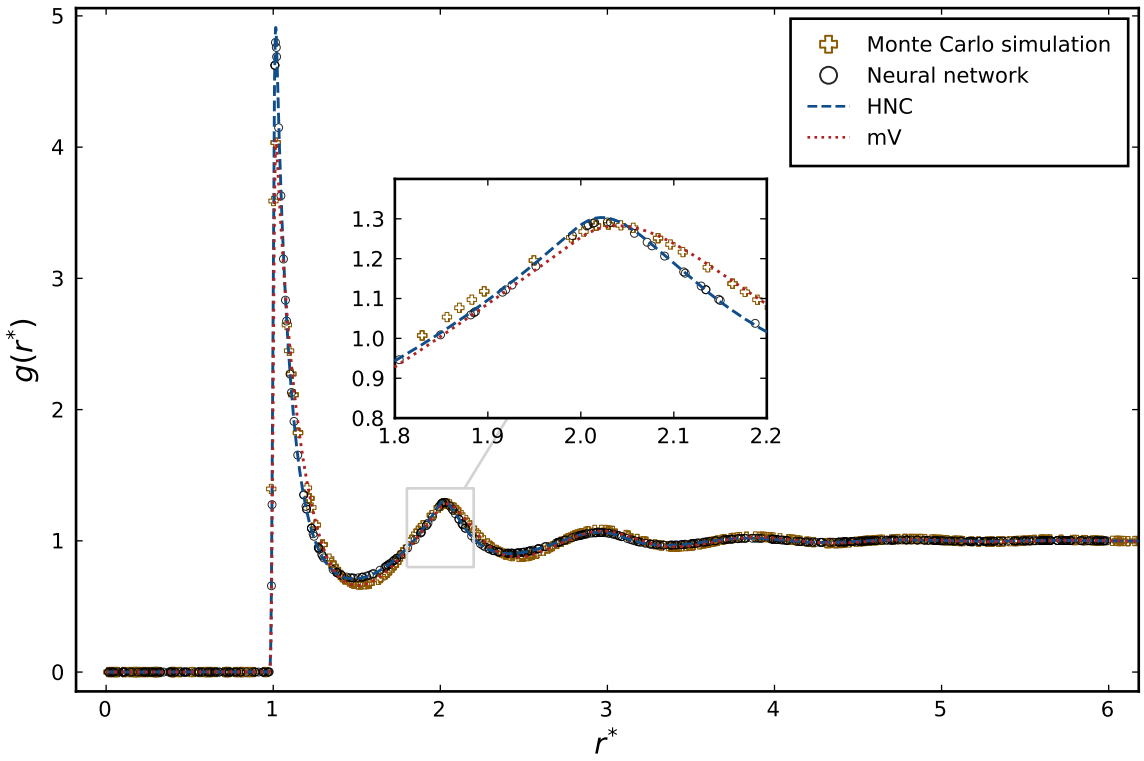

FIGURE 4.5: Radial distribution function for $\phi = 0.45$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

the work by Malijevský and Labík [ML87] shows that the bridge function has particular functional properties, such that the bridge function is non-negative and oscillating, among others. These properties can then be consolidated within the neural network structure and investigate if the neural network weights change considerably. From this, one might expect two outcomes. Firstly, the case where the *weights change*, which would indicate that adding nonlinearities according to some aspect of the interaction potential prove benefitial for the weight update and training dynamics. Secondly, the case where the *weights do not change*, in which case we might be able the have stronger evidence that, regardless of the neural network structure, this kind of approximation will have a high chance of reproducing the HNC results. In any case, with either outcome we do not have the information to assess if the neural network might actually be *more precise* than it is in its current form.

Similar results were found by Goodall and Lee [AGAL21] while using data from simulations. In this approach, a data set was built with several correlation functions that came from physical properties of the liquid. If the data set was built just using the indirect correlation function $\gamma(\boldsymbol{r})$, the neural network trained from this data set would yield results similar or worse to those obtained with the HNC approximation. So, in some sense, the proposed methodology here might actually be better than a fully data-driven methodology. However, this just makes a stronger case for the argument that, indeed, the neural network might reduce to the HNC approximation and not have enough information about the system to adjust the weights properly.

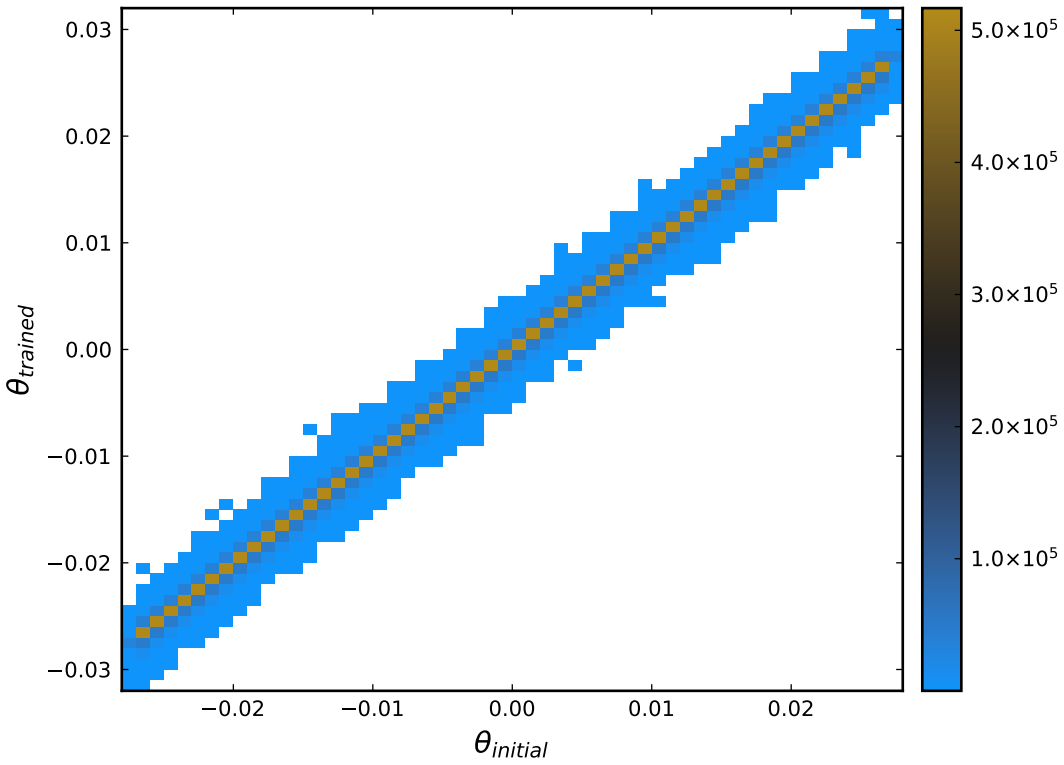

FIGURE 4.6: Relation between the trained weights and the initial weights of $N_\theta(\boldsymbol{r})$ for $\phi = 0.15$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

4.5.4 Neural networks as random polynomial approximations

An interesting result from this investigation is the fact that a random approximation provides a solution to the OZ equation. In what way is it random? Take the weights of the neural network to be the coefficients of some *random polynomial*, i.e. a polynomial with coefficients that come from some probability distribution P. This is something that is not trivial on why it could work as a bridge function approximation, even if the result is very close to the HNC approximation. This would imply that, for the probability distribution P with some finite variance σ^2 and mean $\mu = 0$, the resulting polynomial would yield a bridge function estimation similar to the HNC closure relation. Yet, the reason why a random approximation might be a solution to the OZ equation is not clear.

To understand the significance of this, we must recall that the bridge function can, in fact, be understood as a power series in density [HM13],

$$b(\boldsymbol{r}) = b^{(2)}(\boldsymbol{r})\rho^2 + b^{(3)}(\boldsymbol{r})\rho^3 + \cdots, \tag{4.8}$$

where the notation $b^{(n)}(\boldsymbol{r})$ indicates the n-particle bridge function, i.e. the estimation of the bridge function for n interacting particles. It is specially important to note that the coefficients $b^{(n)}(\boldsymbol{r})$ are, in general, high-dimensional integrals that are mostly *intractable*. Almost always, for a given bridge function approximation, numerical methods are in order if these coefficients are to be determined. For instance, the work by Kwak and Kofke [KK05] uses a Monte Carlo sampling numerical method to evaluate up to $b^{(4)}(\boldsymbol{r})$ for the hard-sphere fluid. They report that even if the

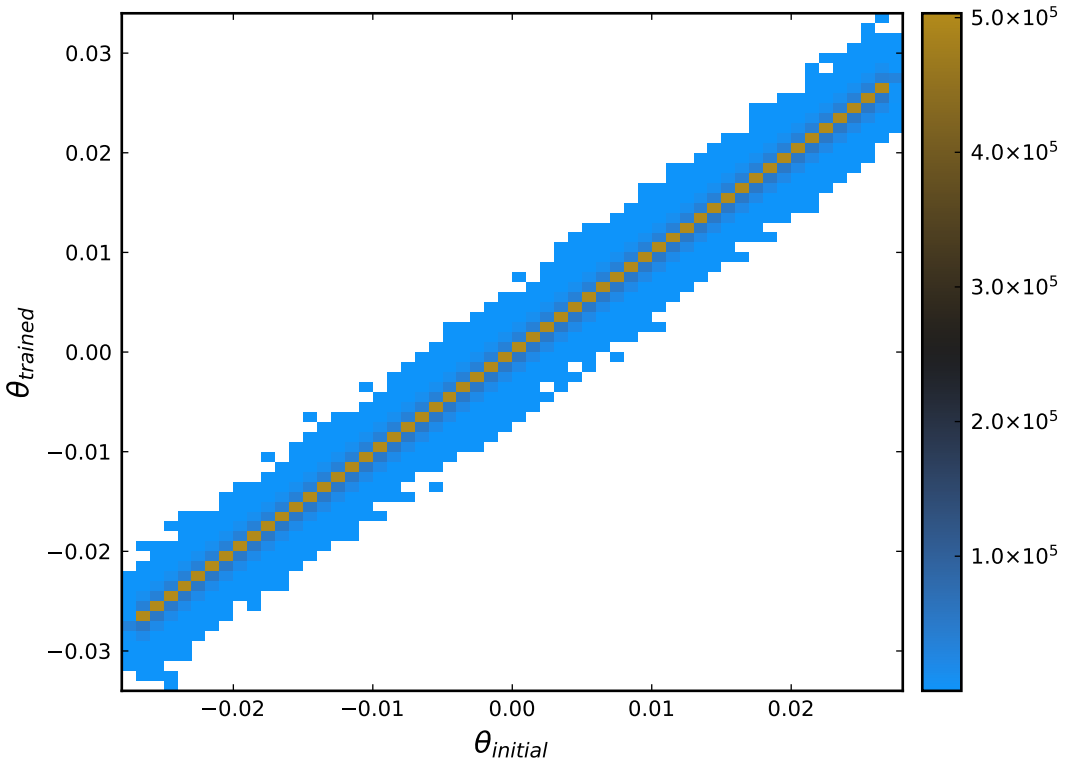

FIGURE 4.7: Relation between the trained weights and the initial weights of $N_\theta(\boldsymbol{r})$ for $\phi = 0.25$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

numerical computation is possible, convergence is slow and computationally costly.

Only for special cases, these coefficients can be determined in closed form, e.g. for the Percus-Yevick approximation the value of

$$b^{(2)}(\boldsymbol{r}) = -\frac{1}{2}\left[g_1(\boldsymbol{r})\right]^2$$

is known from diagrammatic methods [HM13]. In this expression, $g_1(\boldsymbol{r})$ represents the single-particle radial distribution function.

Let us now understand the role of *random polynomials* and their relation to the neural network bridge function approximation used in this investigation. Let p_n be an algebraic polynomial of the form

$$p_n(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots + a_n z^n, \quad z \in \mathbb{C} \tag{4.9}$$

where the coefficients $a_0, a_1, \ldots, a_n$ are independent real-valued random variables with finite mean and finite variance. This kind of polynomials have found successful applications in some areas of physics [Hou+09]. These polynomials have also been the interest of mathematical research [EK95].

Now, by means of the universal approximation theorem, we can regard the neural network as a power series similar to Equation 4.9. To see this more clearly, we can think of the weights of the neural network as some coefficients of a power series, obtained under some transformation that takes in the weights and return the coefficients needed to build a power series. Further, if we compare directly Equation 4.9 and Equation 4.8, we can see that these expressions are

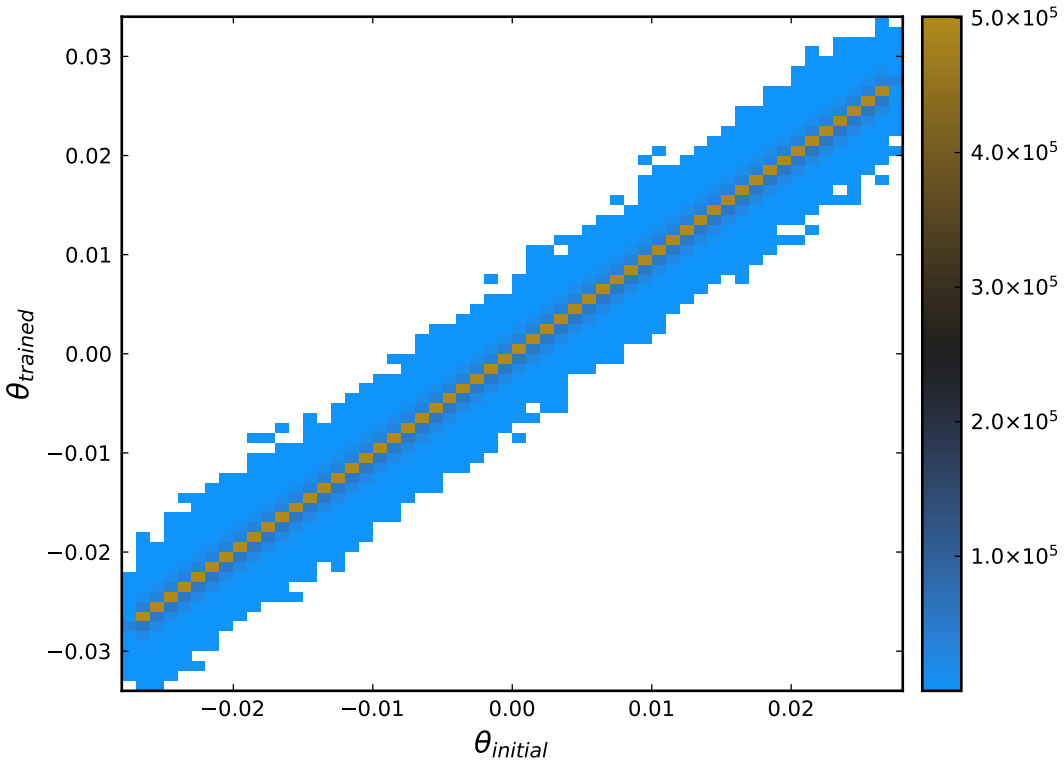

FIGURE 4.8: Relation between the trained weights and the initial weights of $N_\theta(\boldsymbol{r})$ for $\phi = 0.35$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

related through their coefficients, with the exception of the first two terms, implying that random variables might be able to give an answer to the n particle bridge functions in the power series.

Indeed, this is a convenient way of estimating the coefficients of the bridge function, when expressed in a power series of the density or any other correlation functions. Although, the practical way to estimate these coefficients is to enforce thermodynamic self-consistency. Such is the case of the work by Vompe and Martynov [VM94], and the work by Tsednee and Luchko [TL19], where the coefficients are found through a minimization problem when the thermodynamic consistency among different routes has been achieved.

After all, the insight of random polynomials might pave the way for a novel way of computing coefficients by using neural networks, or related probabilistic methods. Even though there is no way to enforce thermodynamic self-consistency when using such methods, it is interesting to see the striking resemblance with these approaches. And yet, one of the downsides of this is the fact that the probability distribution for the coefficients $a_0, a_1, \ldots, a_n$ cannot be known even through the training dynamics of the neural network. A naive way of acquiring the underlying probability distribution is to suppose there is a unique probability distribution and estimate its mean and variance by maximum likelihood estimation techniques [HTF09]. Still, even if there could be a relation between random polynomials and the bridge function, there can be no guarantee that the resulting approximation is good enough, or that it could reproduce the physics of the problem properly. This is just mere speculation that a deeper relationship between the neural network approximation and the bridge function might be exploitable through the lens of random polynomials and probability theory.

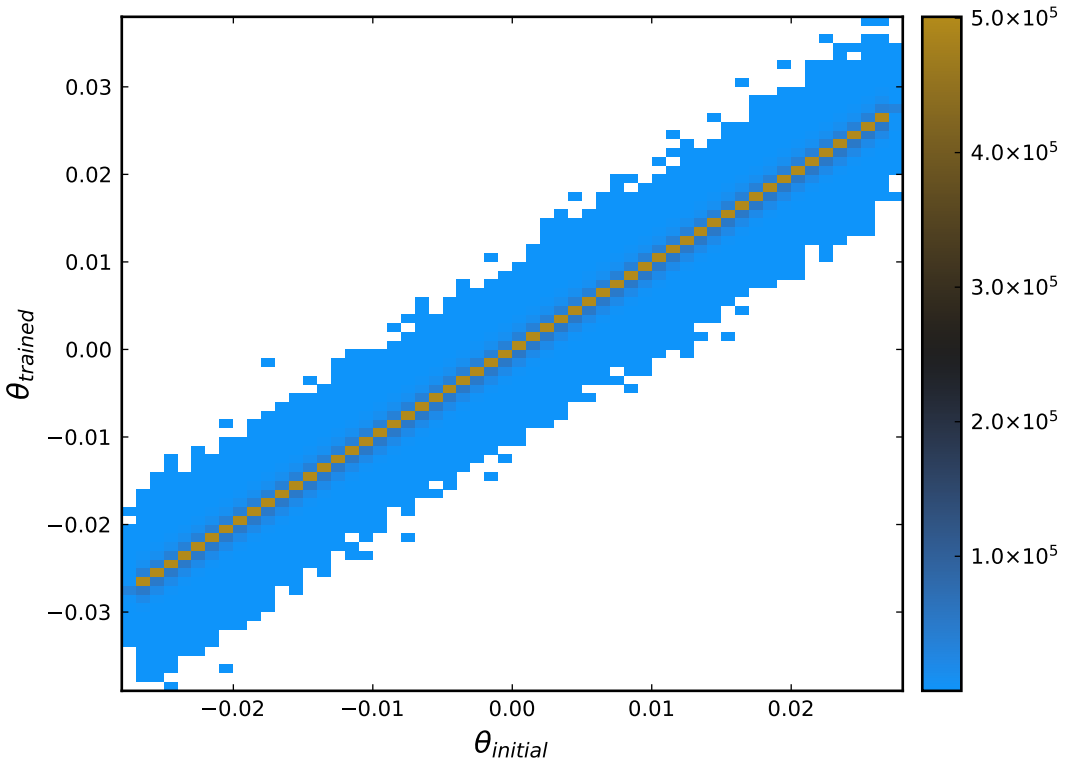

FIGURE 4.9: Relation between the trained weights and the initial weights of $N_\theta(\boldsymbol{r})$ for $\phi = 0.45$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

4.6 Concluding remarks

Even though neural networks can approximate any continuous function, it is up to the implementation that uses them to benefit from the underlying domain knowledge of the problem to be solved. In this case, even if the methodology created is *theoretically possible*, it raises the question of whether the neural networks are the most suitable solution for the OZ equation. Such might be the case for the methodology presented here. After all, the results seem to *strongly suggest* that a neural network reduces to one of the classic approximations, the Hypernetted Chain approximation, which is not the best approximation for all kinds of interaction potentials, and most certainly is not the case of the pseudo hard sphere and hard sphere potentials. At any rate, these results show a promising application of such models, in the form of understanding how or why certain bridge function approximations provide a solution to the OZ equation. Unlike the use of fully data-driven methodologies, like the one from Goodall and Lee [AGAL21], the proposed methodology might somehow be able to provide intuition into what is happening with the bridge function itself without the need of other theoretical methods.

One of the main drawbacks of the current proposal is the fact that the number of weights is too large to effectively train well. This might be a reason of why the spread was small in the weight evolution of $N_\theta(\boldsymbol{r})$. A way to alleviate this problem is to use dimensionality reduction techniques such as Principal Component Analysis [HTF09]. These methods can adequately reduce the total number of weights to be trained, without losing too much information from the learning dynamics.

In order to use the physics of the problem to our advantage, a different optimization problem might be formulated, in which case a thermodynamic consistency cost function might be able to drive the learning dynamics. In this framework, it is expected to minimize the difference between two different routes that compute the pressure, or the isothermal compressibility, similar to the approaches by Zerah and Hansen [ZH86], as well as Rogers and Young [RY84]. A deficiency of such scheme is that the cost function will be a highly nonlinear function, and possibly a *black-box function*, in which case the gradient of the function might be hard or even impossible to compute in a timely manner. This affects not only the computation time but the ability to use gradient descent methods, or any other optimization method that uses gradients to find an extremum. Nevertheless, such approach could be a dramatic improvement over the methodology presented here, even more so if coupled with a dimensionality reduction technique.

In closing, neural networks are capable models for the approximation of any continuous function, but domain knowledge of the problem is needed for these models to succeed when no data set is available. In the proposal presented in this chapter, we wanted to investigate two things, whether neural networks could solve the OZ equation and their quality of approximation. It was shown that, indeed, neural networks can solve the OZ equation, but without more information on the system itself, neural networks do not generalize well and their training dynamics suffer greatly. The bridge function approximation that these models provide is implied to be as accurate as the Hypernetted Chain bridge function. The information that these models gather throughout the training is minimal, but their training dynamics shed some light into the possibilities of using probability methods to approximate bridge functions in liquids.

Chapter 5

Evolutionary optimization for the Kinoshita closure

Some text…

Appendix A

Gradient Computation

In chapter 4 a training scheme was developed to adjust the weights of a neural network while simultaneously solving for the OZ equation. A crucial part of this algorithm is the *gradient computation*. In this section, we shall work out the details of this computation.

A.1 Mathematical development

Recall that we want a neural network $N_\theta(\boldsymbol{r})$ with weights θ to work as a parametrization in the closure expression of the OZ equation, as defined in Equation 4.2. To find the weights of the neural network, an unconstrained optimization problem was presented, which was meant to be solved with an iterative procedure known as *gradient descent*, which has the following general rule

$$\theta_{n+1} = \theta_n - \eta \nabla_{\theta_n} J(\theta_n)$$

where the cost function $J(\theta)$ is defined in Equation 4.3; ∇_θ is the gradient of $J(\theta)$ with respect to the weights, θ; and η is known as the learning rate, which controls the length of the step taken by the gradient towards a minimum. The iteration step is accounted for with the index n, which runs until convergence has been achieved.

We are now interested in the closed form of $\nabla_\theta J(\theta)$. If we use the definition of the cost function as defined in Equation 4.3, we have for the gradient the following expression

$$\nabla_\theta J(\theta) = \nabla_\theta \left[\gamma_n(\boldsymbol{r},\theta) - \gamma_{n-1}(\boldsymbol{r},\theta)\right]^2 \tag{A.1}$$

where $\gamma_n(\boldsymbol{r},\theta)$ is the n-th approximation of the indirect correlation function, $\gamma(\boldsymbol{r})$. The notation $\gamma(\boldsymbol{r},\theta)$ indicates that the function now depends on the weights of the neural network. When the weights θ are modified, the output of $\gamma(\boldsymbol{r},\theta)$ must change as well.

We now apply the gradient operation to Equation A.1 to obtain the following expression

$$\nabla_\theta J(\theta) = 2\left[\gamma_n(\boldsymbol{r},\theta) - \gamma_{n-1}(\boldsymbol{r},\theta)\right]\left[\partial_\theta \gamma_n(\boldsymbol{r},\theta) - \partial_\theta \gamma_{n-1}(\boldsymbol{r},\theta)\right] \tag{A.2}$$

which results from direct application of the chain rule. Now, an expression for $\partial_\theta \gamma_n(\boldsymbol{r},\theta)$ needs to be found by some other route. Notably, this expression should be expressed in terms of a quantity that *depends on the weights*, θ. In this case, this would mean that we seek some expression in terms of the neural network $N_\theta(\boldsymbol{r})$, which has a direct dependency on the weights.

In order to find this new expression, we invoke Equation 4.2 and differentiate it with respect to the weights

$$\frac{\partial c(\boldsymbol{r},\theta)}{\partial \theta} = \frac{\partial}{\partial \theta}\left[e^{p(\boldsymbol{r},\theta)} - \gamma(\boldsymbol{r},\theta) - 1\right] = e^{p(\boldsymbol{r},\theta)}\partial_\theta p(\boldsymbol{r},\theta) - \partial_\theta \gamma(\boldsymbol{r},\theta), \tag{A.3}$$

here we have for $p(\boldsymbol{r},\theta)$

$$p(\boldsymbol{r},\theta) = -\beta u(\boldsymbol{r}) + \gamma(\boldsymbol{r},\theta) + N_\theta(\boldsymbol{r}) \tag{A.4}$$

with derivative with respect to the weights

$$\partial_\theta p(\boldsymbol{r},\theta) = \partial_\theta \gamma(\boldsymbol{r},\theta) + \partial_\theta N_\theta(\boldsymbol{r}). \tag{A.5}$$

We have now found a closed form for the value of $\partial_\theta \gamma_n(\boldsymbol{r},\theta)$ which is essentially the same as Equation A.3, but in a slightly different form, which reads

$$\boxed{\frac{\partial \gamma(\boldsymbol{r},\theta)}{\partial \theta} = e^{p(\boldsymbol{r},\theta)} \partial_\theta p(\boldsymbol{r},\theta) - \partial_\theta c(\boldsymbol{r},\theta).} \tag{A.6}$$

Note, however, that this new expression depends on the value of $\partial_\theta c(\boldsymbol{r},\theta)$ which we do not readily have at this point. It is at this step that we propose to compute the value of $\partial_\theta c(\boldsymbol{r},\theta)$ using numerical differentiation, which should be enough to get a good estimate of the derivative.

A.1.1 General solution scheme

In order to carry out the detailed explanation of the numerical approximation approach, we must recall the order in which the general solution to the OZ equation is achieved. We need to do this in order to understand how the numerical approximation of the derivative for $c(\boldsymbol{r},\theta)$ can be obtained. This is already described in detail in subsection 4.2.4, but we need to recall it here briefly.

In short, the general solution to the OZ equation with a neural network parametrization looks like this:

- For the first part, we solve the OZ equation in a classical fashion, employing Fourier transforms in order to build a set of approximations for the $\gamma(\boldsymbol{r},\theta)$ function. At this step we have found also an approximation for $c(\boldsymbol{r},\theta)$, which is crucial to remember.
- When we have found said approximations, we compute the gradient as shown in Equation A.2. Due to the fact that we have found previous approximations to the correlation functions, we can use them as part of our gradient computation.
- Finally, with the value of the gradient, we compute the loss function and adjust the weights θ for the neural network. We then repeat the process until convergence is reached.

As we can see, given the fact that the OZ solution scheme already provides numerical approximations for the correlation functions, we can use the value of $c(\boldsymbol{r},\theta)$ in other numerical schemes to find the value of Equation A.6.

A.2 Numerical differentiation approach

We will now outline the scheme used in the current work to find the value of Equation A.6 using a numerical differentiation scheme for $\partial_\theta c(\boldsymbol{r},\theta)$.

To achieve such goal, we must briefly outline the method of numerical differentiation for functions of several variables [Ham12]. We wish to work with the $c(\boldsymbol{r},\theta)$ function, which is essentially a function of two variables. The first variable, $\boldsymbol{r}$ is the *Euclidean distance* between two

particles, or in other words $\boldsymbol{r} = |\boldsymbol{r}_1 - \boldsymbol{r}_2|^2$. For the second variable θ, this represents the weights of $N_\theta(\boldsymbol{r})$, which is in fact a vector of its own of size n. The value of n depends on the number of nodes in the neural network, but here we will use a more general approach instead of defining a specific value of n.

So, with this in mind, we can define the function to be $c(\boldsymbol{r},\theta) : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$. For a multivariable function, its partial derivative is computed numerically, for a particular value of distance $\bar{\boldsymbol{r}}$, as

$$\frac{\partial c(\bar{\boldsymbol{r}},\theta)}{\partial \theta} \approx \frac{c(\bar{\boldsymbol{r}},\theta+h) - c(\bar{\boldsymbol{r}},\theta)}{h} \tag{A.7}$$

with $h \in \mathbb{R}$ usually a small number. However, we already know the value of $c(\bar{\boldsymbol{r}},\theta)$ from the approximation obtained with the solution of the OZ equation. We need only the value of $c(\bar{\boldsymbol{r}},\theta+h)$ which can be easily obtained by modifying the weights of $N_\theta(\boldsymbol{r})$ by the value h and evaluating the closure relation

$$c(\bar{\boldsymbol{r}},\theta+h) = \exp\left[-\beta u(\bar{\boldsymbol{r}}) + \gamma(\bar{\boldsymbol{r}}) + N_{\theta+h}(\bar{\boldsymbol{r}})\right] - \gamma(\bar{\boldsymbol{r}}) - 1.$$

It is important to point out that all operations are elementwise.

From here, we simply compute the rest of Equation A.6 using the numerical approximation of $\partial_\theta c(\bar{\boldsymbol{r}},\theta)$ and we use this information to compute the value of the gradient in Equation A.2. We can then continue with the development of the closed form of this gradient. Putting all this information together we are able to find an expression for $\partial_\theta \gamma_n(\boldsymbol{r},\theta)$. We take Equation A.6 together with Equation A.5 to obtain the following expression

$$\frac{\partial c(\boldsymbol{r},\theta)}{\partial \theta} = e^{p(\boldsymbol{r},\theta)}\left[\partial_\theta \gamma(\boldsymbol{r},\theta) + \partial_\theta N_\theta(\boldsymbol{r})\right] - \partial_\theta \gamma(\boldsymbol{r},\theta) \tag{A.8}$$

and now to find an expression for $\partial_\theta \gamma(\boldsymbol{r},\theta)$ we perform the following steps:

$$\begin{aligned}
\partial_\theta c(\boldsymbol{r},\theta) &= e^{p(\boldsymbol{r},\theta)}\left[\partial_\theta \gamma(\boldsymbol{r},\theta) + \partial_\theta N_\theta(\boldsymbol{r})\right] - \partial_\theta \gamma(\boldsymbol{r},\theta) \\
\partial_\theta c(\boldsymbol{r},\theta) &= e^{p(\boldsymbol{r},\theta)}\partial_\theta \gamma(\boldsymbol{r},\theta) + e^{p(\boldsymbol{r},\theta)}\partial_\theta N_\theta(\boldsymbol{r}) - \partial_\theta \gamma(\boldsymbol{r},\theta) \\
\partial_\theta \gamma(\boldsymbol{r},\theta)\left[1 - e^{p(\boldsymbol{r},\theta)}\right] &= e^{p(\boldsymbol{r},\theta)}\partial_\theta N_\theta(\boldsymbol{r}) - \partial_\theta c(\boldsymbol{r},\theta)
\end{aligned}$$

$$\boxed{\partial_\theta \gamma(\boldsymbol{r},\theta) = \frac{e^{p(\boldsymbol{r},\theta)}\partial_\theta N_\theta(\boldsymbol{r}) - \partial_\theta c(\boldsymbol{r},\theta)}{1 - e^{p(\boldsymbol{r},\theta)}}} \tag{A.9a}$$

An equation for the derivative $\partial_\theta \gamma(\boldsymbol{r},\theta)$ has now been found which does satisfy the conditions of being dependent on the weights explicitly, and with the numerical approximation of $\partial_\theta c(\boldsymbol{r},\theta)$.

To finish up the gradient expression, take Equation A.2

$$\nabla_\theta J(\theta) = 2\left[\gamma_n(\boldsymbol{r},\theta) - \gamma_{n-1}(\boldsymbol{r},\theta)\right]\left[\partial_\theta \gamma_n(\boldsymbol{r},\theta) - \partial_\theta \gamma_{n-1}(\boldsymbol{r},\theta)\right],$$

and plug in Equation A.9a to obtain a closed form of the gradient we seek

$$\boxed{\nabla_\theta J(\theta) = 2\left[\gamma_n(\boldsymbol{r},\theta) - \gamma_{n-1}(\boldsymbol{r},\theta)\right]\left[\frac{e^{p_n(\boldsymbol{r},\theta)}\partial_\theta N_\theta(\boldsymbol{r}) - \partial_\theta c(\boldsymbol{r},\theta)}{1 - e^{p_n(\boldsymbol{r},\theta)}} - \frac{e^{p_{n-1}(\boldsymbol{r},\theta)}\partial_\theta N_\theta(\boldsymbol{r}) - \partial_\theta c(\boldsymbol{r},\theta)}{1 - e^{p_{n-1}(\boldsymbol{r},\theta)}}\right].} \tag{A.10}$$

In practice, a smoothing factor of $\varepsilon = 1 \times 10^{-7}$ was used in the denominator of the gradient expression from Equation A.10 to avoid division by zero. Again, the multiplication and division

in the same expression are elementwise operations.

Appendix B

Numerical solution to the OZ equation

Some equations.

Bibliography

[ACT19] Kai Arulkumaran, Antoine Cully, and Julian Togelius. "AlphaStar: An Evolutionary Computation Perspective". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. New York, NY, USA: Association for Computing Machinery, July 2019, pp. 314–315. ISBN: 978-1-4503-6748-6. DOI: 10.1145/3319619.3321894.

[AD98] S. Amari and S.C. Douglas. "Why Natural Gradient?" In: *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*. Vol. 2. May 1998, 1213–1216 vol.2. DOI: 10.1109/ICASSP.1998.675489.

[Ada+19] Stavros P. Adam et al. "No Free Lunch Theorem: A Review". In: *Approximation and Optimization : Algorithms, Complexity and Applications*. Ed. by Ioannis C. Demetriou and Panos M. Pardalos. Springer Optimization and Its Applications. Cham: Springer International Publishing, 2019, pp. 57–82. ISBN: 978-3-030-12767-1. DOI: 10.1007/978-3-030-12767-1_5.

[AGAL21] Rhys E. A. Goodall and Alpha A. Lee. "Data-Driven Approximations to the Bridge Function Yield Improved Closures for the Ornstein–Zernike Equation". In: *Soft Matter* 17.21 (2021), pp. 5393–5400. DOI: 10.1039/D1SM00402F.

[Ago+15] Forest Agostinelli et al. "Learning Activation Functions to Improve Deep Neural Networks". In: *arXiv:1412.6830 [cs, stat]* (Apr. 2015). arXiv: 1412.6830 [cs, stat].

[Ama98] Shun-ichi Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276. ISSN: 0899-7667. DOI: 10.1162/089976698300017746.

[APA20] Abolfazl Abdollahi, Biswajeet Pradhan, and Abdullah Alamri. "VNet: An End-to-End Fully Convolutional Neural Network for Road Extraction From High-Resolution Remote Sensing Data". In: *IEEE Access* 8 (2020), pp. 179424–179436. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3026658.

[Aps] "APS -APS March Meeting 2020 - Event - Inverse Design of Soft Materials: Crystals, Quasi Crystals, Liquid Crystals". In: *Bulletin of the American Physical Society*. Vol. Volume 65, Number 1. American Physical Society.

[AT17] Michael P. Allen and Dominic J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, Aug. 2017. ISBN: 978-0-19-252470-6.

[AW57] B. J. Alder and T. E. Wainwright. "Phase Transition for a Hard Sphere System". In: *The Journal of Chemical Physics* 27.5 (Nov. 1957), pp. 1208–1209. ISSN: 0021-9606. DOI: 10.1063/1.1743957.

[Báe+18] César Alejandro Báez et al. "Using the Second Virial Coefficient as Physical Criterion to Map the Hard-Sphere Potential onto a Continuous Potential". In: *The Journal of Chemical Physics* 149.16 (Oct. 2018), p. 164907. ISSN: 0021-9606. DOI: 10.1063/1.5049568.

[Bay+18] Atilim Gunes Baydin et al. “Automatic Differentiation in Machine Learning: A Survey”. In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43. ISSN: 1533-7928.

[BBTLB13] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. “Machine Learning Strategies for Time Series Forecasting”. In: *Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures*. Ed. by Marie-Aude Aufaure and Esteban Zimányi. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, 2013, pp. 62–77. ISBN: 978-3-642-36318-4. DOI: 10.1007/978-3-642-36318-4_3.

[BDF19] Emanuele Boattini, Marjolein Dijkstra, and Laura Filion. “Unsupervised Learning for Local Structure Detection in Colloidal Systems”. In: *The Journal of Chemical Physics* 151.15 (Oct. 2019), p. 154901. ISSN: 0021-9606. DOI: 10.1063/1.5118867.

[Bea04] Richard Beals. *Analysis: An Introduction*. Cambridge University Press, Sept. 2004. ISBN: 978-0-521-60047-7.

[Beh16] Jörg Behler. “Perspective: Machine Learning Potentials for Atomistic Simulations”. In: *The Journal of Chemical Physics* 145.17 (Nov. 2016), p. 170901. ISSN: 0021-9606. DOI: 10.1063/1.4966192.

[Ber+21] Julius Berner et al. “The Modern Mathematics of Deep Learning”. In: *arXiv:2105.04026 [cs, stat]* (May 2021). arXiv: 2105.04026 [cs, stat].

[Bia+12] Emanuela Bianchi et al. “Predicting Patchy Particle Crystals: Variable Box Shape Simulations and Evolutionary Algorithms”. In: *The Journal of Chemical Physics* 136.21 (June 2012), p. 214102. ISSN: 0021-9606. DOI: 10.1063/1.4722477.

[Bis+21] Koushik Biswas et al. “TanhSoft—Dynamic Trainable Activation Functions for Faster Learning and Better Performance”. In: *IEEE Access* 9 (2021), pp. 120613–120623. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3105355.

[Boa+20] Emanuele Boattini et al. “Modeling of Many-Body Interactions between Elastic Spheres through Symmetry Functions”. In: *The Journal of Chemical Physics* 153.6 (Aug. 2020), p. 064902. ISSN: 0021-9606. DOI: 10.1063/5.0015606.

[BP07] Jörg Behler and Michele Parrinello. “Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces”. In: *Physical Review Letters* 98.14 (Apr. 2007), p. 146401. DOI: 10.1103/PhysRevLett.98.146401.

[BPC20] Edwin Bedolla, Luis Carlos Padierna, and Ramón Castañeda-Priego. “Machine Learning for Condensed Matter Physics”. In: *Journal of Physics: Condensed Matter* 33.5 (Nov. 2020), p. 053001. ISSN: 0953-8984. DOI: 10.1088/1361-648X/abb895.

[BSF21] Emanuele Boattini, Frank Smallenburg, and Laura Filion. “Averaging Local Structure to Predict the Dynamic Propensity in Supercooled Liquids”. In: *Physical Review Letters* 127.8 (Aug. 2021), p. 088007. DOI: 10.1103/PhysRevLett.127.088007.

[BSS87] Richard H. Byrd, Robert B. Schnabel, and Gerald A. Shultz. “A Trust Region Algorithm for Nonlinearly Constrained Optimization”. In: *SIAM Journal on Numerical Analysis* 24.5 (Oct. 1987), pp. 1152–1170. ISSN: 0036-1429. DOI: 10.1137/0724076.

[Car+19] Giuseppe Carleo et al. “Machine Learning and the Physical Sciences”. In: *Reviews of Modern Physics* 91.4 (Dec. 2019), p. 045002. DOI: 10.1103/RevModPhys.91.045002.

[CC95] Tianping Chen and Hong Chen. "Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems". In: *IEEE Transactions on Neural Networks* 6.4 (July 1995), pp. 911–917. ISSN: 1941-0093. DOI: 10.1109/72.392253.

[CM17] Juan Carrasquilla and Roger G. Melko. "Machine Learning Phases of Matter". In: *Nature Physics* 13.5 (May 2017), pp. 431–434. ISSN: 1745-2481. DOI: 10.1038/nphys4035.

[CMR21] Omry Cohen, Or Malka, and Zohar Ringel. "Learning Curves for Overparametrized Deep Neural Networks: A Field Theory Perspective". In: *Physical Review Research* 3.2 (Apr. 2021), p. 023034. DOI: 10.1103/PhysRevResearch.3.023034.

[Cyb89] G. Cybenko. "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274.

[DB18] Vedran Dunjko and Hans J. Briegel. "Machine Learning & Artificial Intelligence in the Quantum Domain: A Review of Recent Progress". In: *Reports on Progress in Physics* 81.7 (June 2018), p. 074001. ISSN: 0034-4885. DOI: 10.1088/1361-6633/aab406.

[de 92] P. G. de Gennes. "Soft Matter". In: *Reviews of Modern Physics* 64.3 (July 1992), pp. 645–648. DOI: 10.1103/RevModPhys.64.645.

[DG97] M. Dorigo and L.M. Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 53–66. ISSN: 1941-0026. DOI: 10.1109/4235.585892.

[Dho96] J. K. G. Dhont. *An Introduction to Dynamics of Colloids*. Elsevier, May 1996. ISBN: 978-0-08-053507-4.

[DL21] Marjolein Dijkstra and Erik Luijten. "From Predictive Modelling to Machine Learning and Reverse Engineering of Colloidal Self-Assembly". In: *Nature Materials* 20.6 (June 2021), pp. 762–773. ISSN: 1476-4660. DOI: 10.1038/s41563-021-01014-2.

[DvE99] Marjolein Dijkstra, René van Roij, and Robert Evans. "Phase Diagram of Highly Asymmetric Binary Hard-Sphere Mixtures". In: *Physical Review E* 59.5 (May 1999), pp. 5744–5771. DOI: 10.1103/PhysRevE.59.5744.

[EFD19] Robert Evans, Daan Frenkel, and Marjolein Dijkstra. "From Simple Liquids to Colloids and Soft Matter". In: *Physics Today* 72.2 (Feb. 2019), pp. 38–39. ISSN: 0031-9228. DOI: 10.1063/PT.3.4135.

[EK95] Alan Edelman and Eric Kostlan. "How Many Zeros of a Random Polynomial Are Real?" In: *Bulletin of the American Mathematical Society* 32.1 (Jan. 1995), pp. 1–38. ISSN: 0273-0979. DOI: 10.1090/S0273-0979-1995-00571-9.

[FK09] Alexander I. J. Forrester and Andy J. Keane. "Recent Advances in Surrogate-Based Optimization". In: *Progress in Aerospace Sciences* 45.1 (Jan. 2009), pp. 50–79. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2008.11.001.

[FL19] Jianli Feng and Shengnan Lu. "Performance Analysis of Various Activation Functions in Artificial Neural Networks". In: 1237 (June 2019), p. 022030. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1237/2/022030.

[Fog00] D.B. Fogel. "What Is Evolutionary Computation?" In: *IEEE Spectrum* 37.2 (Feb. 2000), pp. 26–32. ISSN: 1939-9340. DOI: 10.1109/6.819926.

[Fon+10] Oscar Fontenla-Romero et al. "A New Convex Objective Function for the Supervised Learning of Single-Layer Neural Networks". In: *Pattern Recognition* 43.5 (May 2010), pp. 1984–1992. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2009.11.024.

[FS01] Daan Frenkel and B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Elsevier, Oct. 2001. ISBN: 978-0-08-051998-2.

[Fuk+11] Nobusumi Fukushima et al. "Proposal of Distance-Weighted Exponential Natural Evolution Strategies". In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. June 2011, pp. 164–171. DOI: 10.1109/CEC.2011.5949614.

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256.

[GB96] William M. Gelbart and Avinoam Ben-Shaul. "The "New" Science of "Complex Fluids"". In: *The Journal of Physical Chemistry* 100.31 (Jan. 1996), pp. 13169–13189. ISSN: 0022-3654. DOI: 10.1021/jp9606570.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, June 2011, pp. 315–323.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Nov. 2016. ISBN: 978-0-262-33737-3.

[Gib14] J. Willard Gibbs. *Elementary Principles in Statistical Mechanics*. Courier Corporation, Dec. 2014. ISBN: 978-0-486-78995-8.

[GPS02] Herbert Goldstein, Charles P. Poole, and John L. Safko. *Classical Mechanics*. Addison Wesley, 2002. ISBN: 978-0-201-65702-9.

[GR98] Alice P. Gast and William B. Russel. "Simple Ordering in Complex Fluids". In: *Physics Today* 51.12 (Dec. 1998), pp. 24–30. ISSN: 0031-9228. DOI: 10.1063/1.882495.

[Grz+09] Bartosz A. Grzybowski et al. "Self-Assembly: From Crystals to Cells". In: *Soft Matter* 5.6 (Mar. 2009), pp. 1110–1128. ISSN: 1744-6848. DOI: 10.1039/B819321P.

[Ham12] Richard Hamming. *Numerical Methods for Scientists and Engineers*. Courier Corporation, Apr. 2012. ISBN: 978-0-486-13482-6.

[Han06] Nikolaus Hansen. "The CMA Evolution Strategy: A Comparing Review". In: *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*. Ed. by Jose A. Lozano et al. Studies in Fuzziness and Soft Computing. Berlin, Heidelberg: Springer, 2006, pp. 75–102. ISBN: 978-3-540-32494-2. DOI: 10.1007/3-540-32494-1_4.

[HJ12] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Oct. 2012. ISBN: 978-1-139-78888-5.

[HM13] Jean-Pierre Hansen and I. R. McDonald. *Theory of Simple Liquids: With Applications to Soft Matter*. Academic Press, Aug. 2013. ISBN: 978-0-12-387033-9.

[Hor91] Kurt Hornik. “Approximation Capabilities of Multilayer Feedforward Networks”. In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T.

[Hou+09] J. Hough et al. “Zeros of Gaussian Analytic Functions and Determinantal Point Processes”. In: *University Lecture Series*. 2009. DOI: 10.1090/ULECT/051.

[HR68] William G. Hoover and Francis H. Ree. “Melting Transition and Communal Entropy for Hard Spheres”. In: *The Journal of Chemical Physics* 49.8 (Oct. 1968), pp. 3609–3617. ISSN: 0021-9606. DOI: 10.1063/1.1670641.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer Feedforward Networks Are Universal Approximators”. In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Science & Business Media, Aug. 2009. ISBN: 978-0-387-84858-7.

[Hua87] Kerson Huang. *Statistical Mechanics*. Wiley, May 1987. ISBN: 978-0-471-81518-1.

[JLT17] R. B. Jadrich, B. A. Lindquist, and T. M. Truskett. “Probabilistic Inverse Design for Self-Assembling Materials”. In: *The Journal of Chemical Physics* 146.18 (May 2017), p. 184103. ISSN: 0021-9606. DOI: 10.1063/1.4981796.

[JLT18] R. B. Jadrich, B. A. Lindquist, and T. M. Truskett. “Unsupervised Machine Learning for Detection of Phase Transitions in Off-Lattice Systems. I. Foundations”. In: *The Journal of Chemical Physics* 149.19 (Nov. 2018), p. 194109. ISSN: 0021-9606. DOI: 10.1063/1.5049849.

[Kar+21] George Em Karniadakis et al. “Physics-Informed Machine Learning”. In: *Nature Reviews Physics* 3.6 (June 2021), pp. 422–440. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00314-5.

[KB17] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980 [cs].

[Kin03] Masahiro Kinoshita. “Interaction between Surfaces with Solvophobicity or Solvophilicity Immersed in Solvent: Effects Due to Addition of Solvophobic or Solvophilic Solute”. In: *The Journal of Chemical Physics* 118.19 (May 2003), pp. 8969–8981. ISSN: 0021-9606. DOI: 10.1063/1.1566935.

[Kit04] Charles Kittel. *Elementary Statistical Physics*. Courier Corporation, Mar. 2004. ISBN: 978-0-486-43514-5.

[KK05] Sang Kyu Kwak and David A. Kofke. “Evaluation of Bridge-Function Diagrams via Mayer-Sampling Monte Carlo Simulation”. In: *The Journal of Chemical Physics* 122.10 (Mar. 2005), p. 104508. ISSN: 0021-9606. DOI: 10.1063/1.1860559.

[KLM96] L. P. Kaelbling, M. L. Littman, and A. W. Moore. “Reinforcement Learning: A Survey”. In: *Journal of Artificial Intelligence Research* 4 (May 1996), pp. 237–285. ISSN: 1076-9757. DOI: 10.1613/jair.301.

[KP15] Janusz Kacprzyk and Witold Pedrycz. *Springer Handbook of Computational Intelligence*. Springer, May 2015. ISBN: 978-3-662-43505-2.

[Kro+14] Dirk P. Kroese et al. “Why the Monte Carlo Method Is so Important Today”. In: *WIREs Computational Statistics* 6.6 (2014), pp. 386–392. ISSN: 1939-0068. DOI: 10.1002/wics.1314.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Red Hook, NY, USA: Curran Associates Inc., Dec. 2012, pp. 1097–1105.

[LB21] David Landau and Kurt Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, July 2021. ISBN: 978-1-108-49014-6.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[LD08] Wolfgang Lechner and Christoph Dellago. “Accurate Determination of Crystal Structures Based on Averaged Local Bond Order Parameters”. In: *The Journal of Chemical Physics* 129.11 (Sept. 2008), p. 114707. ISSN: 0021-9606. DOI: 10.1063/1.2977970.

[Lec+98] Y. Lecun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[Leu+05] Mirjam E. Leunissen et al. “Ionic Colloidal Crystals of Oppositely Charged Particles”. In: *Nature* 437.7056 (Sept. 2005), pp. 235–240. ISSN: 1476-4687. DOI: 10.1038/nature03946.

[Li+16] Jiwei Li et al. “A Diversity-Promoting Objective Function for Neural Conversation Models”. In: *arXiv:1510.03055 [cs]* (June 2016). arXiv: 1510.03055 [cs].

[LJT16] Beth A. Lindquist, Ryan B. Jadrich, and Thomas M. Truskett. “Communication: Inverse Design for Self-Assembly via on-the-Fly Optimization”. In: *The Journal of Chemical Physics* 145.11 (Sept. 2016), p. 111101. ISSN: 0021-9606. DOI: 10.1063/1.4962754.

[LKDV15] Zhenwei Li, James R. Kermode, and Alessandro De Vita. “Molecular Dynamics with On-the-Fly Machine Learning of Quantum-Mechanical Forces”. In: *Physical Review Letters* 114.9 (Mar. 2015), p. 096405. DOI: 10.1103/PhysRevLett.114.096405.

[LN15] Maxwell W. Libbrecht and William Stafford Noble. “Machine Learning Applications in Genetics and Genomics”. In: *Nature Reviews Genetics* 16.6 (June 2015), pp. 321–332. ISSN: 1471-0064. DOI: 10.1038/nrg3920.

[McQ00] Donald A. McQuarrie. *Statistical Mechanics*. University Science Books, June 2000. ISBN: 978-1-891389-15-3.

[Meh21] B. Mehlig. “Machine Learning with Neural Networks”. In: *arXiv:1901.05639 [cond-mat, stat]* (Feb. 2021). arXiv: 1901.05639 [cond-mat, stat].

[ML87] Anatol Malijevský and Stanislav Labík. “The Bridge Function for Hard Spheres”. In: *Molecular Physics* 60.3 (Feb. 1987), pp. 663–669. ISSN: 0026-8976. DOI: 10.1080/00268978700100441.

[MPP18] Thomas H. McCoy Jr, Amelia M. Pellegrini, and Roy H. Perlis. "Assessment of Time-Series Machine Learning Methods for Forecasting Hospital Discharge Volume". In: *JAMA Network Open* 1.7 (Nov. 2018), e184087. ISSN: 2574-3805. DOI: 10.1001/jamanetworkopen.2018.4087.

[MS67] I. R. McDonald and K. Singer. "Calculation of Thermodynamic Properties of Liquid Argon from Lennard-Jones Parameters by a Monte Carlo Method". In: *Discussions of the Faraday Society* 43.0 (Jan. 1967), pp. 40–49. ISSN: 0366-9033. DOI: 10.1039/DF9674300040.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Aug. 2012. ISBN: 978-0-262-01802-9.

[Ney+18] Behnam Neyshabur et al. "The Role of Over-Parametrization in Generalization of Neural Networks". In: *International Conference on Learning Representations*. Sept. 2018.

[Nie15] Michael Nielsen. *Neural Networks and Deep Learning*. 2015.

[Nom+21] Masahiro Nomura et al. "Distance-Weighted Exponential Natural Evolution Strategy for Implicitly Constrained Black-Box Function Optimization". In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. June 2021, pp. 1099–1106. DOI: 10.1109/CEC45853.2021.9504865.

[NW06] Jorge Nocedal and S. Wright. *Numerical Optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer-Verlag, 2006. ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[Nwa+18] Chigozie Nwankpa et al. "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning". In: *arXiv:1811.03378 [cs]* (Nov. 2018). arXiv: 1811.03378 [cs].

[Par+20] Sejun Park et al. "Minimum Width for Universal Approximation". In: *International Conference on Learning Representations*. Sept. 2020.

[PM47] Walter Pitts and Warren S. McCulloch. "How We Know Universals the Perception of Auditory and Visual Forms". In: *The bulletin of mathematical biophysics* 9.3 (Sept. 1947), pp. 127–147. ISSN: 1522-9602. DOI: 10.1007/BF02478291.

[Pow02] M.J.D. Powell. "UOBYQA: Unconstrained Optimization by Quadratic Approximation". In: *Mathematical Programming* 92.3 (May 2002), pp. 555–582. ISSN: 1436-4646. DOI: 10.1007/s101070100290.

[Pv86] P. N. Pusey and W. van Megen. "Phase Behaviour of Concentrated Suspensions of Nearly Hard Colloidal Spheres". In: *Nature* 320.6060 (Mar. 1986), pp. 340–342. ISSN: 1476-4687. DOI: 10.1038/320340a0.

[PY58] Jerome K. Percus and George J. Yevick. "Analysis of Classical Statistical Mechanics by Means of Collective Coordinates". In: *Physical Review* 110.1 (Apr. 1958), pp. 1–13. DOI: 10.1103/PhysRev.110.1.

[Rad+18] Alexander Radovic et al. "Machine Learning at the Energy and Intensity Frontiers of Particle Physics". In: *Nature* 560.7716 (Aug. 2018), pp. 41–48. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0361-2.

[RF18] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv:1804.02767 [cs]* (Apr. 2018). arXiv: 1804.02767 [cs].

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

[RHC18] Dina Razafindralandy, Aziz Hamdouni, and Marx Chhay. “A Review of Some Geometric Integrators”. In: *Advanced Modeling and Simulation in Engineering Sciences* 5.1 (June 2018), p. 16. ISSN: 2213-7467. DOI: 10.1186/s40323-018-0110-y.

[Ric06] John A. Rice. *Mathematical Statistics and Data Analysis*. Cengage Learning, Apr. 2006. ISBN: 978-0-534-39942-9.

[RKD20] Clémence Réda, Emilie Kaufmann, and Andrée Delahaye-Duriez. “Machine Learning Applications in Drug Development”. In: *Computational and Structural Biotechnology Journal* 18 (Jan. 2020), pp. 241–252. ISSN: 2001-0370. DOI: 10.1016/j.csbj.2019.12.006.

[RLS14] Miguel Robles, Mariano López de Haro, and Andrés Santos. “Note: Equation of State and the Freezing Point in the Hard-Sphere Model”. In: *The Journal of Chemical Physics* 140.13 (Apr. 2014), p. 136101. ISSN: 0021-9606. DOI: 10.1063/1.4870524.

[Ros58] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471. DOI: 10.1037/h0042519.

[RS03] Robin D. Rogers and Kenneth R. Seddon. “Ionic Liquids–Solvents of the Future?” In: *Science* 302.5646 (Oct. 2003), pp. 792–793. DOI: 10.1126/science.1090313.

[Rud13] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 2013. ISBN: 978-1-259-06478-4.

[Rud17] Sebastian Ruder. “An Overview of Gradient Descent Optimization Algorithms”. In: *arXiv:1609.04747 [cs]* (June 2017). arXiv: 1609.04747 [cs].

[RWB06] Carl Edward Rasmussen, Christopher K. I. Williams, and Francis Bach. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN: 978-0-262-18253-9.

[RY84] Forrest J. Rogers and David A. Young. “New, Thermodynamically Consistent, Integral Equation for Simple Fluids”. In: *Physical Review A* 30.2 (Aug. 1984), pp. 999–1007. DOI: 10.1103/PhysRevA.30.999.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: *arXiv:1710.05941 [cs]* (Oct. 2017). arXiv: 1710.05941 [cs].

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition: An Introduction*. MIT Press, Nov. 2018. ISBN: 978-0-262-35270-3.

[SC08] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Science & Business Media, Sept. 2008. ISBN: 978-0-387-77242-4.

[Sch+16] S. S. Schoenholz et al. “A Structural Approach to Relaxation in Glassy Liquids”. In: *Nature Physics* 12.5 (May 2016), pp. 469–471. ISSN: 1745-2481. DOI: 10.1038/nphys3644.

[SGO20] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. “Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005–2019”. In: *Applied Soft Computing* 90 (May 2020), p. 106181. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2020.106181.

[She+20] Zachary M. Sherman et al. “Inverse Methods for Design of Soft Materials”. In: *The Journal of Chemical Physics* 152.14 (Apr. 2020), p. 140902. ISSN: 0021-9606. DOI: 10.1063/1.5145177.

[SNR83] Paul J. Steinhardt, David R. Nelson, and Marco Ronchetti. “Bond-Orientational Order in Liquids and Glasses”. In: *Physical Review B* 28.2 (July 1983), pp. 784–805. DOI: 10.1103/PhysRevB.28.784.

[Sto37] M. H. Stone. “Applications of the Theory of Boolean Rings to General Topology”. In: *Transactions of the American Mathematical Society* 41.3 (1937), pp. 375–481. ISSN: 0002-9947, 1088-6850. DOI: 10.1090/S0002-9947-1937-1501905-7.

[Sto48] M. H. Stone. “The Generalized Weierstrass Approximation Theorem”. In: *Mathematics Magazine* 21.4 (1948), pp. 167–184. ISSN: 0025-570X. DOI: 10.2307/3029750.

[Tao11] Terence Tao. *An Introduction to Measure Theory*. American Mathematical Soc., Sept. 2011. ISBN: 978-0-8218-6919-2.

[TL19] Tsogbayar Tsednee and Tyler Luchko. “Closure for the Ornstein-Zernike Equation with Pressure and Free Energy Consistency”. In: *Physical Review E* 99.3 (Mar. 2019), p. 032130. DOI: 10.1103/PhysRevE.99.032130.

[Tol79] Richard Chace Tolman. *The Principles of Statistical Mechanics*. Courier Corporation, Jan. 1979. ISBN: 978-0-486-63896-6.

[Tre13] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, Jan. 2013. ISBN: 978-1-61197-240-5.

[van+20] Robin van Damme et al. “Classifying Crystals of Rounded Tetrahedra and Determining Their Order Parameters Using Dimensionality Reduction”. In: *ACS Nano* 14.11 (Nov. 2020), pp. 15144–15153. ISSN: 1936-0851. DOI: 10.1021/acsnano.0c05288.

[Ver67] Loup Verlet. “Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules”. In: *Physical Review* 159.1 (July 1967), pp. 98–103. DOI: 10.1103/PhysRev.159.98.

[VLC18] Néstor E. Valadez-Pérez, Yun Liu, and Ramón Castañeda-Priego. “Reversible Aggregation and Colloidal Cluster Morphology: The Importance of the Extended Law of Corresponding States”. In: *Physical Review Letters* 120.24 (June 2018), p. 248004. DOI: 10.1103/PhysRevLett.120.248004.

[VM94] A. G. Vompe and G. A. Martynov. “The Bridge Function Expansion and the Self-consistency Problem of the Ornstein–Zernike Equation Solution”. In: *The Journal of Chemical Physics* 100.7 (Apr. 1994), pp. 5249–5258. ISSN: 0021-9606. DOI: 10.1063/1.467189.

[Wie+14] Daan Wierstra et al. “Natural Evolution Strategies”. In: *Journal of Machine Learning Research* 15.27 (2014), pp. 949–980. ISSN: 1533-7928.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *arXiv:1708.07747 [cs, stat]* (Sept. 2017). arXiv: 1708.07747 [cs, stat].

[YG10] Xinjie Yu and Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Springer Science & Business Media, June 2010. ISBN: 978-1-84996-129-5.

[ZH86] Gilles Zerah and Jean-Pierre Hansen. "Self-consistent Integral Equations for Fluid Pair Distribution Functions: Another Attempt". In: *The Journal of Chemical Physics* 84.4 (Feb. 1986), pp. 2336–2343. ISSN: 0021-9606. DOI: 10.1063/1.450397.

[Zho20] Ding-Xuan Zhou. "Universality of Deep Convolutional Neural Networks". In: *Applied and Computational Harmonic Analysis* 48.2 (Mar. 2020), pp. 787–794. ISSN: 1063-5203. DOI: 10.1016/j.acha.2019.06.004.

[Zhu+19] Yinhao Zhu et al. "Physics-Constrained Deep Learning for High-Dimensional Surrogate Modeling and Uncertainty Quantification without Labeled Data". In: *Journal of Computational Physics* 394 (Oct. 2019), pp. 56–81. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.05.024.