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

September 28, 2021

"*Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.*"

Dave Barry

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

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

For/Dedicated to/To my...

Chapter 1

Introduction

Since the mainstream adoption of *Machine Learning* (ML) methods on common tasks such as object recognition, face identification, 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 important research fields 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 the use of *computer vision* and *deep learning* techniques in particle physics 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 the have been developed for. However, it turns out that not all ML techniques can be readily applied to they problem at hand, and one should instead try to use the Physics of the problem and use it along with the ML method to boost its usefulness and flexibility.

Instead, scientists have preferred to incorporate most of the Physics into the ML method, and thus create a new form of *physics-inspired machine learning*. 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 defined 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. 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 of such successful applications was 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 still being part of the exploration step is that Soft Matter and Liquid Theory is hard to find suitable descriptors that actually tell the information we want from the system [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.

Chapter 2

Neural networks as an approximation for the bridge function

Neural networks can be used as *universal approximators* [HSW89; Hor91; Cyb89], i.e., they can take the form of any continuous function if and only if the conditions of the *universal approximation theorem* hold [Par+20; Zho20]. 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.

2.1 Parametrization of the bridge function

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

$$\begin{aligned} h(\mathbf{r}) &= c(\mathbf{r}) + n\int_V c(\mathbf{r}')h(|\mathbf{r}-\mathbf{r}'|)d\mathbf{r}' \\ c(\mathbf{r}) &= \exp\left[-\beta u(\mathbf{r}) + \gamma(\mathbf{r}) + B(\mathbf{r})\right] - \gamma(\mathbf{r}) - 1, \end{aligned} \tag{2.1}$$

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

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

$$c(\mathbf{r}) = \exp\left[-\beta u(\mathbf{r}) + \gamma(\mathbf{r}) + N_\theta(\mathbf{r})\right] - \gamma(\mathbf{r}) - 1. \tag{2.2}$$

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

2.2 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 $N_\theta(\mathbf{r})$.

2.2.1 Cost function

It was mentioned previously that the main problem is to find the weights of $N_\theta(\mathbf{r})$ 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, $\{\gamma_1(\mathbf{r}), \gamma_2(\mathbf{r}), \ldots, \gamma_n(\mathbf{r})\}$. 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

$$J(\theta) = \left[\gamma_n(\mathbf{r},\theta) - \gamma_{n-1}(\mathbf{r},\theta)\right]^2, \tag{2.3}$$

where $\gamma_n(\mathbf{r},\theta)$ is the n-th approximation of the indirect correlation function, $\gamma(\mathbf{r})$. The notation $\gamma(\mathbf{r},\theta)$ indicates that the function now depends on the weights of the neural network, as seen in Equation 2.2. This means that, if the weights of $N_\theta(\mathbf{r})$ change, we should expect a change in the output from the γ function.

Another way of looking at Equation 2.3 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.

2.2.2 Optimization problem

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

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

$$\min_{\theta\in\mathbb{R}^n} \quad J(\theta). \tag{2.4}$$

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.

2.2.3 Weight updates

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

reads [GBC16]

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta), \tag{2.5}$$

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.

2.2.4 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 2.2:

1. Given a particular interaction potential $\beta u(\mathbf{r})$, Equation 2.2 is used to obtain the value of the direct correlation function $c(\mathbf{r})$. In this step, an initial value for $\gamma_n(\mathbf{r})$ is needed, which is initialized based on the five-point Ng methodology shown in Appendix B. The weights of the neural network $N_\theta(\mathbf{r})$ are initialized randomly. The initialization method is discussed in detail in the next section.

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

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, $\hat{\gamma}_{n+1}(\mathbf{k},\theta)$.

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

5. Both estimations, γ_n and γ_{n+1}, are used to evaluate Equation 2.3. 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 2.5, 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.

2.2.5 Convergence criterion

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

$$\sum_{i=1}^{N} \left(\gamma_i^{n+1} - \gamma_i^n\right)^2 \leq \epsilon. \tag{2.6}$$

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).

2.3 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.

2.3.1 Choice of optimization algorithm

The general rule for the weight update based on Equation 2.5 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

$$\begin{aligned}
m_t &= \beta_1 m_{t-1} - (1-\beta_1)\nabla_{\theta_{t-1}} J(\theta_{t-1}) \\
s_t &= \beta_2 s_{t-1} + (1-\beta_2)\nabla_{\theta_{t-1}} J(\theta_{t-1}) \odot \nabla_{\theta_{t-1}} J(\theta_{t-1}) \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{s}_t &= \frac{s_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} + \eta \hat{m}_t \oslash \sqrt{\hat{s}_t + \varepsilon}
\end{aligned} \tag{2.7}$$

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.

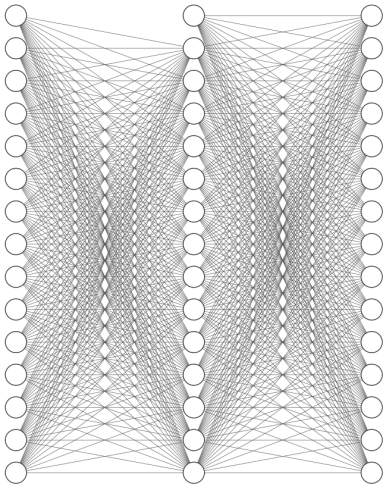

FIGURE 2.1: Cartoon of a fully connected multilayer neural network. Note that there is one *hidden layer*. The circles represent the *nodes* or *units* used to compute the final output. These nodes are being evaluated by an activation function to account for nonlinearities. The top-most nodes that seem 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 2.5, 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].

2.3.2 Neural network topology

The neural network topology used in all the experiments is identical to the one shown in Figure 2.1, 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).

2.3.3 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.

2.4 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.

2.4.1 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 2.2 and Figure 2.3, 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 2.2 and Figure 2.3. 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.

2.4.2 High densities

We now turn our attention to the high density values, namely, $\phi = 0.35$ and 0.45, represented in Figure 2.4 and 2.5. 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 2.4 and Figure 2.5, 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 $B(\mathbf{r}) \approx 0$. If we now pay attention to the modified Verlet approximation, again in Figure 2.4 and Figure 2.5, 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.

2.5 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.

2.5.1 Weight evolution of the neural network

We shall now examine the evolution of the weights θ from $N_\theta(\mathbf{r})$, 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 2.6, Figure 2.7, Figure 2.8, and Figure 2.9, 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

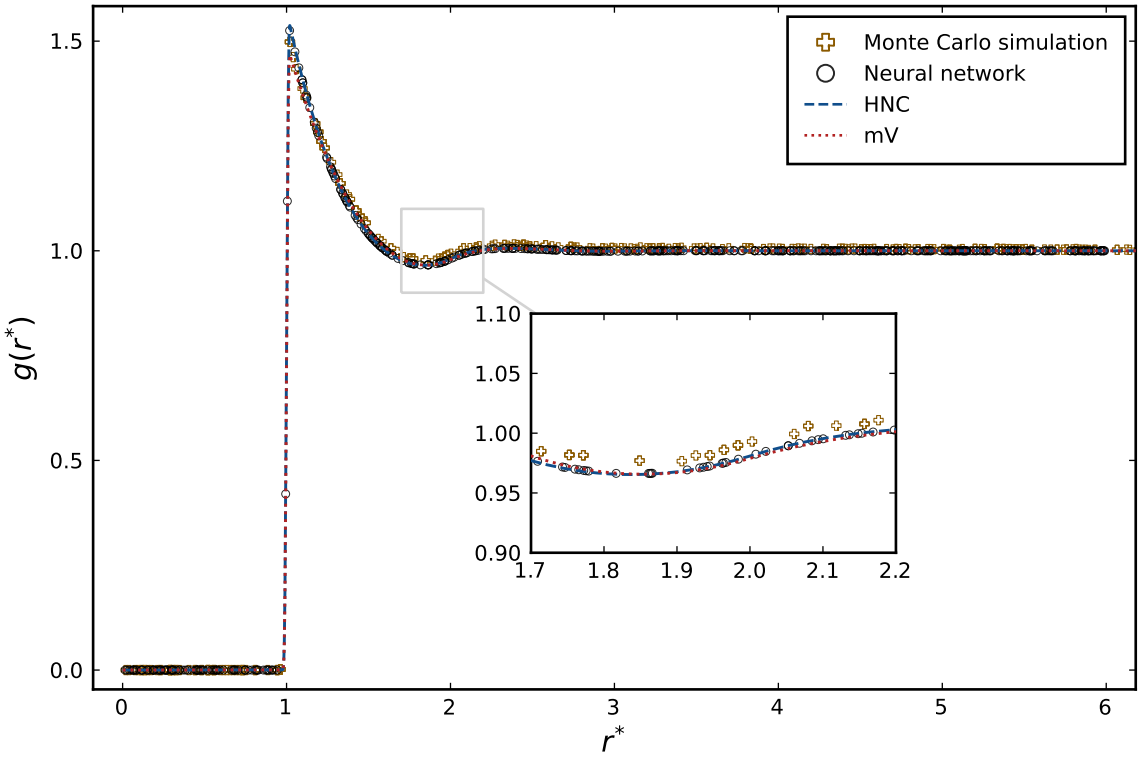

FIGURE 2.2: Radial distribution function for $\phi = 0.15$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

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 2.6 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 2.9 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 $N_\theta(\mathbf{r})$ 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.

2.5.2 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

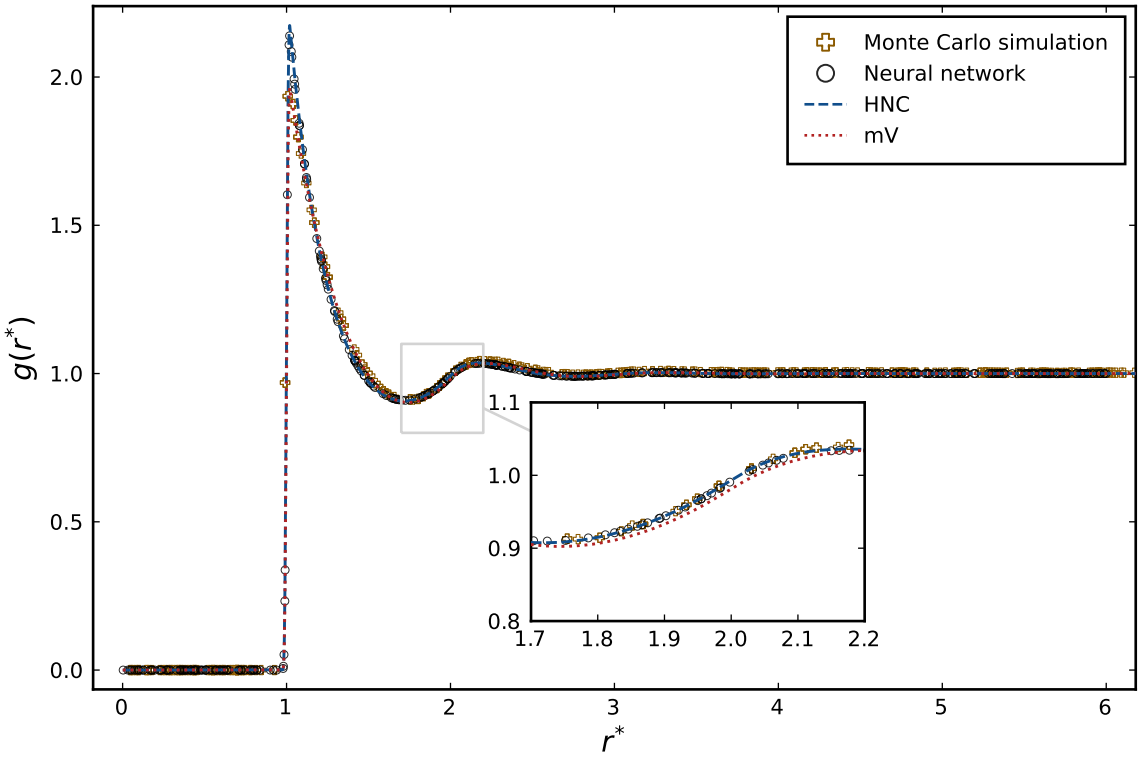

FIGURE 2.3: Radial distribution function for $\phi = 0.25$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

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 2.3. With the results presented so far, it might be possible to postulate that the *HNC approximation is a stable minimum* for the neural network $N_\theta(\mathbf{r})$. 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 2.2, 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 2.5, 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 2.6 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

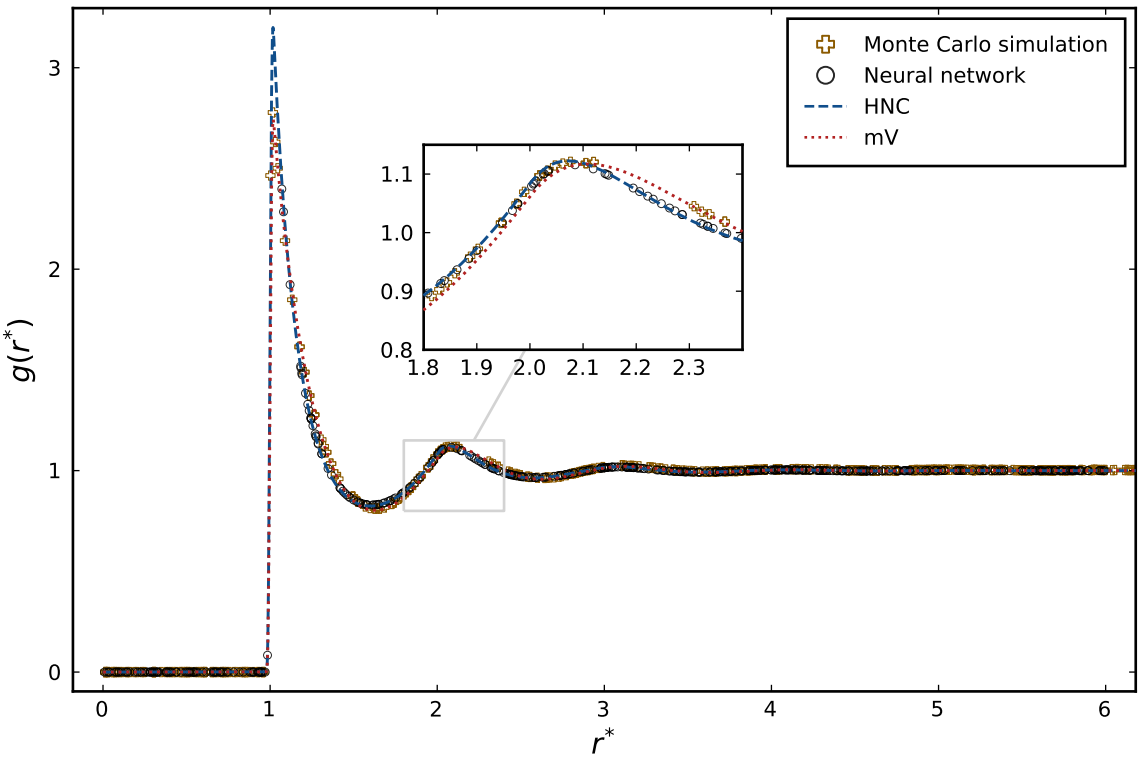

FIGURE 2.4: Radial distribution function for $\phi = 0.35$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

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.

2.5.3 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,

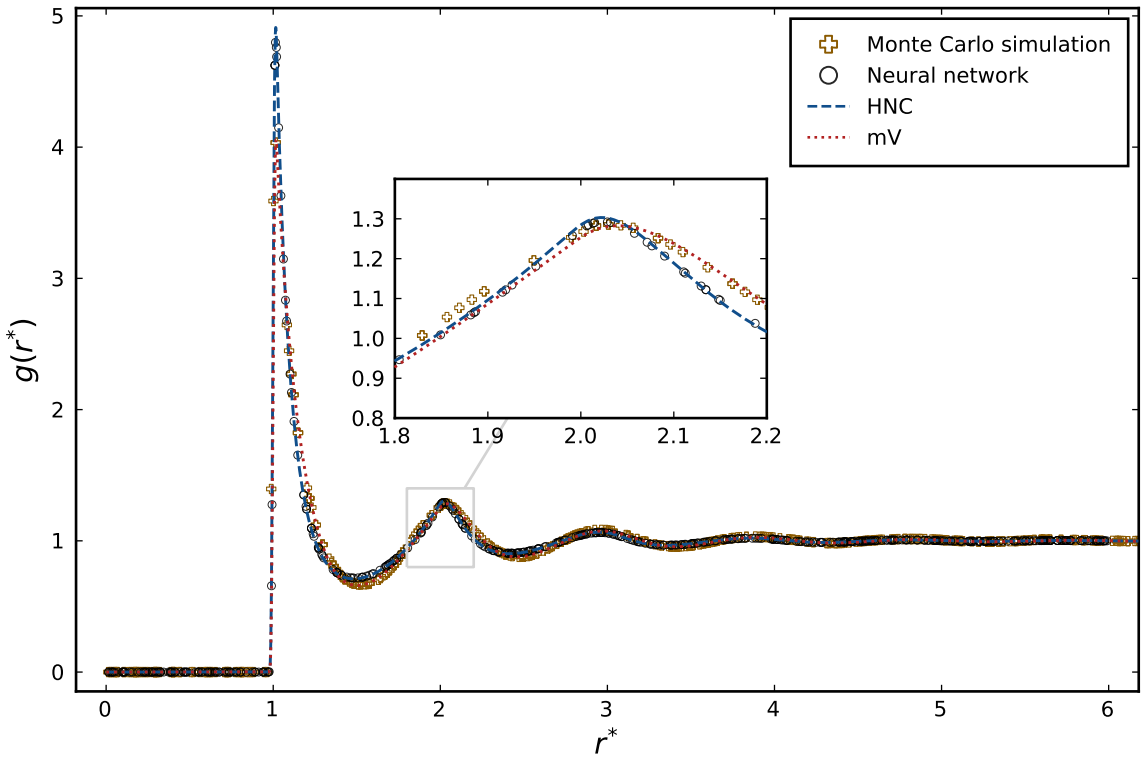

FIGURE 2.5: Radial distribution function for $\phi = 0.45$ obtained from Monte Carlo simulations, and three different approximations: (a) *mV*, (modified Verlet), (b) *HNC*, (Hypernetted Chain), (c) *NN*, (neural network approximation). Inset shows the region close to the peak about $r^* = 2$.

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 $\gamma(\mathbf{r})$, 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.

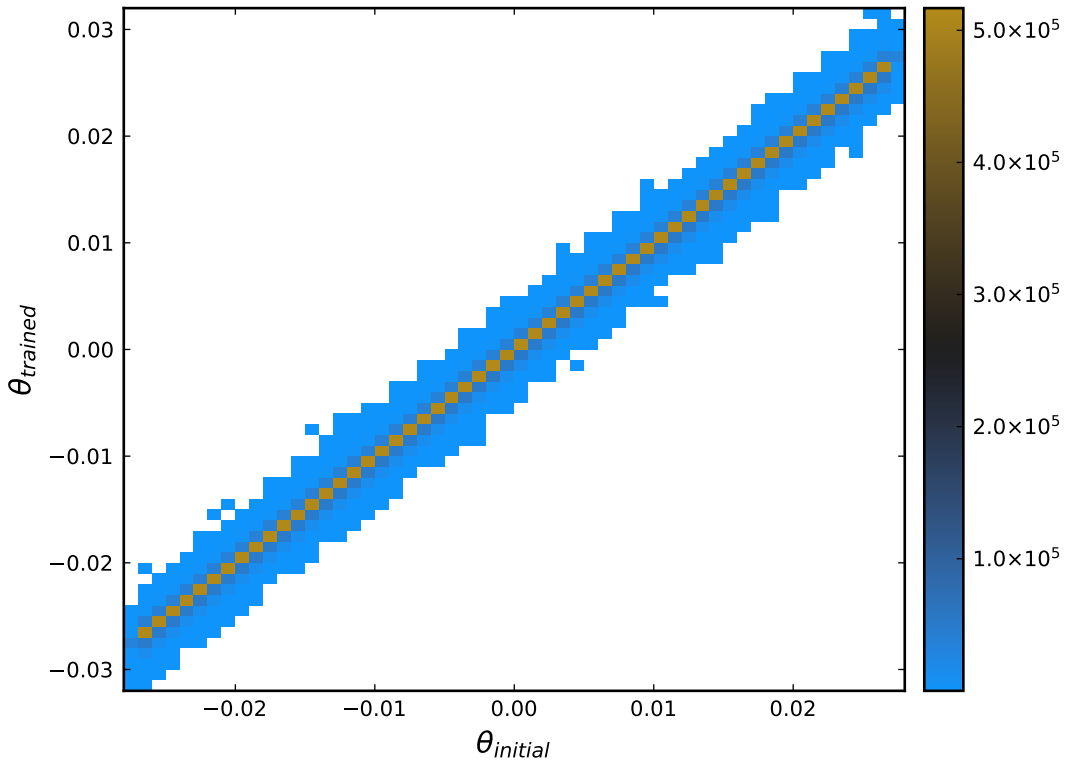

FIGURE 2.6: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for $\phi = 0.15$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

2.5.4 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],

$$b(\mathbf{r}) = b^{(2)}(\mathbf{r})\rho^2 + b^{(3)}(\mathbf{r})\rho^3 + \cdots, \tag{2.8}$$

where the notation $b^{(n)}(\mathbf{r})$ 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 $b^{(n)}(\mathbf{r})$ 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 $b^{(4)}(\mathbf{r})$ for the hard-sphere fluid. They report that even if the

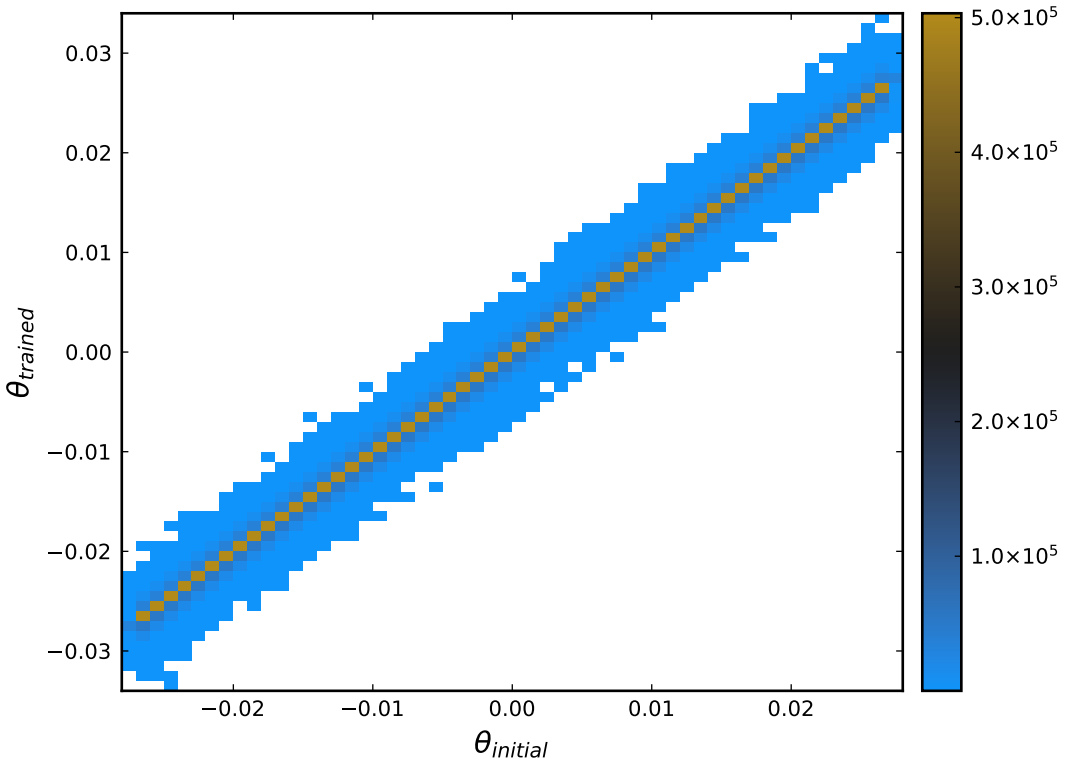

FIGURE 2.7: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for $\phi = 0.25$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

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

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

is known from diagrammatic methods [HM13]. In this expression, $g_1(\mathbf{r})$ 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

$$p_n(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots + a_n z^n, \quad z \in \mathbb{C} \tag{2.9}$$

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 2.9. 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 2.9 and Equation 2.8, we can see that these expressions are

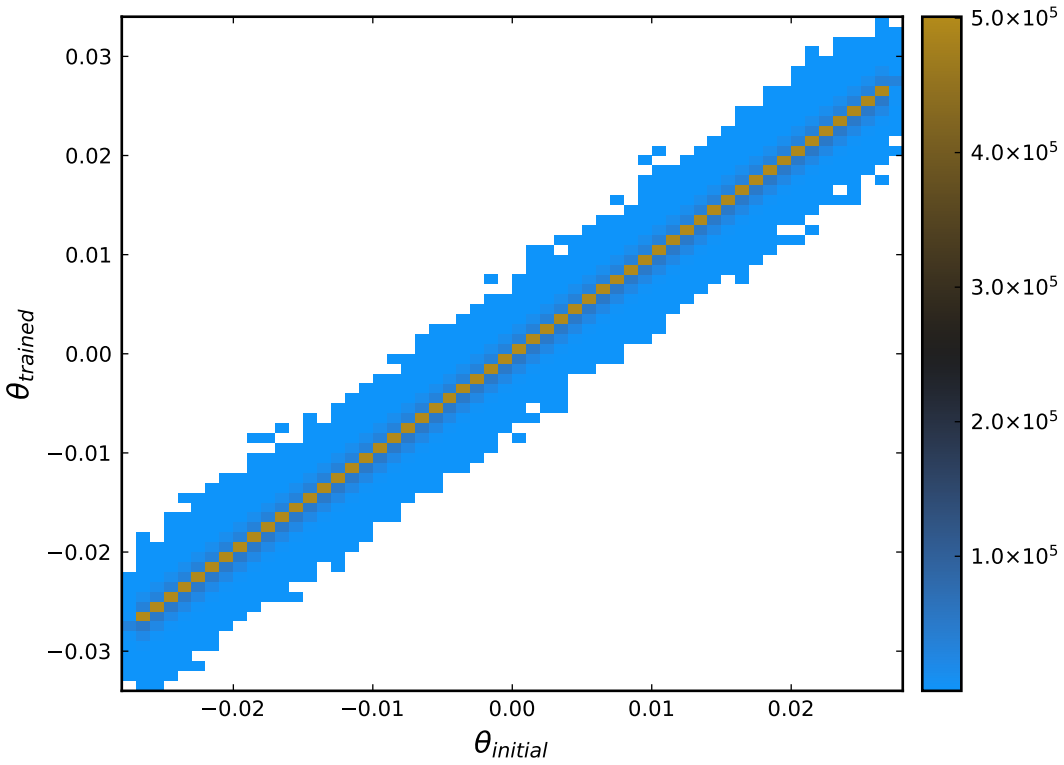

FIGURE 2.8: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for $\phi = 0.35$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

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.

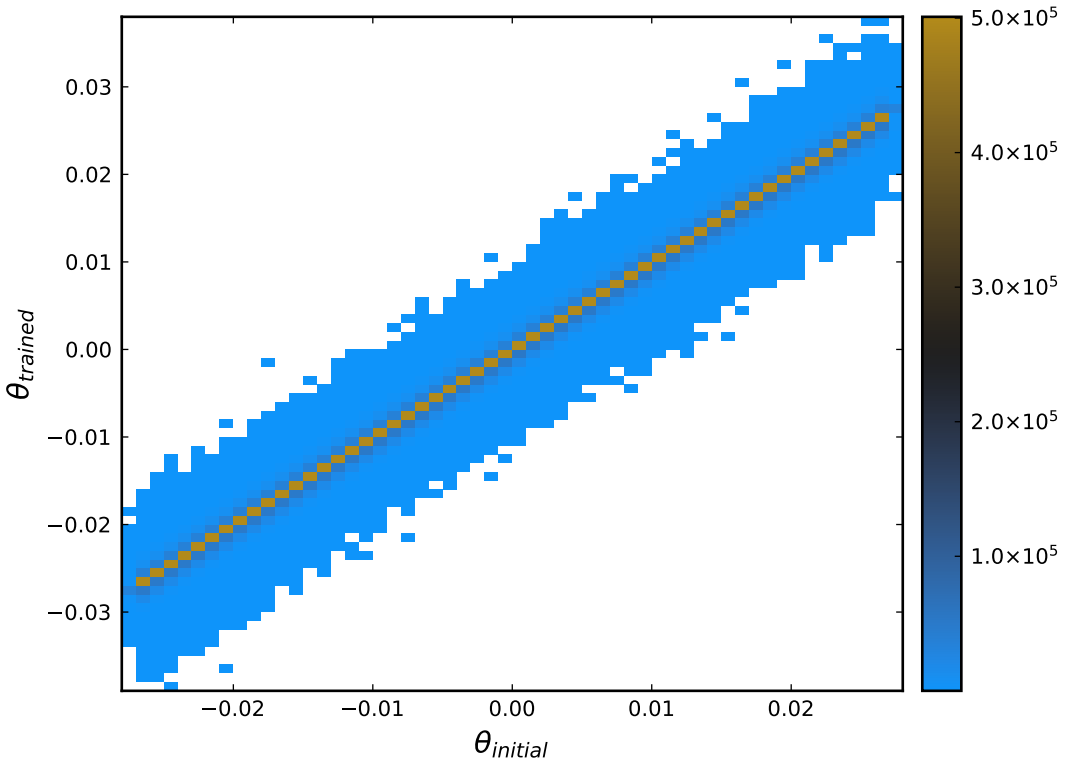

FIGURE 2.9: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for $\phi = 0.45$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

2.6 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 $N_\theta(\mathbf{r})$. 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 3

Evolutionary optimization for the Kinoshita closure

Some text…

Appendix A

Gradient Computation

In chapter 2 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 $N_\theta(\mathbf{r})$ with weights θ to work as a parametrization in the closure expression of the OZ equation, as defined in Equation 2.2. 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 2.3; ∇_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 2.3, we have for the gradient the following expression

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

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

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

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

which results from direct application of the chain rule. Now, an expression for $\partial_\theta \gamma_n(\mathbf{r},\theta)$ 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 $N_\theta(\mathbf{r})$, which has a direct dependency on the weights.

In order to find this new expression, we invoke Equation 2.2 and differentiate it with respect to the weights

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

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

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

with derivative with respect to the weights

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

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

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

Note, however, that this new expression depends on the value of $\partial_\theta c(\mathbf{r},\theta)$ which we do not readily have at this point. It is at this step that we propose to compute the value of $\partial_\theta c(\mathbf{r},\theta)$ 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 $c(\mathbf{r},\theta)$ can be obtained. This is already described in detail in subsection 2.2.4, 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 $\gamma(\mathbf{r},\theta)$ function. At this step we have found also an approximation for $c(\mathbf{r},\theta)$, 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 $c(\mathbf{r},\theta)$ 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 $\partial_\theta c(\mathbf{r},\theta)$.

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 $c(\mathbf{r},\theta)$ function, which is essentially a function of two variables. The first variable, $\mathbf{r}$ is the *Euclidean distance* between two

particles, or in other words $\mathbf{r} = |\mathbf{r}_1 - \mathbf{r}_2|^2$. For the second variable θ, this represents the weights of $N_\theta(\mathbf{r})$, 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 $c(\mathbf{r},\theta) : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$. For a multivariable function, its partial derivative is computed numerically, for a particular value of distance $\bar{\mathbf{r}}$, as

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

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

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

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 $\partial_\theta c(\bar{\mathbf{r}},\theta)$ 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 $\partial_\theta \gamma_n(\mathbf{r},\theta)$. We take Equation A.6 together with Equation A.5 to obtain the following expression

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

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

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

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

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

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

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

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

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

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

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

[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.

[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.

[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”. en. In: *Journal of Physics: Condensed Matter* 33.5 (Nov. 2020), p. 053001. ISSN: 0953-8984. DOI: 10.1088/1361-648X/abb895.

[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.

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

[Cyb89] G. Cybenko. “Approximation by Superpositions of a Sigmoidal Function”. en. 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”. en. In: *Reports on Progress in Physics* 81.7 (June 2018), p. 074001. ISSN: 0034-4885. DOI: 10.1088/1361-6633/aab406.

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

[EK95] Alan Edelman and Eric Kostlan. “How Many Zeros of a Random Polynomial Are Real?” en. 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.

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

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. en. 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*. en. MIT Press, Nov. 2016. ISBN: 978-0-262-33737-3.

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

[HJ12] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. en. 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*. en. Academic Press, Aug. 2013. ISBN: 978-0-12-387033-9.

[Hor91] Kurt Hornik. “Approximation Capabilities of Multilayer Feedforward Networks”. en. 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.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer Feedforward Networks Are Universal Approximators”. en. 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*. en. Springer Science & Business Media, Aug. 2009. ISBN: 978-0-387-84858-7.

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

[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.

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

[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”. en. In: *Nature Reviews Genetics* 16.6 (June 2015), pp. 321–332. ISSN: 1471-0064. DOI: 10.1038/nrg3920.

[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.

[NW06] Jorge Nocedal and S. Wright. *Numerical Optimization*. en. 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.

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

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

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

[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.

[SC08] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. en. 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”. en. In: *Nature Physics* 12.5 (May 2016), pp. 469–471. ISSN: 1745-2481. DOI: 10.1038/nphys3644.

[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.

[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.

[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”. en. 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”. en. In: *Journal of Computational Physics* 394 (Oct. 2019), pp. 56–81. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.05.024.