UNIVERSIDAD DE GUANAJUATO

MASTER'S THESIS

Using Computational Intelligence to solve the Ornstein-Zernike equation

Author:
Edwin Armando Bedolla Montiel

Supervisor:
Dr. James SMITH

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

in the

Soft Matter Group
Department of Physical Engineering

July 8, 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
Department of Physical Engineering

Master in Science

Using Computational Intelligence to solve the Ornstein-Zernike equation

by Edwin Armando Bedolla Montiel

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Acknowledgements

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

Contents

List of Figures

List of Tables

For/Dedicated to/To my…

Chapter 1

Neural networks as an approximation for the bridge function

Neural networks can be used as *universal approximators* [HSW89; Hor91; Cyb89], in other words, they can take the form of any continuous function for some specific types of architectures. In particular, it is hypothesized that a neural network might be useful as a bridge function parametrization in the closure expression for the Ornstein-Zernike equation. 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 the Ornstein-Zernike equation.

In this chapter, we show in detail the methodology created to achieve such a task, and the mathematical structure with which a neural network can be used to solve the Ornstein-Zernike equation. These results are compared to those obtained from 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.

1.1 Parametrization of the bridge function

The Ornstein-Zernike equation is given by the following expression

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

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

$$c(\mathbf{r}) = \exp\left[-\beta u(\mathbf{r}) + \gamma(\mathbf{r}) + N_\theta(\mathbf{r})\right] - \gamma(\mathbf{r}) - 1. \tag{1.2}$$

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, $\beta u(\mathbf{r})$.

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

1.2.1 Cost function

It was mentioned previously that 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. To solve such 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, $\{\gamma_1(\mathbf{r}), \gamma_2(\mathbf{r}), \dots, \gamma_n(\mathbf{r})\}$. 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

$$J(\theta) = \left[\gamma_n(\mathbf{r};\theta) - \gamma_{n-1}(\mathbf{r};\theta)\right]^2 \tag{1.3}$$

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 implicitly on the weights of the neural network, as seen in Equation 1.2. This means that, if the weights of $N_\theta(\mathbf{r})$ change, we should expect a change in the output from the γ function. Nevertheless, this does not mean that the indirect correlation function itself depends explicitly, nor directly, on the weights of $N_\theta(\mathbf{r})$.

Another way of looking at Equation 1.3 is that we require that the last two approximations of the γ function in each iteration from the numerical scheme to be as equal as possible. This will enforce a change on the weights every time both approximations deviate between them.

1.2.2 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

$$\min_{\theta} \quad J(\theta) \quad . \tag{1.4}$$

This formulation is just a search for the best values for the weights that minimize the squared difference between successive approximations. This optimization problem can be solved iteratively, along with the solution of the OZ equation, which is also an iterative process.

1.2.3 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

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta). \tag{1.5}$$

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.

Regardless of 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.

1.2.4 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 using the parametrization described by Equation 1.2:

1. Given a particular interaction potential $\beta u(\mathbf{r})$, Equation 1.2 is used to obtain the value of the direct correlation function $c(\mathbf{r};\theta)$, which now depends implicitly on the weights of $N_{\theta}(\mathbf{r})$. 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.

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

3. Then, the full OZ equation(Ref a ec) 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 the cost function (1.3), while simultaneously computing the gradient $\nabla_{\theta} J(\theta)$.

6. The weights θ are updated a gradient descent rule, similar to Equation 1.5, 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.

1.2.5 Convergence criterion

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

$$\sum_{i=1}^{N} \left(\gamma_i^{n+1} - \gamma_i^{n}\right)^2 \leq \epsilon. \tag{1.6}$$

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 $\epsilon \in [0,1]$ 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. Specifically, the numerical tolerance in all the experiments was fixed to be $\epsilon = 1 \times 10^{-5}$. 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.

1.3 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.

1.3.1 Choice of optimization algorithm

The general rule for the weight update based on gradient descent (1.5) 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 so 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 by the method; β_1, which controls the moving average of the computed gradient; and β_2, which controls the value of the gradient squared. Both parameters are necessary for the optimal convergence of the algorithm.

The equations that define the optimization method are the following

$$
\begin{aligned}
m &= \beta_1 m - (1-\beta_1)\nabla_\theta J(\theta) \\
s &= \beta_2 s + (1-\beta_2)\nabla_\theta J(\theta) \odot \nabla_\theta J(\theta) \\
\hat{m} &= \frac{m}{1-\beta_1^t} \\
\hat{s} &= \frac{s}{1-\beta_2^t} \\
\theta &= \theta + \eta \hat{m} \oslash \sqrt{\hat{s}+\varepsilon}
\end{aligned}
\tag{1.7}
$$

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.

In the results presented in this chapter, the parameters were fixed to the ones reported as optimal in the original work [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 1.5, was fixed to $\eta = 1 \times 10^{-4}$ for all the experiments.

1.3.2 Neural network architecture

The neural network architecture used in all the experiments is very similar to the one shown in Figure 1.1, with the exception of the number of nodes in all the layers. Particularly, the neural network is made of *three layers*, all connected among 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*.

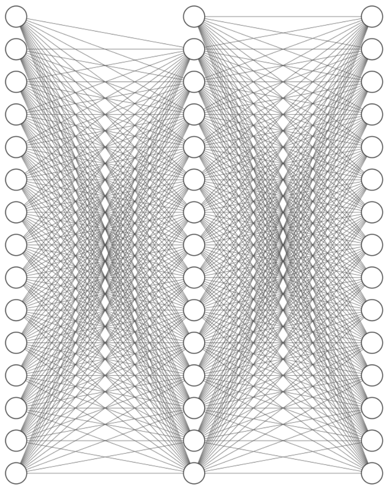

FIGURE 1.1: Cartoon of a fully connected multilayer neural network. Note that there is one *hidden layer*. The circles represent the *nodes* or *units* used to compute the final output. These nodes are being evaluated by an activation function to account for nonlinearities. The top-most nodes that seem different from the main nodes are known as the *bias* nodes. The real architecture used in this chapter is larger, with many more nodes and connections, but the topology is the same.

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 $\theta_{ij} \sim \mathscr{U}\left[-\frac{6}{\sqrt{(in+out)}}, \frac{6}{\sqrt{(in+out)}}\right]$, where $\mathscr{U}$ 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

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

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.

1.3.3 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 for the NVT ensemble (Ref a marco teórico). 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 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 were applied 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.

1.4 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.

1.4.1 Low densities

In this section we will deal with the low density values of $\phi = 0.15$ and 0.25, which are shown in figures (1.2, 1.3). The results show that, at low densities, the HNC and neural network approximations are more precise than the modified Verlet approximation. Although, all approximations seem to fall short compared to computer simulations. This is seen espcially in the neighborhood around the second peak, which are represented in the insets from figures (1.2, 1.3). It is specially important to note that the neural network approximation is a little bit 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 undervalue the main peak.

It is also important to notice the functional form of $g(r^*)$. For the HNC and neural network approximations, it appears to have the same form between 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 slightly 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.

1.4.2 High densities

We now turn our attention to the high density values of $\phi = 0.35$ and 0.45, represented in figures (1.4, 1.5). 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 appraised in figures (1.4, 1.5), where the main peak is well estimated by the

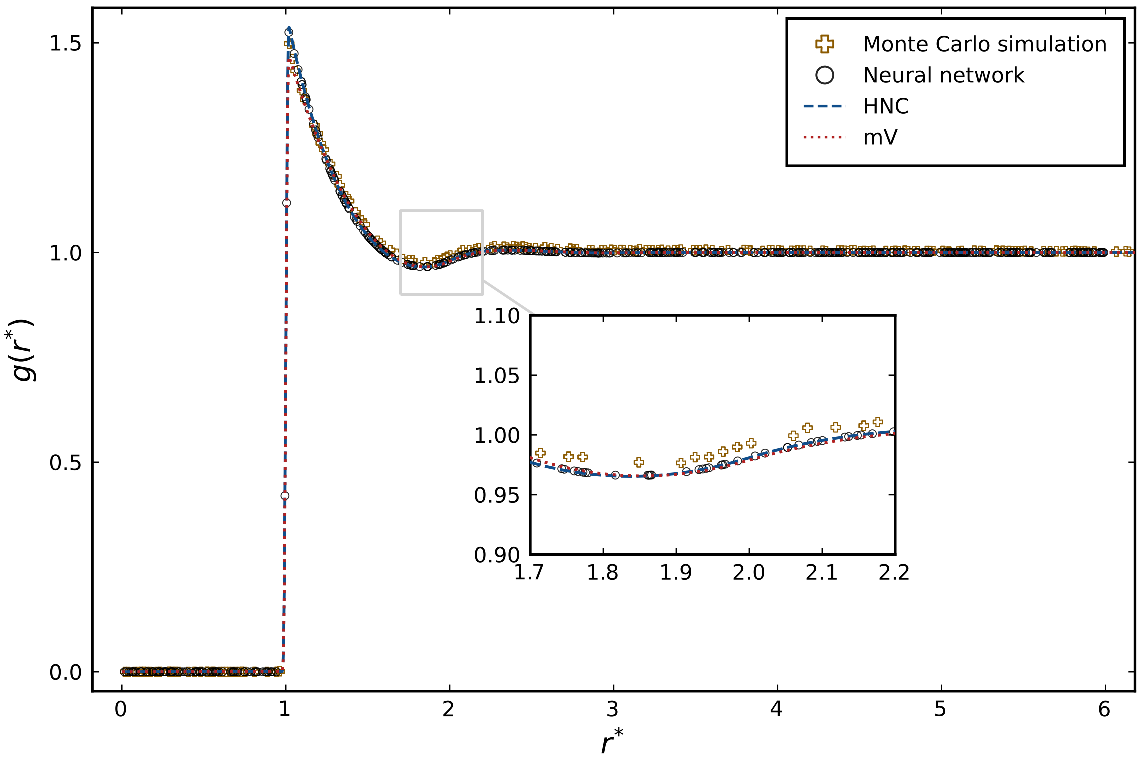

FIGURE 1.2: The radial distribution function for density value $\phi = 0.15$ obtained from computer simulations, and three different approximations: (a) *mV*, modified Verlet, (b) *HNC*, Hypernetted Chain, (c) *Neural network*, neural network approximation. Inset shows the region close to the peak about $r^* = 2$.

approximation when compared to the computer simulation results. However, the HNC and neural network approximation overestimate this property.

Further, the functional form of $g(r^*)$ computed with the neural network approximation is quite different to the one obtained with computer simulations. Indeed, the result obtained is similar to the one obtained with the HNC approximation. 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, 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 predicts the main peak well, as can be seen when compared to the results obtained from computer simulations.

1.5 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.

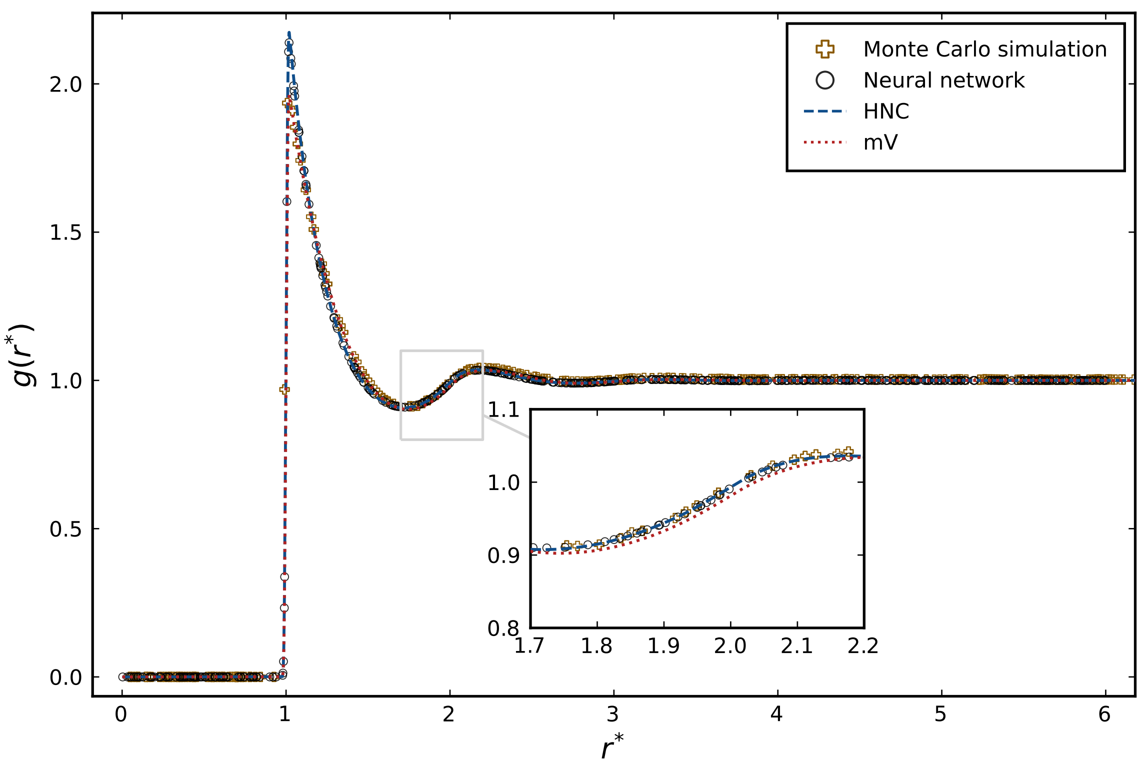

FIGURE 1.3: The radial distribution function for density value $\phi = 0.25$ obtained from computer simulations, and three different approximations: (a) *mV*, modified Verlet, (b) *HNC*, Hypernetted Chain, (c) *Neural network*, neural network approximation. Inset shows the region close to the peak about $r^* = 2$.

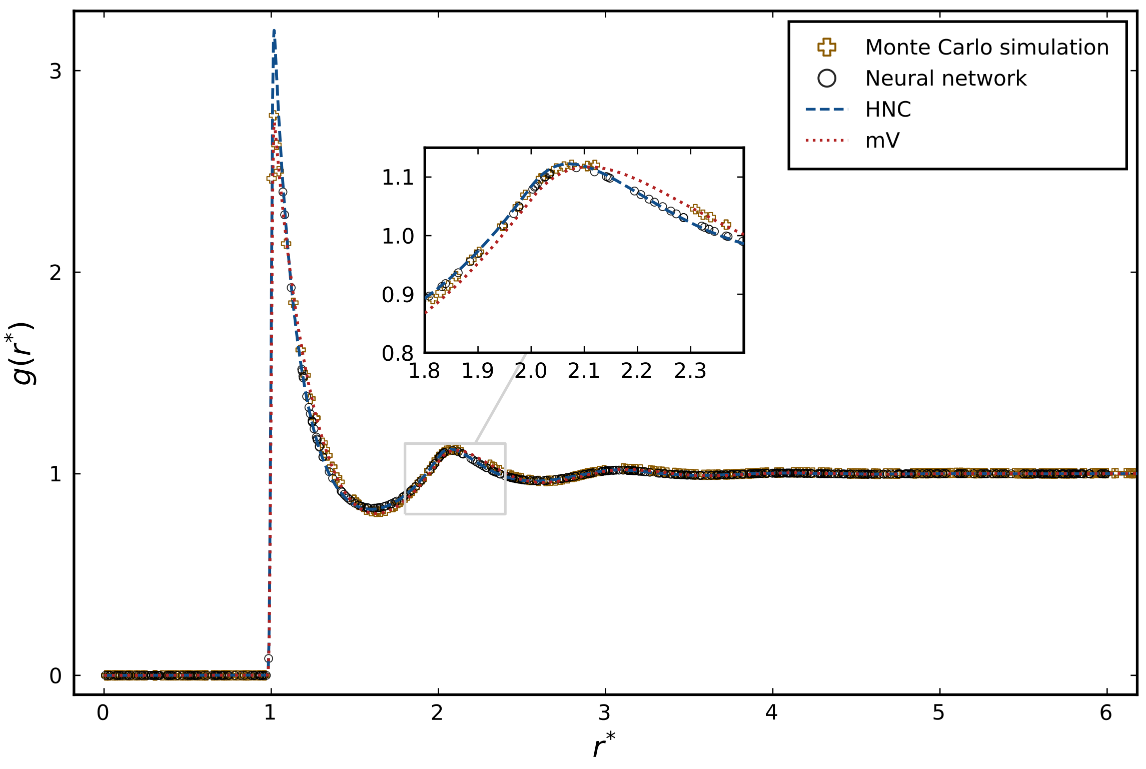

FIGURE 1.4: The radial distribution function for density value $\phi = 0.35$ obtained from computer simulations, and three different approximations: (a) *mV*, modified Verlet, (b) *HNC*, Hypernetted Chain, (c) *Neural network*, neural network approximation. Inset shows the region close to the peak about $r^* = 2$.

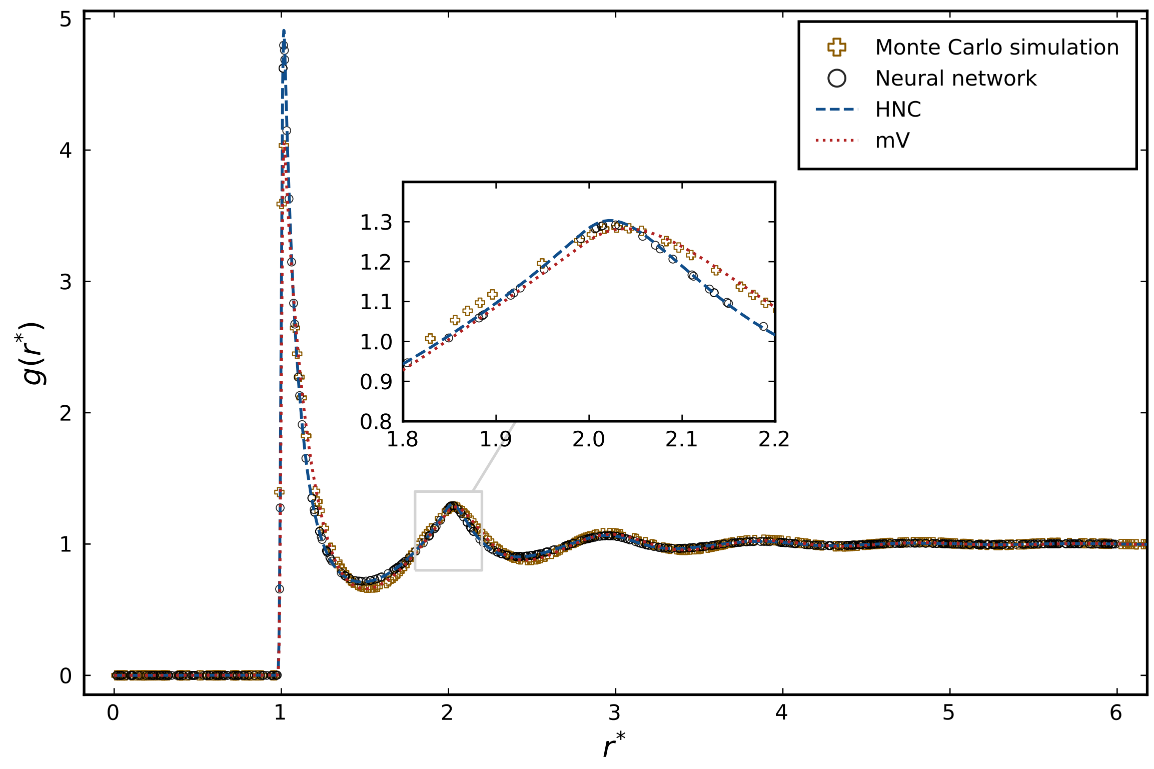

FIGURE 1.5: The radial distribution function for density value $\phi = 0.45$ obtained from computer simulations, and three different approximations: (a) *mV*, modified Verlet, (b) *HNC*, Hypernetted Chain, (c) *Neural network*, neural network approximation. Inset shows the region close to the peak about $r^* = 2$.

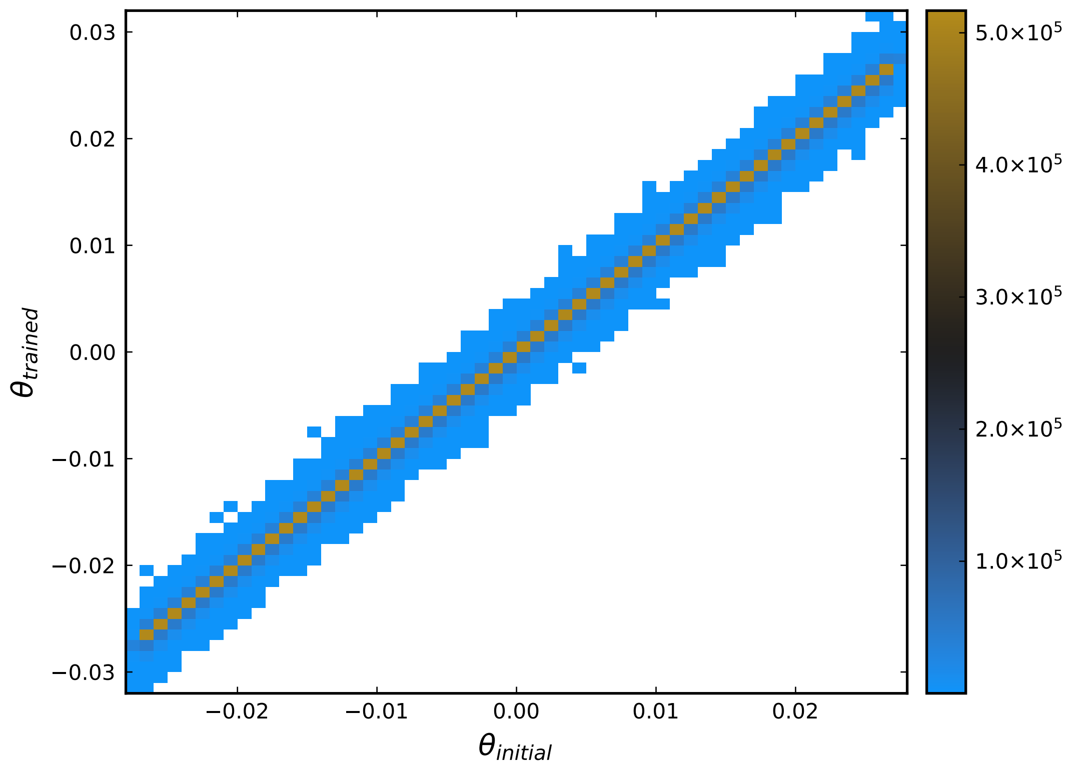

FIGURE 1.6: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for the density value $\phi = 0.15$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

1.5.1 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 each density value can be seen in figures (1.6, 1.7, 1.8, 1.9). 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 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 value for the density, the larger the variance is. If we observe the variance for the density $\phi = 0.15$ in Figure 1.6 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 1.9 for the density value of $\phi = 0.45$ we see 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, for the HNC is actually a solution to the OZ equation, and

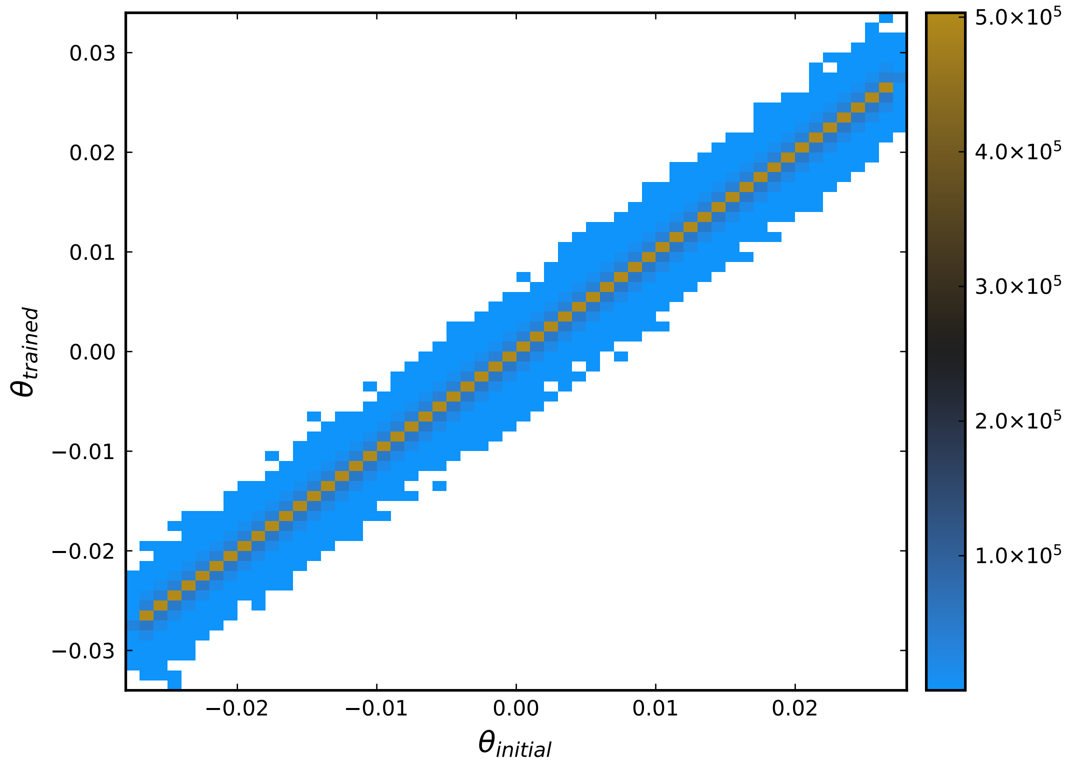

FIGURE 1.7: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for the density value $\phi = 0.25$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

solutions around this particular approximation might be as well solutions. This, however, does not answer the question of why the spread is larger when higher densities are inspected.

1.5.2 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 having reached a minimum already. We must recall that the weight update and neural network training is essentially an optimization problem (1.4), and the main goal is to find a minimum of the cost function (1.3). 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 large for higher density values. If we pay close attention to the approximation results for the *low density* values in item 1.2, we can see that although every approximation given a low accuracy estimation of the second peak as shown in the inset, for the main peak, the neural network approximation is very accurate. If we now observe the item 1.5 which refers to the *high density* value we can see that the estimation is quite poor. 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, there is

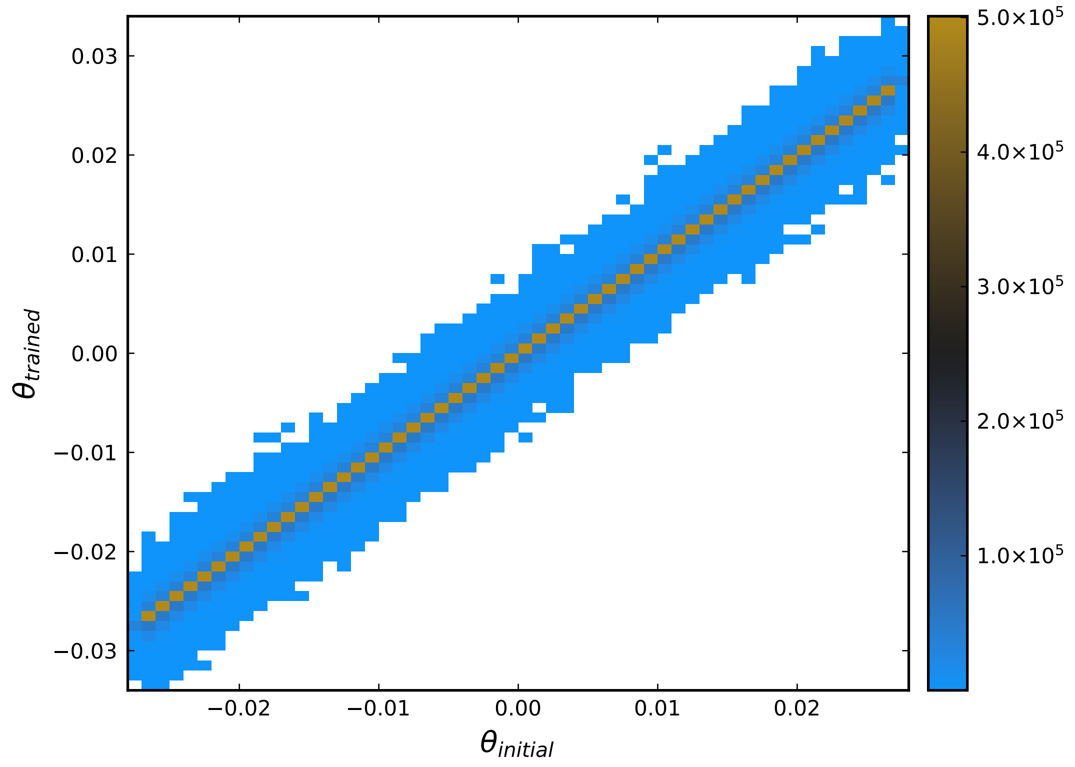

FIGURE 1.8: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for the density value $\phi = 0.35$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

now a *higher variance* in the weight evolution. This suggests that, for *lower density* values, there was no need to adjust the weights more than shown in Figure 1.6 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.

1.5.3 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 approximation fail to provide an accurate solution.

If, in fact, the neural network is 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 the 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, 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,

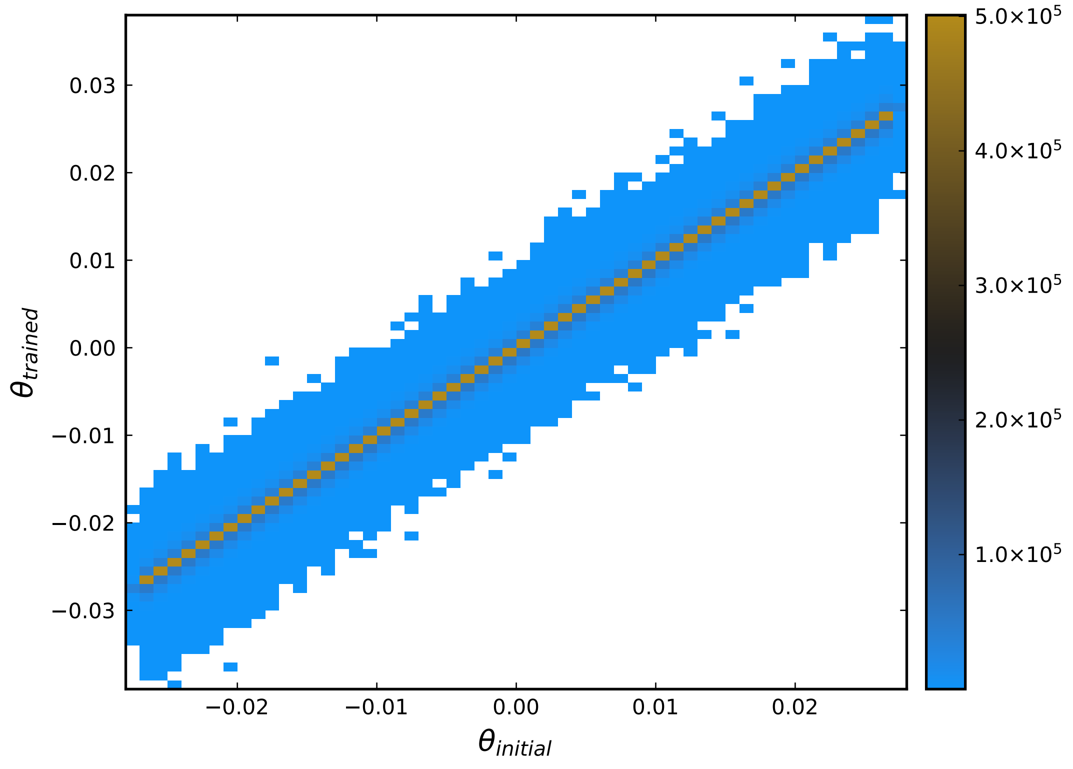

FIGURE 1.9: Relation between the trained weights and the initial weights of $N_\theta(\mathbf{r})$ for the density value $\phi = 0.45$. The scale on the right-hand side represents the total number of instances for the trained-initial pair of weights.

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 mean that adding nonlinearities according to some aspect of the interaction potential prove beneficial for the weight update and training dynamics. Secondly, the case where the *weights do not change*. In this 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, in either scenario we do not have the information to assess if the neural network might actually be *more precise* than it is in its current form.

1.5.4 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. 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]

$$b(\mathbf{r}) = b^{(2)}(\mathbf{r})\rho^2 + b^{(3)}(\mathbf{r})\rho^2 + \cdots \tag{1.8}$$

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 espcially important to note that the coefficients $b^{(n)}(\mathbf{r})$ are, in general, very 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. Only for special cases can these coefficients 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

$$p_n(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots + a_n z^n, \quad z \in \mathbb{C} \tag{1.9}$$

where the coefficients $a_0, a_1, \ldots, a_n$ are independent real-valued random variables with finite mean and finite variance. These 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 (1.9). 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 take in the weights and return the coefficients needed to build a power series, as needed. Further, if we compare directly Equation 1.9 and Equation 1.8, we can see that these expressions are related through their coefficients, implying that random variables might be able to give an answer to the n particle bridge functions in the power series. This insight might pave the way for an important way of computing coefficients by using neural networks, or related probabilistic methods. 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.

Appendix A

Gradient Computation

In chapter 1 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.

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 1.2. 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

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

where the cost function $J(\theta)$ is defined in Equation 1.3; ∇_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.

We are now interested in the closed form of ∇_θ. If we now use the definition of the cost function as defined in Equation 1.3, we have for the gradient

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

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 implicitly on the weights of the neural network. This is essential to note, because the fact that the dependence is implicit will have an important role in the development of the computations.

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, we must recall that an essential property of $\gamma_n(\mathbf{r};\theta)$ is its implicit dependency on the weights, thus 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 *does in fact depend* on the weights. In this case, 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 1.2 and differentiate it with respect to the weights

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

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

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

Now, in Equation A.3 we have the expression $\frac{\partial c(\mathbf{r};\theta)}{\partial\theta}$ but the function $c(\mathbf{r};\theta)$ does not depend explicitly on the weights, hence $\frac{\partial c(\mathbf{r};\theta)}{\partial\theta} = 0$.

Putting all this information together we are able to find an expression for $\partial_\theta \gamma_n(\mathbf{r};\theta)$. We take Equation A.3 together with Equation A.5, and we obtain the following expression

$$\frac{\partial c(\mathbf{r};\theta)}{\partial\theta} = 0 = e^{p(\mathbf{r},\theta)}\left[\partial_\theta\gamma(\mathbf{r};\theta) + \partial_\theta N_\theta(\mathbf{r})\right] - \partial_\theta\gamma(\mathbf{r};\theta) \tag{A.6}$$

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

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

$$\boxed{\partial_\theta\gamma(\mathbf{r};\theta) = \frac{e^{p(\mathbf{r},\theta)}\partial_\theta N_\theta(\mathbf{r})}{1 - e^{p(\mathbf{r},\theta)}}} \tag{A.7a}$$

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 by means of the derivative of $N_\theta(\mathbf{r})$ with respect to θ.

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.7a to obtain a closed form of the gradient we seek

$$\boxed{\nabla_\theta J(\theta) = 2\left[\gamma_n(\mathbf{r};\theta) - \gamma_{n-1}(\mathbf{r};\theta)\right]\left[\frac{e^{p_n(\mathbf{r},\theta)}\partial_\theta N_\theta(\mathbf{r})}{1 - e^{p_n(\mathbf{r},\theta)}} - \frac{e^{p_{n-1}(\mathbf{r},\theta)}\partial_\theta N_\theta(\mathbf{r})}{1 - e^{p_{n-1}(\mathbf{r},\theta)}}\right]} \tag{A.8}$$

with

$$p_n(\mathbf{r},\theta) = -\beta u(\mathbf{r}) + \gamma_n(\mathbf{r};\theta) + N_\theta(\mathbf{r}).$$

In practice, a smoothing factor of $\varepsilon = 1\times 10^{-7}$ was used in the denominator of the gradient expression from Equation A.8 to avoid division by zero. Also, the multiplication and division in the same expression were the Hadamard product and division respectively, i.e. elementwise product and division.

Appendix B

Numerical solution to the OZ equation

Some equations.

Bibliography

[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.

[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.

[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].

[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.