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 1, 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 appendix, the detailed 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

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

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, $\{\gamma_1(\mathbf{r}), \gamma_2(\mathbf{r}), \ldots, \gamma_n(\mathbf{r})\}$. 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

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

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

In other words, the expression (1.3) is requiring for the last two approximations 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. Tthis 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 (Ref a apéndice). 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 have the following step to solve the OZ using the parametrization (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. (Ref a apéndice)

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), and the computation of the gradient $\nabla_\theta J(\theta)$ is performed.

6. The weights θ are updated with the gradient descent rule (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

$$|\gamma_{n+1} - \gamma_n|^2 \leq \epsilon \tag{1.6}$$

where $|\cdot|$ indicates the absolute value, and $\epsilon \in 0 and 1$. In particular, the numerical tolerance in all the experiments has a value of $\epsilon = 1 \times 10^{-5}$. 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 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 instabilities rendered this method unstable and convergence was almost never achieved.

To solve this issue, the *Adam* [KB17] optimization method was then 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 zeroes. 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 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 *four layers*, all connected among them. There is an *input* layer, two *hidden* layers, and a final *output* layers. 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*.

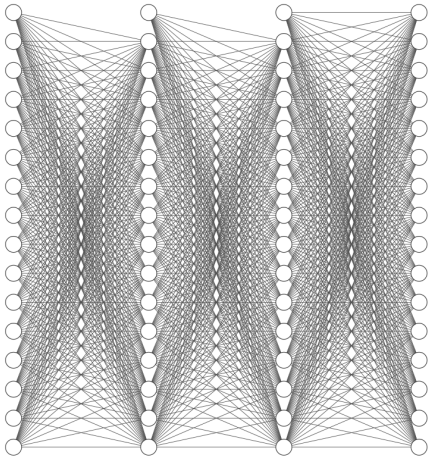

FIGURE 1.1: Cartoon of a fully connected multilayer neural network. Note that there are two *hidden layers*. The circles represent the *nodes* or *units* used to compute the final output. These nodes are using an activation function to account for nonlinearities. The top-most nodes that seem different 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 a way to help the convergence of neural networks. This means that the weights are initialized as $\theta_{ij} \sim U\left[-\frac{6}{\sqrt{(in+out)}}, \frac{6}{\sqrt{(in+out)}}\right]$, where 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.) generated numerical instabilities in the algorithm when training the neural network.

1.3.3 Physical parameters

To solve the OZ equation a cutoff radius of $r_c = 7\sigma$ was used, where σ is the particle diameter and it 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 of the computer simulations, where such partition is not needed.

1.4 Results

Now that all the elements needed have been described in previous sections, all the obtained results in this chapter are shown in figure 1.2, which correspond to the radial distribution functions, $g(r^*)$, of the densities studied. These functions describe de structure of the liquid, and are the main talking point of the application from the methodology developed in this work.

1.4.1 Low densities

To begin the discussion of the results obtained, this section will deal with the low density values $\phi = 0.15 and 0.25$, which are shown in plots (a) and (b) in figure 1.2. The results show that, at low densities, the HNC and neural network approximations are more precise than the Percus-Yevick and modified Verlet approximations. Although, all approximations seem to fall short compared to computer simulations. It is specially important to note that the neural network approximation is a little bit more precise than the HNC approximation, which can be quantitatively measured by the estimation of the principal peak in the plot. This peak can be found in the vicinity of $r^* = \sigma$. Nevertheless, it is still overestimated, which is the same case for the HNC approximation. However, this is not the case for the Percus-Yevick and modified Verlet approximations, 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 both approximations, Percus-Yevick and modified Verlet, are closer to the form found in the computer simulations results.

1.4.2 High densities

We now turn our discussion to the high density values of $\phi = 0.35 and 0.45$, represented in plots (c) and (d) in the figure 1.2. 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 Percus-Yevick and modified Verlet are even more precise, which was expected. This is because the HNC approximation is a very good approximation for long range interaction potentials (Ref faltante), and Percus-Yevick as well as modified Verlet are 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 figure 1.2, where the main peak is well estimated by the 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 Percus-Yevick and modified Verlet

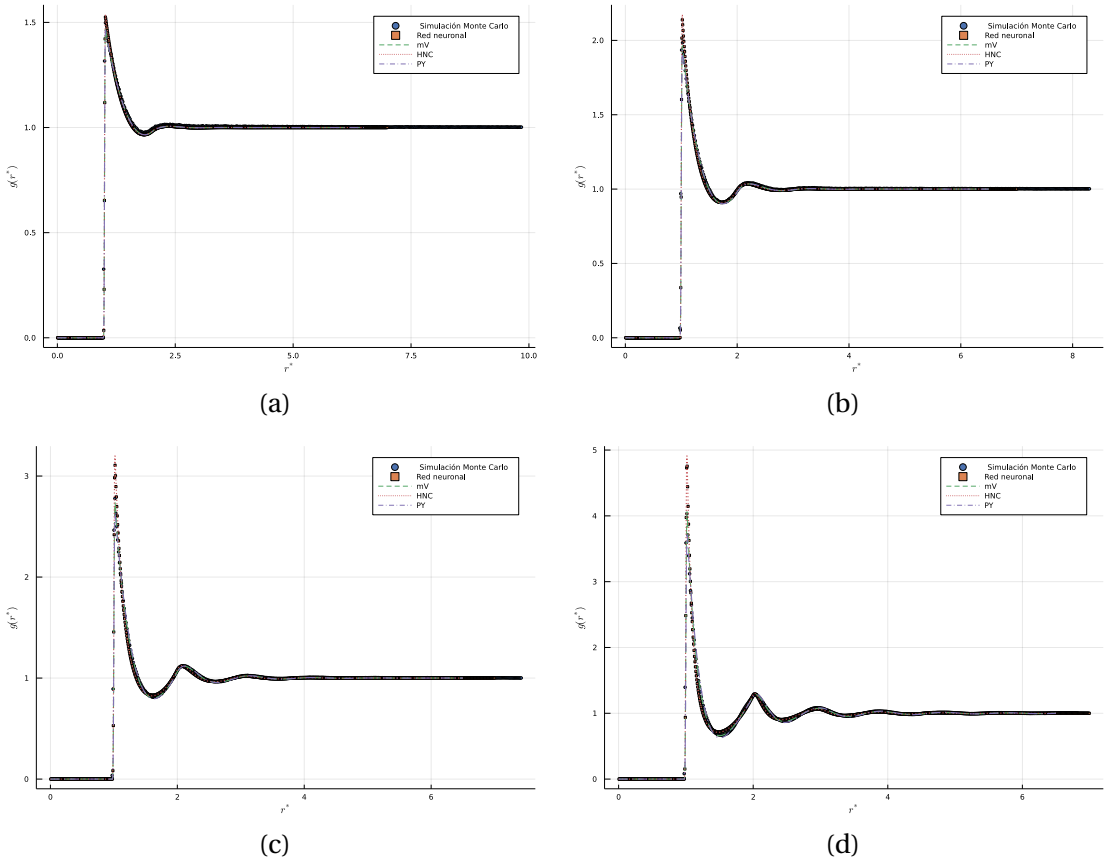

FIGURE 1.2: Radial distribution functions obtained from computer simulations and with the methodology describes in this chapter, with the neural network parametrization. Four different density values are presented: (a) $\phi = 0.15$, (b) $\phi = 0.25$, (c) $\phi = 0.35$, (d) $\phi = 0.45$. In each figure, a comparison between three common bridge function approximation appear along with the computer simulation results and the neural network approximation. The approximations are *mV*, modified Verlet; *PY*, Percus-Yevick; y *HNC*, and Hypernetted Chain.

approximations, 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.

1.4.3 Does the neural network approximation reduce to the HNC bridge function?

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.

One way to approach this is to investigate 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 can be seen in figure (Ref a figura). 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 standard deviation σ.

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.

[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.

[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.

[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.

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

[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.