UNIVERSITY OF OVIEDO

FACULTY OF CHEMISTRY

THEORETICAL AND COMPUTATIONAL CHEMISTRY GROUP

Master's thesis:

Development of a Neural Network model to predict Atoms in Molecules properties in simple biosystems

In order to obtain the qualification in the Master's degree in

THEORETICAL CHEMISTRY AND COMPUTATIONAL MODELLING

Presented by Alejandro Luis García Muñoz

Supervised by Ángel Martín Pendás

June 2024

Contents

Acknowledgements // Agradecimientos

En primer lugar, quiero dar las gracias a los dos pilares fundamentales que han hecho posible que pueda sacar adelante un proyecto como este. Con Ángel poniéndome firme en la parte de Química Teórica y Miguel guiándome más por la de Inteligencia Artificial, ambos transmitiéndome todo lo que saben con paciencia, dedicación y, dada mi semipresencialidad, además en tiempos récord. Agradezco vuestro apoyo, y la de todo el grupo de Oviedo, del primero al último. Ya tengo una parada obligatoria cuando pase por el norte.

En segundo lugar, papá, mamá, quiero que sepáis que esto también es gracias a vosotros. No estaría donde estoy sin esos valores que tanto os definen y de los que espero que con tanto tiempo juntos se me haya pegado alguno. Al mismo nivel estás, Dani, que para ser el hermano pequeño me demuestras una y otra vez que me queda mucho por aprender de ti, y estaré encantado de seguir haciéndolo toda la vida.

Si a alguien más le debo mucho, y no solo de este proyecto, es a ti Bea. Me has enseñado y enseñas que a distancia también se pueden construir (y "déclencher") cosas muy bonitas.

Ya por último, me gustaría dar las gracias a todos los buenos amigos que me llevo de este máster. Empezando con el grupo de Salamanca, que me acogió como a uno más. María, sigo esperando a que me des luz verde para que dejemos la Química Teórica y montemos el bar de "nuestros" (mis) sueños. Víctor, quiero que sepas que no me he vuelto a morder las uñas gracias a ti. Cosmin, nuestras madres lo predijeron y aquí estamos, tú viviendo en la capital y yo de okupa robándote comida (y dejándome algo, eso seguro). Santi, o tienes mentes de cobre o no sé cómo haces para saber de todo y tanto, vas a llegar lejos y espero estar para verlo.

Por motivos de copyright no puedo decir el nombre del siguiente grupo, pero sí el de sus integrantes. José, compañero filosofal, al final me vas a convencer de que Murcia existe, ¿o de qué en realidad nada existe? Y Valentín, el inventor de la espontaneidad, armado con unos zapatos de 5-Dedos y un puñado de canicas, el mundo será tuyo.

Raúl, has nacido para crear, desde canciones hasta lenguas, y encantado estaré de saber de ellas en nuestras llamadas a las 8. Unai, mejor dicho, "United States of my Heart", vete calentando que en nada subo a echarte una "euskal pilota". Josep, gracias por ser así de auténtico y, obviamente, por las pipas de calabaza. Pau, maestro, gracias por tu humor y sabiduría. Y Carlos, Xevi, aunque haya sido brevemente, gracias por cruzaros en mi camino, ¡y hacerlo más bonito!.

Abstract

In the Quantum Theory of Atoms in Molecules (QTAIM), the localization index (LI) allows us to know how many non-bonding electrons each atom in a chemical system has, which is very useful to reveal information about bonding and shell structure. Nevertheless, the computational cost of predicting a QTAIM property scales with the number of atoms, making it difficult to obtain results for very large systems or for many geometries in reasonable timescales.

In order to address this problem, a fully flexible architecture to develop and use Neural Network models for the prediction of chemical properties, called NNAIMGUI, was developed in Oviedo. It comes with a built-in model, called NNAIMQ, for partial charge (q) prediction but any other model will be created to predict different (usually QTAIM) properties.

For these reasons, this Master's thesis focuses mainly on the development of a Feed-Forward Neural Network (FFNN) model for the prediction of the localization index. To do so, a dataset composed of different geometries of several amino acids is used, generated under the QTAIM alongside Density Functional Theory (DFT) to obtain the wavefunction data. Additionally, both the NNAIMQ model and the model developed in this project is validated with the aforementioned geometries and the latter applied to a few simple biosystems.

Acronyms and abbreviations

QTAIM	Quantum Theory of Atoms in Molecules
DFT	Density Functional Theory
ACSF	Atom-Centered Symmetry Functions
AE	Absolute Error
MAE	Mean Absolute Error
MSE	Mean Square Error
RMSE	Root Mean Square Error
FFNN	Feed-Forward Neural Network
LI	Localization Index
MP	Møller-Plesset
CI	Configuration Interaction
CC	Coupled Cluster
LDA	Local Density Approximation
GGA	Generalized Gradient Approximation
m-GGA	Meta Generalized Gradient Approximation
GD	Gradient Descent
SGD	Stochastic Gradient Descent
AdaGrad	Adaptative Gradient Descent
RMSProp	Root Mean Square Propagation
Adam	Adaptative Moment Estimation
tanh	Hyperbolic tangent
ReLU	Rectified Linear Unit
SELU	Scaled Exponential Linear Unit
NMS	Normal Mode Sampling
MD	Molecular Dynamics
GMM	Gaussian Mixture Model
NNA	Non-Nuclear Atractors
PCA	Principal Component Analysis

1 Objectives

The purposes of this work can be divided into two groups:

- **Scientific objectives of the thesis:**

  In this work, by merging the Theoretical Chemistry and Artificial Intelligence disciplines, a code for developing and/or running Neural Network models, called NNAIMGUI, was used in order to cover three main objectives.

  The first one consists of re-validating a pre-built model in NNAIMGUI, called NNAIMQ, created for the prediction of partial charges (a QTAIM property). The second one consists of developing with this aforementioned code, a new model for the prediction of the localization index (another well-known property in QTAIM). It is important to highlight that for these first two objectives it will be necessary to create a database of diverse geometries of biochemical interest, as well as to encode, for the second objective, its different coordinates in symmetry functions interpretable by neural networks.

  Finally, as a third objective, in order to show a practical example of this tool and its possible limitations, the created model was applied in new bio-systems (not included in the database), comparing its predictions for the localization index with the results obtained with QTAIM.

- **Implementation of skills covered in the master's degree:**

  During this project, I applied various skills acquired throughout my master's degree. These include the use of *Bash* scripting (from Linux and Linux System Management course) and *Python3* with a focus on Deep Learning (from the Multiscale, Machine Learning, and QSAR course). I also applied Density Functional Theory using *Gaussian* software (from multiple subjects) and the Quantum Theory of Atoms in Molecules using the *AIMALL* package (from the Advanced Methods in Electronic Structure course). Additionally, I performed Molecular Dynamics on simple biosystems (from Chemical Dynamics and Computational Biochemistry courses) and structured the project and managed the bibliography (from Transversal Scientific and Linguistic Competence course).

2 Introduction

With the advent of the information age[1] (also known as digital revolution), a paradigm shift occurred at the scientific level: the first advances in computing made it possible to solve problems that were previously unapproachable. For instance, Internet meant the appearance of an area for consultation, discussion and scientific communication, opening up access to a wide variety of knowledge.[2]

Today we are living in a "new digital age": artificial intelligence has begun to enable problem-solving and optimization of their solutions by ways that emulate the functioning of the human brain.[3] Now, in practically any area, the use of, for example, machine learning tools makes possible to carry out diverse tasks with little cost in terms of time and resources.[4]

By this, it is undeniable that we live in a digitalized world and, like it, science has also evolved alongside the technology. Computational tools make possible to obtain remarkably accurate predictions of physical and chemical properties in agreement with experimental data,[5] and can even predict results that are not yet experimentally demonstrable.

On the other side and also in connection with the project presented, Quantum Mechanics, hand-in-hand with computation, has made decisive contributions to all branches of science. As examples of its interdisciplinary character, different contributions of this field to various areas will be mentioned. Starting in physics where, for example, it explained phenomena like the tunnel effect[6] and superconductivity[7]. In chemistry, quantum mechanics has contributed to the development of structural determination methods such as nuclear magnetic resonance (NMR)[8], protein structure prediction[9] and also the design of nanomaterials[10]. Moreover, it has revolutionized medicine and biology through the development of microscopes based on quantum effects[11], innovative diagnostic methods with the application of artificial intelligence[12] and quantum dots for the diagnostic and treatment of cancer[13]. In electronics, quantum effects are fundamental to explaining the behaviour of the semiconductors[14] that form the basis of traditional computers, not to mention the design of quantum computers, which will revolutionize the world of computing within a few years.[15]

More specifically, in the quantum chemistry area, computation is a key element in the study

of molecular structure and dynamics.[16] Thus, Computational Modeling has led to numerous achievements by simulating physical systems and extracting information through complex calculations.[17]. In order to find a balance between accuracy and computational cost, in the theoretical chemistry area, the density functional theory (DFT) emerged, where the entire system is constructed exclusively from its charge distribution in space.

For large molecules, such as biosystems, models with even lower computational costs can be advantageous. Thus, semi-empirical methods[18] were developed, which reduce computational costs by incorporating pre-existing data without sacrificing excessive accuracy. However, the current digital era enables the use of Artificial Intelligence in computational modeling[19]. For example, a neural network can be trained with data from previously studied systems to generate a model capable of predicting properties in new systems.

In this Master's thesis, a code developed for this purpose was used for the implementation of the Quantum Theory of Atoms in Molecules (QTAIM): a theory widely used in quantum chemistry that allows the prediction of molecular properties from their atomic contributions. Under the name of *NNAIMGUI*[20], this Python code allows the training and use of neural networks for the prediction of atoms in molecules properties, as well as their visualization. Moreover, a built-in FFNN model, called NNAIMQ, is available for the prediction of QTAIM partial charges (individual charges of each atom under this theory). The usefulness and accuracy of this model has been demonstrated in different systems.[21] In this project its re-validation will be performed and a new FFNN model for the prediction of Localization Index, an important property in the Atoms in Molecules approach, will be developed. Additionally, examples of applying the created model to simple biosystems will be provided to demonstrate the potential of Artificial Intelligence in Computational Modeling.

3 Quantum Chemistry

This chapter describes the concepts of Quantum Mechanics and Theoretical Chemistry used in this project, starting with the birth of Quantum Mechanics (Section 3.1) and its relation with computation (Section 3.2), followed by the Hartree-Fock method (Section 3.3). Finally, the Density Functional Theory (Section 3.4) and the Quantum Theory of Atoms in Molecules (Section 3.5) are explained as the chemical cornerstones of this thesis.

3.1. The origin of Quantum Mechanics

Since Newton, classical physics seemed to be able to explain all phenomena without any inconsistencies, remained consistent in its entirety. And it persisted like this until the end of the 19th century. With the so-called "ultraviolet catastrophe"[22] in the study of the black body, discrepancies appeared between the theoretical model and the experiments.

The solution came from M. Planck, who proposed that energy is exchanged in discrete amounts (quanta), thus solving the black body problem. A. Einstein further extended this idea by proposing the concept of light quanta (later called photons) to explain the photoelectric effect, winning the Nobel Prize in 1921.[23] Building on these concepts, L. de Broglie proposed the total wave-corpuscle duality[24], applicable to all the matter. Moreover, such was the influence of this new theory that even the basis of the actual atomic model was established. E. Schödinger proposed an atomic model that adequately predicted the lines of the emission spectra of neutral and ionized atoms.[25]

Gradually, throughout the 20th century this "theory of the quantum" gained prominence in the scientific community. It became the central theme of the *V Solvay Congress* (1927)[26], which gathered the greatest scientists of the time. Over time, it evolved to the present day, where quantum mechanics has entirely replaced the old classical view in the explanation of phenomena at the microscopic level.

3.2. Computational chemistry

Over a century after its origin, quantum mechanics continues to grow and influence in an interdisciplinary way. From the postulates of quantum mechanics, it is possible to conceive a wavefunction that contains all the information obtainable from the system. For this purpose, the wavefunction must fulfill several characteristics. For instance, it has to be normalized in order to ensuring that the total probability of finding particles in all states adds up to one. Also, for fermions like electrons, the wavefunction must be antisymmetric under particle exchange, following the Pauli exclusion principle[1]. Fulfilling these and more characteristics, by applying specific operators in this wavefuncion, the value of the desired observable is extracted from it. An example of this is the "energy operator", the Hamiltonian, which permits to obtain the energy of a given system from its wavefunction by solving the famous Schrödinger equation[27]

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle \tag{3.1}$$

where $\hat{H}$ is the Hamiltonian, $|\psi\rangle$ in the system wavefunction and E is the energy of the system.

But it is not that easy: the equations to be solved are not trivial and beyond hydrogen-like atoms are even intractable. For this reason, quantum chemistry is currently going hand in hand with computation. In particular, in the area of quantum chemistry, through computation, it has been possible to develop packages capable of simulating quantum systems, based on theoretical chemistry, from which information can be extracted. This field of study is called computational chemistry.[28]

Nevertheless, in order to be able to deal with systems larger than the above-mentioned hydrogen-like atoms, additional approximations are required. Within the methods developed in theoretical chemistry for this purpose, the first and probably the best known is the so-called "Hartree-Fock" method, which will be briefly explained in the next section.

[1]Pauli exclusion principle: two electrons cannot simultaneously occupy the same quantum state (have same quantum numbers)

3.3. The Hartree-Fock method

In order to be able to solve the Scrhödinger equation for many-body systems several approximations are necessary. The first, in chronological order, is the Born-Oppenheimer approximation. This approach assumes that due to the large mass difference between the nuclei and the electrons, the nuclei can be considered fixed while solving the Schrödinger equation for the electrons. This means that the wave function can be separated into electric and nuclear components as follows:

$$\psi(r, R) = \psi_e(r, R) \cdot \psi_N(R) \tag{3.2}$$

where r denotes distances electron-electron or electron-nucleus, R denotes distances between nuclei, $\psi_e(r, R)$ is the electronic wavefunction and $\psi_N(R)$ is the nuclear wavefunction. These two components can be associated to the following two terms of the Hamiltonian

$$\hat{H} = \hat{H}_e + \hat{T}_N \tag{3.3}$$

where $\hat{H}_e$ is the electronic Hamiltonian and $\hat{T}_N$ is the nuclear kinetic energy operator. The latter term is neglected due to the approximation, meanwhile the first term can be expressed as

$$\hat{H}_e = \hat{T}_e + \hat{V}_{eN} + \hat{V}_{ee} + \hat{V}_{NN} \tag{3.4}$$

where $\hat{T}_e$ refers to the kinetic energy of the electrons, $\hat{V}_{eN}$ to the electron-nucleus attractions, $\hat{V}_{ee}$ to the electron-electron repulsion and $\hat{V}_{NN}$ to the nucleus-nucleus repulsion, term typically not considered in the electronic Hamiltonian because it is a constant that does not influence the eigenfunctions. Thus, the Time-Independent Schrödinger Equation of the electronic part is expressed as

$$\hat{H}_e \psi_e = E_e \psi_e \tag{3.5}$$

where E_e is the electronic energy. Finally, the total energy of the system can be obtained adding the nucleus-nucleus repulsion contribution as follows

$$E = E_e + \sum_{A=1}^{M} \sum_{B>A}^{M} \frac{Z_A \cdot Z_B}{R_{AB}} \tag{3.6}$$

where M is the total number of atoms and Z_i is the atomic number of each atom.

Nevertheless, the 3.5 equation must still be simplified to deal with multi-electron systems. For

this reason Hartree proposed the following: approximate the total system wave function (Ψ) as a product of single-electron wave functions (ψ) as follows

$$\Psi(r_1, r_2, ..., r_N) = \psi(r_1) \cdot \psi(r_2) \dots \psi(r_N) \tag{3.7}$$

where each r_i represents the spatial coordinates of each 'i' electron. This approach, known as the Hartree method (or Hartree product), simplifies the problem to a set of N coupled mono-electron equations. However, this approach had a major problem: it did not consider the Pauli exclusion principle, which is essential for electron representation. As a solution, Fock and other scientists improved this method by introducing the concept of the Slater determinant, which did include the Pauli exclusion principle. This is called the Hartee-Fock approximation[29], and in practical terms, it implies that for each nuclear geometry, the electronic wavefunction can be written as a single Slater determinant as follows

$$\Psi_e = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(1) & \chi_2(1) & \dots & \chi_n(1) \\ \chi_1(2) & \chi_2(2) & \dots & \chi_n(2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(n) & \chi_2(n) & \dots & \chi_n(n) \end{vmatrix} \tag{3.8}$$

where x_i represents the spin and the spatial coordinates of each 'i' electron, N is the total number of electrons and χ_i represents the spin-orbital formed by the product of a mono-electronic space wavefunction and a spin function (α or β). It is noteworthy that, over this approximation, for the description of molecules, another approximation is usually applied: LCAO (linear combination of atomic orbitals) approximation[30].

The main limitation of the Hartree-Fock methodology is the absence[2] of electron correlation description, since approximating the interactions of all electrons influencing each other with an average potential can be, in some cases, a too crude approximation of the reality. Indeed, the correlation energy (E_c) can be calculated as

$$E_c = E_{exact} - E_{HF} \tag{3.9}$$

where E_{HF} in the Hartree-Fock energy and E_{exact} is the exact energy of the multi-electron system (obtained theoretically or experimentally).

[2]The correlation can be considered non-null since the Hartree-Fock method includes the Fermi correlation (or exchange correlation) through the Pauli exclusion principle.

For this reason, in order to include the electronic correlation in the calculations, post-Hartree-Fock methods emerged[31]. Møller-Plesset Perturbation Theory[32] (MP2, MP3, ...), Configuration Interaction[33] (CI) or Coupled Cluster[34] (CC) are some examples, but the chosen method will depend on the system and the property to be studied. Nevertheless, they are not discussed here because they are beyond the scope of this work. Instead, in the following two sections, the theories that form the basis of the project will be introduced. Starting with the Density Functional Theory and continuing with the Quantum Theory of Atoms In Molecules.

3.4. Density Functional Theory (DFT)

As an alternative to traditional ab initio methods (i.e. methods only based in fundamental principles, without experimental parameters) and in order to address the computational and scalability limitations associated with larger and more complex molecular systems, Density Functional Theory emerged.[35]

Under this approach, the aim is to obtain solutions to the Schorondinger equation based solely on the electron density $\rho(r)$. In this way, the properties of the chemical system can be obtained through an electron density functional. In DFT, after the Born-Oppenheimer approximation, two theorems are the basis for the theory: the Hohenberg-Kohn theorems.

3.4.1. Hohenberg-Kohn theorems

The first theorem states that the energy of a (non-degenerate) ground state of a multi-electron quantum system can be computed exactly from the electron density of such a ground state. Moreover, it is one and only one electron density that minimizes the total energy of the system, and this density contains all the necessary information about the system (as does the wavefunction in Quantum Mechanics normally). Thus, the energy of the system as a function of the density can be expressed as

$$\langle \psi_0 | \hat{H} | \psi_0 \rangle = E[\rho] = F[\rho] + \int V_{ext}(\vec{r}) \rho(\vec{r}) \cdot d\vec{r} \tag{3.10}$$

where $E[\rho]$ is the energy functional, $F[\rho]$ is an universal term that includes the functionals related to the kinetic energy and the electron-electron interaction, $V_{ext}(\vec{r})$ is the external potential and $\rho(\vec{r})$ is the electron density.

The great advantage of this theorem is that it makes it possible to express the energy of the system in terms of three spatial coordinates only, instead of depending on N electrons (3N spatial coordinates).

Regarding the second theorem, it starts by employing the variational principle which states that in the ground state of the system, a new wavefunction ψ_0' with Hamiltonian $\hat{H}'$ it is satisfied that

$$\langle \psi_0' | \hat{H} | \psi_0' \rangle \geq \langle \psi_0 | \hat{H} | \psi_0 \rangle \tag{3.11}$$

where both terms are equals only if $\psi_0' = \psi_0$ i.e. if $\hat{H}' = \hat{H}$. And if on this inequality the first theorem is applied, the following expression is obtained

$$F[\rho'] + \int V_{ext}(\vec{r})\rho'(\vec{r}) \cdot d\vec{r} = E[\rho'] \geq E[\rho] \tag{3.12}$$

where both terms are equals only if $\rho' = \rho$. The latter inequality is known as the second Hohenberg-Kohn theorem, and establishes that the minimum energy is obtained when the electron density is equal to the ground state density associated to the external potential ($V_{ext}(\vec{r})$).

Until now, this theory does not seem to have any apparent problems, being able to simplify the solution of the Schrodinger equation in a precise way. Unfortunately, there is one major problem which is the main weakness of this theory and will be addressed immediately below.

3.4.2. Kohn-Sham equations

The downside of DFT is that the exact value of the universal functional $F[\rho]$ is unknown. To deal with it, Kohn and Sham propose a set of equations to face this problem. The main idea is to approximate the real system with another one of equal electron density composed of non-interacting electrons (as in the Hartree-Fock approximation). The starting point is the expression of the functional for the real system

$$F[\rho] = T[\rho] + V_{ee}[\rho] \tag{3.13}$$

where $T[\rho]$ is the kinetic energy and $V_{ee}[\rho]$ represents the electron-electron interaction. Here, as the approximated system considers non-interacting electrons, this new kinetic energy ($T_s[\rho]$) can be divided into the kinetic energies of each electron. On the other hand, in the electron-electron interaction its coulombic contribution can be defined as $J[\rho]$. Since these last two

terms can be known, they are introduced into the above equation (by adding and subtracting them) in order to isolate the unknown terms together as follows

$$\begin{aligned} F[\rho] &= T[\rho] + V_{ee}[\rho] + T_s[\rho] - T_s[\rho] + J[\rho] - J[\rho] \\ &= T_s[\rho] + J[\rho] + [(T[\rho] - T_s[\rho]) + (V_{ee}[\rho] - J[\rho])] \\ &= T_s[\rho] + J[\rho] + E_{XC} \end{aligned} \tag{3.14}$$

where $E_{XC}[\rho]$ is defined as the exchange-correlation energy functional which reflects the quantum interactions of the system (including the before-mentioned unknown terms). Taking into account the definition of this functional, which is

$$E_{XC}[\rho] = (T[\rho] - T_s[\rho]) + (V_{ee}[\rho] - J[\rho]) \tag{3.15}$$

the E_{XC} includes the following contributions: electron exchange, electron correlation, correction of the kinetic term (energy difference between T and T_s) and correction of the self-interaction induced by the classical Coulomb potential ($V_{ee} - E_H$).

On this new formalism, applying the Hohenberg-Kohn theorems, the following expression is obtained

$$\frac{\delta E[\rho]}{\delta \rho} = \frac{\delta T_s[\rho]}{\delta \rho} + V_{eff} \tag{3.16}$$

where

$$V_{eff} = V_{ext} + \frac{\delta J[\rho]}{\delta \rho} + \frac{\delta E_{XC}[\rho]}{\delta \rho} = V_{ext} + V_J + V_{XC} \tag{3.17}$$

where V_J is the Coulombic potential (the J derivative) and V_{XC} is the exchange-correlation potential (the E_{XC} derivative). In addition, δ represents a functional derivative, a generalization of derivatives applied to functions.

In order to express the equation (3.16) in terms of orbitals, it is multiplied on both sides by $\frac{\delta \rho}{\delta \phi_i^*(\vec{r})}$ applying the chain rule in the following way

$$\frac{\delta E[\rho]}{\delta \rho} \cdot \frac{\delta \rho}{\delta \phi_i^*(\vec{r})} = \frac{\delta T_s[\rho]}{\delta \rho} \cdot \frac{\delta \rho}{\delta \phi_i^*(\vec{r})} + V_{eff} \cdot \frac{\delta \rho}{\delta \phi_i^*(\vec{r})} \tag{3.18}$$

$$\frac{\delta E[\rho]}{\delta \phi_i^*(\vec{r})} = \frac{\delta T_s[\rho]}{\delta \phi_i^*(\vec{r})} + V_{eff} \cdot \frac{\delta \rho}{\delta \phi_i^*(\vec{r})} = \varepsilon \phi_i(\vec{r}) \tag{3.19}$$

$$\left(-\frac{1}{2}\nabla^2 + V_{eff}\right) = \boxed{\hat{h}_{KS}\phi_i(\vec{r}) = \varepsilon_i \phi_i(\vec{r})} \tag{3.20}$$

where $\hat{h}_{KS}$ is the Kohn-Sham Hamiltonian, $\phi_i(\vec{r})$ are the Kohn-Sham orbitals and ε_i are the energies of these orbitals.

It seems that these equations can now be solved exactly, but there is still one term that cannot be known in the construction of the effective potential, the exchange-correlation contribution.

3.4.3. Exchange-correlation functional

In the different ways of approaching the exchange-correlation term, the families of DFT functionals currently available have been created. The strategies used are very varied, but the most important ones will be discussed below (without entering in depth).

Local Density Approximation (LDA)

The LDA approach assumes that the exchange-correlation term depends only on the density at a given point in the space, i.e. the local electron density. This generates the following exchange-correlation functional

$$E_{XC}^{LDA}[\rho] = \int \rho(r)\epsilon_{XC}\rho(r) \cdot dr = E_X[\rho] + E_C[\rho] \tag{3.21}$$

where ϵ_{XC} represents the exchange-correlation energy per particle.

Although simple and computationally efficient, it may not be accurate enough for strongly correlated systems, such as highly covalent bonds or strong electronic interactions.

Generalized Gradient Approximation (GGA)

The GGA functionals incorporate additional information about the gradient of the electron density ($\nabla\rho$) at each point in space. This allows to take into account how the electron density changes along the space, so it is considered a semi-local functionals. The general structure of the exchange-correlation term can be expressed as

$$E_{XC}^{GGA}[\rho] = \int f(\rho, |\nabla\rho|)dr \tag{3.22}$$

Some well-known GGA exchange-correlation functionals are PW91-PW91[36], PBE-PBE[37] or B-LYP[38].

Meta Generalized Gradient Approximation (m-GGA)

These functionals, over the GGA methodology includes extra information of two types: second order derivatives of the electron density ($\nabla^2\rho$) and kinetic energy density (τ). Therefore, the m-GGA functionals general structure can be written as:

$$E_{XC}^{m-GGA}[\rho] = \int f(\rho, \nabla\rho, \nabla^2\rho, \tau)dr \tag{3.23}$$

Examples of these functionals are TPSS[39] or M06-L[40].

Hybrid functionals

In DFT, hybrid functionals are so named because they include in its exchange term a portion of the exchange from the Hartree-Fock approximation. It should be mentioned that this proportion is carefully adjusted, for example, by high-level calculations or experimental data (opening the discussion of how far DFT is a purely ab-initio method).

Among the approximations mentioned for the exchange-correlation functional, this one presents the DFT functionals most widely used nowadays for their proven superiority in the so-called "Jacob's ladder"[41]: a metaphorical stairway where the hierarchy of chemical accuracy is represented by its rungs and where the top is the absolute accuracy. The general expression for the hybrid exchange-correlation functional is

$$E_{XC}^{hyb} = \frac{X}{100}E_X^{HF} + \left(1 - \frac{X}{100}\right)E_X^{DFT} + E_C^{DFT} \tag{3.24}$$

Of all the options, among the most commonly used hybrid functionals are the hybrid-GGA B3LYP[42] and the meta-hybrid-GGA M06-2X[43].

Specifically, the latter functional was the one used for this project, due to its proven accuracy and consistency, being also the reference method for the training of NNAIMQ. In M06-2X the Hartree Fock portion, i.e. the X in the equation (3.24), is 54%.

3.5. The Quantum Theory of Atoms In Molecules (QTAIM)

Taking as a cornerstone the electron density of the system but with a different approach to DFT, the Quantum Theory of Atoms in Molecules (QTAIM) emerged. This theory focuses

on an analysis of the wavefunction for its chemical interpretation but, and here lies one of its great advantages, from a fundamentally atomic perspective, managing to construct molecular properties from their individual atomic contributions.[44; 45] This approach is a non-referential method, because instead of relying on pre-established models or empirical parameters, QTAIM is based solely on the electron distribution in the molecule along its bond critical points.

In order to understand what these critical points are and how the partitioning of space is carried out, a brief introduction to Quantum Chemical Topology is necessary.

3.5.1. Electron density topology

Considering the Born-Oppenheimer approximation, the stationary density can be defined as

$$\rho(r;R) = N \sum_{s_1} \int \cdots \int \psi^*(x1,...,x_N;R)\psi(x1,...,x_N;R) \cdot dx_2 \cdots dx_N \tag{3.25}$$

where s_1 is the spin coordinate of the electron 1, x_i are the spacial and spin cordinates of each electron and R_i its nuclear coordinates.

Working on this expression, the electron density can be represented for a plane of the chemical space where each atomic nuclei can be related to basins. In this way, QTAIM divides this real space generated by the electron density topology into different regions (Ω). Each of these regions corresponding to an attraction basin and, as mentioned above, can be typically assigned to an atom of the system[3]. Under this approach, the global values of a property can be constructed as the sum of its local contributions, the latter being either atomic (belonging to a single atom: located on a specific basis) or interatomic (belonging to a pair of neighbouring atoms: distributed between two neighbouring basins).

In the topology of this formed space, different critical points can be found, among which the most interesting for the description of bonds with this theory are the bond critical points. To be considered a bond critical point, a critical point must fulfill the following requirements: it must have a minimum in one direction, called "bond path" (which usually is the line of union of two nuclei), and a maximum in the other two directions.

[3]Nevertheless, a common problem is the appearance of Non-Nuclear Attractors, basins of attraction not associated with an atomic nucleus.

In order to visualize this real space and this bond critical points, in Figure 3.1 the electron density topology and its bond critical points can be seen for the plane that contains the three atoms nuclei in the water molecule.

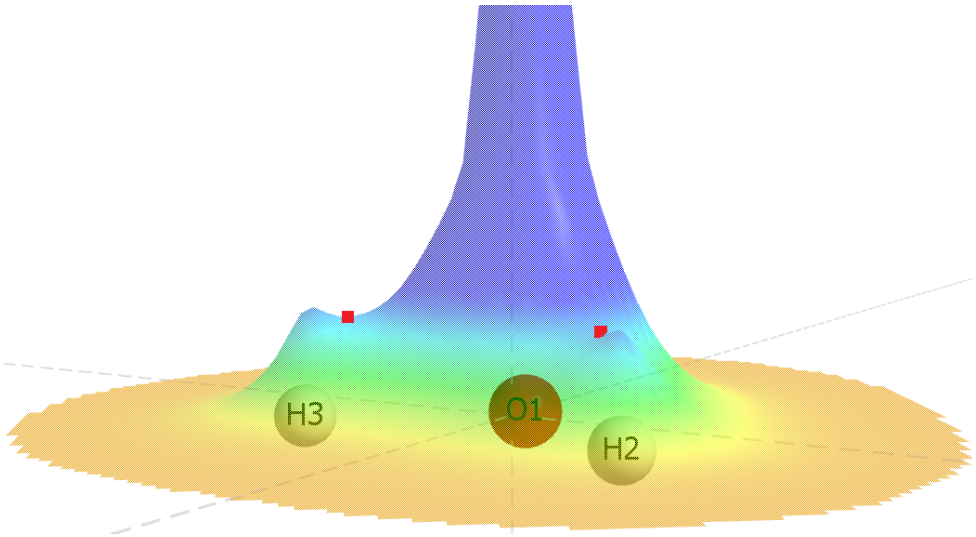


Figure 3.1: Water molecule atoms (balls) with the electron density (heat map surface) colored the colder the higher density for the molecular plane which contains the three atoms, and the two bond critical points in the system (red squares)

As a last concept of topology to highlight, an important theorem should be mentioned. The Poincaré-Hopf theorem[46], applied to electron density topology (i.e. a surface whose Euler characteristic, a famous topological invariant, is equal to 1) states that the sum of the indices of all the critical points of the surface must be equal to 1. The critical indices are related with the number of directions where electron density increases or decreases. It will not be discussed in detail, but it is important to keep in mind that this theorem must always be fulfilled for the electron density topology. Once these concepts have been introduced, it remains to see how they relate to chemical properties.

3.5.2. Properties of Atoms in Molecules

The QTAIM's ability to allow the splitting of a molecular property into its atomic contributors is very useful when explaining bonds or atomic interactions. For example, the number of electrons

in a molecule (N) can be split into its atomic contributors in the following way

$$N = \int_{R^3} \rho(r)dr = \sum_{A}^{M} \int_{\Omega_A} \rho(r)dr = \sum_{M}^{A} N_A \tag{3.26}$$

where N_A is the number of electrons of each atom. At this point it is important to know that QTAIM is a theory of open quantum sub-systems: since the operators $\hat{N}$ (for the number of electrons) and $\hat{H}$ (for the energy) do not commute, it is not possible to define an exact number of electrons for a region (just in the whole system), that is why it works with averages or expectation values, called atomic observables. [44]

Following with the properties, from the definition of the atomic number of electrons, the concept of partial charge can be deduced as

$$q_A = Z_A - N_A \tag{3.27}$$

where q_A is the atomic partial charge (also called Bader charge or QTAIM charge) and Z_A is the atomic number of the same atom which is associated to the basin Ω_A.

Furthermore, the concept of global charge (Q) can be defined from its atomic contributions in the following way

$$Q = \sum_{i=1}^{N} q_i \tag{3.28}$$

On the other hand, by applying a statistical analysis to the electron distribution between the basins, two components can be clearly separated. First, an electronic contribution inherent to each basin, formed by the so-called localized electrons, and second, an interatomic contribution corresponding to the group of electrons shared between two basins, called delocalized electrons. For both types of electrons, localization ($\lambda(A)$) and delocalization ($\delta(A, B)$) indices correspond respectively, which are in turn related to the number of electrons in each atom as follows

$$N_A = \lambda(A) + \frac{1}{2} \sum_{B \neq A} \delta(A, B) \tag{3.29}$$

where

$$\lambda(A) = N_A - \sigma_A^2 \tag{3.30}$$

$$\delta(A, B) = -2\sigma_{A,B} \tag{3.31}$$

where σ_A^2 is the variance of Ω_A and $\sigma_{A,B}$ is the covariance between Ω_A and Ω_B.

Through a chemical interpretation of these indices one can characterize and quantify the nature of chemical bonds [47], predict other useful properties (like dipole moments or dissociation energies)[48] or study itermolecular interactions (like hydrogen bonds [49], bond formation/breakage [50] or topology of periodic systems [51]), just to mention a few.

4 Deep Learning

This chapter presents concepts from the area of Artificial Intelligence, more specifically Deep Learning, which have been used in the project. In the first section, the place of Deep Learning within Artificial Intelligence is explained (Section 4.1). The following sections explain the dataset needed for the learning (Section 4.2), the artificial neuron as the basic unit of Neural Networks (Section 4.3) and the development of these networks (Section 4.4). Finally, the concept of Atom-Centered Symmetry Functions, an indispensable element for the training of neural networks using coordinate files, is introduced (Section 4.5).

4.1. Artificial Intelligence, Machine Learning and Deep Learning

In 1950, Alan Turing, considered one of the fathers of computing and precursor of modern computer science, started his article "*Computing machinery and intelligence*"[52] by throwing out a curious question to the reader: "Can machines think?" Throughout his life, with his reasoning, he laid the foundations of a proposal that, matured by a multitude of contributors, is now widespread worldwide in many different areas: artificial intelligence.

This field of computer science is based on the creation of systems capable of performing tasks that used to be only achievable through human intelligence. It is currently a very diverse area made up of huge fields such as, to mention a few, the following:

- Natural Language Processing: focused on human-computer interaction through human language.
- Robotics: based on the development of robots capable of performing tasks autonomously in physical environments.
- Computer Vision: responsible for enabling computers to interpret and understand the visual content of the world.
- Machine Learning: focused on developing algorithms and models that allow computers to automatically learn patterns from data.

Among them, the field of Machine Learning is defined as the "field of study that gives computers the ability to learn without being explicitly programmed" by Arthur Samuel (1959), a pioneer in this area.

Furthermore, Machine Learning algorithms are typically grouped into three categories, depending on the strategy followed for automatic learning:

- Supervised learning: models learn from labelled examples, i.e. they are provided with a dataset that includes inputs and desired outputs. Here, the algorithm learns to make predictions or classifications based on that information. Tasks here include:
  - Classification: the goal is to assign labels to observations based on their features.
  - Regression: the goal is to predict a continuous value based on the features of the observations. Such as Neuronal Networks models to predict properties, used in this project.
- Unsupervised learning: models are trained on unlabelled data. Here, the algorithm looks for patterns or intrinsic structures in the data without external guidance, allowing it to discover natural relationships or groupings within the data. This includes tasks as:
  - Clustering: the goal is to group observations into clusters based on their similarity (without labels of reference). Such as Gaussian Mixture Model [53], used in this work.
  - Dimensionality reduction: the goal is to reduce the number of features (dimensions) in a dataset while preserving as much relevant information as possible. Such as Principal Component Analysis [54], also used in this project.
- Reinforcement learning: here models interact with a dynamic environment by receiving feedback in the form of rewards or punishments based on their actions. The goal is to learn how to make sequential decisions to maximize the accumulated reward over time. Here, the task depend on the area to be addressed, ranging from play optimization to automatic driving.

There is a sub-discipline of Machine Learning called Deep Learning, widely used today. It uses artificial neural networks with multiple layers (an input layer, several hidden layers and

an output layer) to learn representations of data hierarchically, by imitating the structure of biological neurons.

In the following sections it will be shown where exactly the models learn from (Section 4.2), what an artificial neuron is (Section 4.3), how to develop a network of them (Section 4.4) and how to transform coordinate files into a usable functions for the Neuronal Network (Section 4.5).

4.2. Database for learning

In the field of Machine Learning, the data is the fundamental element that drives the training and the performance of the models. In a nutshell, the learning process consists of looking for complex patterns among the input data. These patterns allow the model to make decisions and predictions. As will be explained in the following, the quantity, quality and format of this data directly influence the ability of an algorithm to generate accurate predictions.

First, a big and diverse dataset provides a greater amount of information for the search of those patterns and relationships between variables. This is why it is so important to have a large and varied dataset that allows the model to capture the inherent complexity of the problem to be tackled.

Second, data quality is also essential for successful learning. The data must be clean, free of noise or outliers that obstruct the search for patterns. Furthermore, it must be representative of the problem domain: if, for example, the aim is to predict properties of particular systems, those systems (or similar) must be included in the dataset. In this way, the patterns found by the model will be useful for future predictions.

Finally, this data must be fed into the model in a correct format. Depending on the problem to be addressed and the shape of the data, a stage of transformation of the data into a format understandable by the neural networks may be necessary.

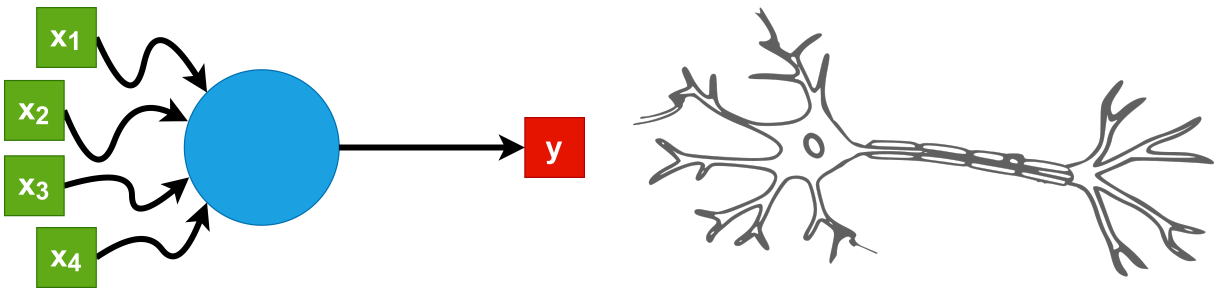
In this way, a database will be obtained to construct a function to associate these inputs, and even new ones, with their corresponding outputs. For example, as is the case in this project, the aim is to associate a molecular structure (input) with a QTAIM property for each atom (output).

All this because with a suitable database it is possible to define any function, however complex it may be, with a Feed-Forward Neural Network (FFNN), a network where the information only flows forward, without any loops (the simplest possible Neural Network). The latter statement is justified by the "Universal Approximation Theorem". Initially introduced by G. Cybenko (1989)[55] and finally extended by K. Hornik (1991)[56], this theorem states that: a single-layer hidden Feed-Forward Neural Network with a finite number of neurons can approximate any continuous function with arbitrary accuracy.

In this way, a neural network can use the collected information to form the input layer. Then, if these fulfill the characteristics described above, thanks to the Universal Approximation Theorem can be sure that the function which describes the given problem can be fitted. Once this is understood, the next question that arises is how exactly each of the neurons in the network manages to achieve this. For this reason, the following section will explain what an artificial neuron is and how it works.

4.3. Artificial neuron

The artificial neuron is the basic processing unit in Deep Learning and, like a biological neuron, it has input data that it processes to generate a single output. (See Figure 4.1).



(a) Artificial neuron

(b) Biological neuron

Figure 4.1: Schematic comparison of biological and artificial neurons where the information flows from left to right: from multiple inputs (in green) to one output (in red) in the artificial neuron and from multiple dendrites (on the left) to an axon (on the right) in the biological neuron.

This input processing in the case of the artificial neuron can be separated into two phases:

1. It starts with a weighted sum of elements where each input element has a specific weight assigned to it. Additionally, an adjustable parameter is added to this sum in order to control the activation of the neuron called "bias". Overall, this first stage consists of the application of the following equation:

$$z = \left(\sum_i w_i \cdot x_i \right) + b \tag{4.1}$$

   where z is the output element from the weight adjustment, b is the bias, x_i represents each input element and w_i its assigned weights.

   This equation represents a linear fit. Thus, this step can be understood as performing a weighted linear regression on the input data.

2. To understand the need for an additional step, let's imagine several neurons connected in sequence, which can be expressed as a sum of linear regressions. Knowing the equation that fits a line, similar to equation (4.1), it can be understood that the sum of several equations of the line generates a new equation of the line. And here is the problem, since this means that the work of multiple neurons in a network would be equivalent to that of a single neuron, preventing the prediction of more complex problems (shapes other than the straight line). For this reason, there is a second step in charge of including non-linear deformations to the output of the neuron. This phase applies a non-linear function, commonly called "activation function'', to the output of the weight adjustment, as follows

$$y = f(z) \tag{4.2}$$

   where y is the output value of the neuron and $f(z)$ is the activation function applied over the weight adjustement output (z).

In order to visualise it better, the final shape of an artificial neuron can be seen schematically in Figure 4.2.

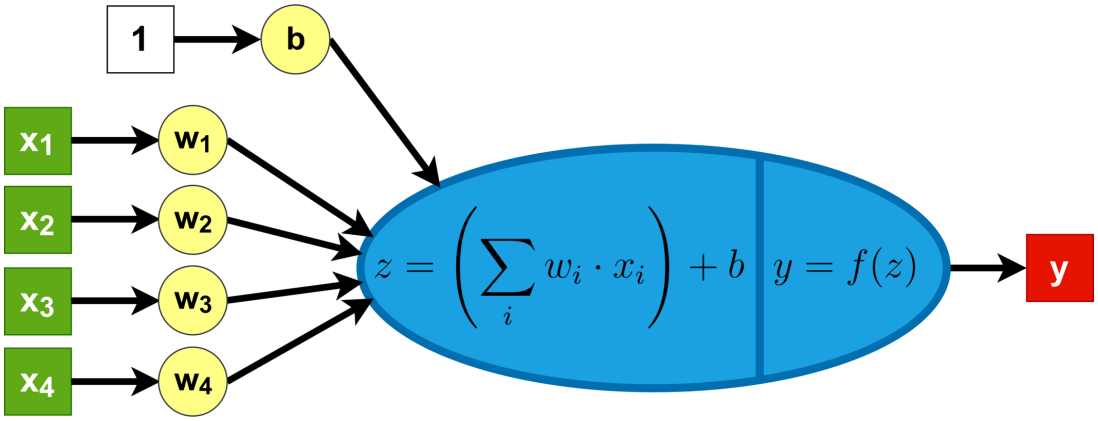


Figure 4.2: General structure of an artificial neuron

Once a neuron is constructed, it can be distributed in layers to form a Neural Network. These networks initially consist of an input layer, with the input data. This is followed by a succession of hidden layers, where pre-optimized parameters (w_i and b) transform the input data into output data by applying a succession of functions. And finally an output layer, with the desired result.

The optimization of the parameters that are included in the hidden layers, necessary for the correct fitting of the function that defines the problem to be addressed, will be specified in the next section.

4.4. Development of the Neural Network

It is important to remember that Deep Learning is a subset of Machine Learning, so the learning of the neural network must be done automatically. For this reason, the entire data set must be separated into subsets, as explained later in detail. Basically, the data is initially separated into a part for training and a part for further testing. Within the first part, in turn, another portion is reserved for the validation carried out during training.

First, it will be explained how the neural network uses the validation subset to "learn" at each epoch of the training and second, the usefulness of the test subset to analyze its final predictive performance will be detailed.

4.4.1. Training step

With the portion of the dataset that forms the validation subset, it is possible to obtain and minimize the error made at each epoch, improving the model during training. The most basic strategy executable by the model to carry out this optimization is called "Gradient Descent", which will be explained together with how exactly it is calculated below.

Gradient Descent

The neural network has some parameters (w_i and b) that must be constantly changing in order to minimize the error of the model. The way to express this error depends on the chosen loss metric, concept explained below (in Subsection 4.4.3, "loss metric" part), but in general it is a reflection of the discrepancy between the values predicted by the model and the expected ones. In this way, a loss function can be defined which, for given parameter values, assigns an error. If there is only one parameter to be optimized for the error minimization, the way to proceed with the Gradient Descent method is the following:

1. Assign a random value to the parameter, placing it at a location on the loss function.
2. Calculate the slope of the curve where the loss function is located.
3. Move a given distance in the direction opposite to the slope.
4. Repeat 2 and 3 until reaching convergence

By applying this methodology to all the parameters simultaneously, the values of the parameters that minimize the cost function can be obtained. It should be noted that if there are N parameters, the search for the minimum is being carried out in a space of N dimensions, which reflects that the greater the number of parameters, the more complex this optimization will be.

The gradient, centerpiece of this methodology, is a vector formed by the partial derivatives of each parameter of the network. Among the different ways of calculating it, one of them stands out for its efficiency and is currently the most widely used in Neural Network models. This algorithm is called "backpropagation" and will be briefly introduced below.

Backpropagation

Each vector of the gradient corresponds to a parameter and each group of parameters, in turn, corresponds to a neuron in the network. Taking this into account, the backpropagation algorithm obtains the partial derivatives very efficiently: starting from the final error obtained by the whole model, it starts stepping backwards splitting the error between the different neurons and thus between the different parameters. [For a detailed mathematical explanation of the algorithm, see chapter 6.5 of the Reference [57]]

Applying this methodology will result in an optimized model that minimizes the error in the train set prediction. But to study the final performance of the model, the test set must be used. Thus, this methodology will be introduced in the next section.

4.4.2. Test step

Minimizing training error excessively can lead to a problem called overfitting: the model basically memorizes the training data, predicting it with high accuracy but being inapplicable to new data, such as the training set. For this reason, and with the general objective of analyzing the performance of the model on an unseen dataset, a second step of testing is carried out.

There are multiple strategies to do this. The most basic option would be to separate the entire dataset into fixed train and test subsets. On the other hand, one of the best known techniques is the k-fold cross validation (not related to the validation set).

K-fold cross validation

In this technique, the entire dataset is initially divided into 'k' subsets. Then, in a first step, the first subset is taken for the test, forming a train set with the others. In a second step, the second subset is taken for the test and the others for the train, and so on. This will result in 'k' steps, each with a corresponding error. The average of these errors forms the total error of the model.

Finally, if the results obtained have acceptable errors, it can be considered that the model is valid for the chosen task. Nevertheless, it must be taken into account that the correct performance of the model is only ensured for the interpolation range. If it is requested, for example, to make

a prediction of a property in a range very different from the trained one, the neural network will be extrapolating the information it knows, and this, working on multidimensional surfaces (such as those used to minimize the loss function) is a very risky approach.

With this strategy, the internal parameters of the model will finally have been optimized. However, there are several fixed parameters that must be specified to the network before training, as they define its structure and global characteristics. In the next section, these elements will be introduced and explained one by one.

4.4.3. Model hyperparameters

In the area of Deep Learning, the concept of parameter refers to internal elements of the network that are automatically adjusted during training, while hyperparameter refers to an external element set by the neural network programmer. For example, the weights (w_i) or the bias (b) would be referred to as parameters, while the chosen loss metric or the percentages of data for testing or validation would count as hyperparameters. This section will explain the main hyperparameters, which are also the ones that must be specified to NNAIMGUI for training.

Fraction of data for training

From the whole database, a part of the data is separated for training and the remaining part for the subsequent test, where the ability of the model to act on a dataset other than the one used for training is studied.

Typical values for this separation are 80% of the data for the training and 20% of the data for the test.

Validation split factor

In addition to evaluating our model after with the test step, a validation set is extracted from the training data set in order to use it to evaluate the performance of the model at each epoch.

Typical values for this hyperparameter are between 0.1 and 0.2 (usually depending on the size of the whole database).

Maximum number of epochs

An epoch corresponds to each iteration where the whole training set (without the validation set) is passed through the neural network to update its parameters and to monitor them (with the validation set). To avoid that the training takes an excessive number of epochs, it is advisable to specify a maximum number of iterations in the training.

This number should be big enough to allow convergence without being excessively high (to limit the computational cost).

Patience

In spite of defining a maximum number of epochs, it is possible that the training reach its optimal peak earlier. If the number of iterations is further increased, a waste of resources or an overfitting of the model to the training set will be caused. To avoid this, the early stopping approach can be used. In this approach, patience is defined as the maximum number of consecutive iterations that the model is allowed to get worsen error metrics in the validation step. In this way, when the model starts to learn by focusing excessively on the training set, the early-stopping approach should be able to stop it before it starts to overfit.

This patience should be large enough to permit the natural fluctuations of the model but small enough to stop the training when a bad trend is detected.

Learning rate

Understanding the training process as a optimization looking for the minimum error, the learning rate can be interpreted as the step size of each iteration (or epoch). In this way, with a high learning rate we will take larger steps towards the minimum (faster optimization) but with the danger that if it is not low enough, it will not be able to easily enter into the hole of the minimum, leading for divergence behaviors. On the other hand, a very low learning rate from the beginning will make the optimization unnecessarily slow even getting stuck in areas away from a low enough minima. Both of these problems can be seen schematically in Figure 4.3.

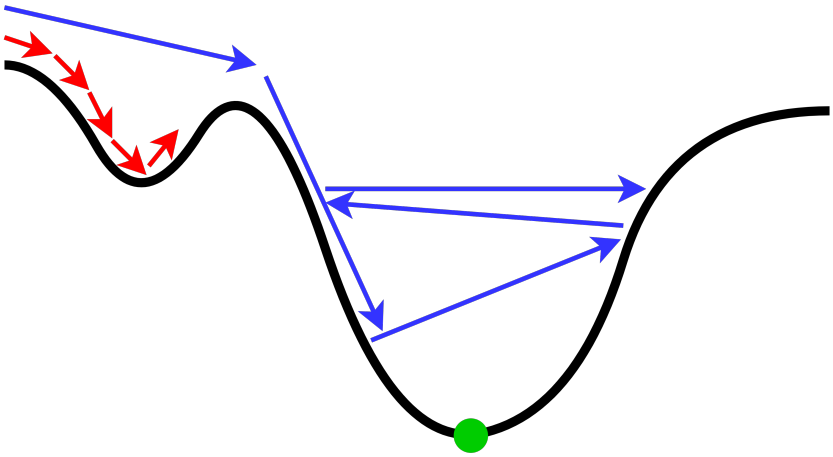


Figure 4.3: Diagram of the optimization process with too small steps (low learning rates), red arrows, and with too high step sizes (high learning rates), blue arrows.

For this reason, given that at the beginning of the optimization it is convenient to take larger steps to locate the minimum region but once it is reached it would be convenient to take these smaller steps to locate exactly the location of the lowest point, a very common strategy is to reduce the learning rate throughout the training. The responsible of carrying out this methodology, and in general for helping to find the minimum, is another hyperparameter that will be discussed later, called optimizer.

Typical values for a fix learning rate will be around 10^{-5} meanwhile for an initial value, if it will decrease along the training, it tend to be a number between 0.1-0.0001.

Loss metric

Previously it has been mentioned concepts such as error, in the validation step, or worse results, in the early stopping approach, but this raises a question: what criteria are being used to determine these errors? The answer lies in the loss metric chosen, as it determines the way to represent the discrepancy between the model predictions and the real values of the training data. Among the most commonly used the following can be distinguished:

- Mean Absolute Error (MAE): easy to understand and perform, this metric calculates the absolute difference between each model prediction and the corresponding actual value for each sample in the validation set and taking the average of all these differences the MAE

is obtained. The formula for its calculation is

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.3}$$

where n is the number of samples in the data set, $\hat{y}_i$ is the model prediction value and y_i corresponds to the actual value.

- Mean Squared Error (MSE): is another commonly used metric to evaluate the accuracy of a model. This metric calculates the squared difference between each model prediction and the corresponding actual value for each sample in the validation dataset, and then averages all these squared differences. This provides insights into the extent of the errors, giving more weight to larger errors. The formula for its calculation is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.4}$$

Another interesting option is to design an own based loss metric on the data set. For example, if the relationships/rules that the output data must comply with are known, it is possible to impose as a metric how close the predictions are to satisfying them.

Optimizer

There are algorithms to assist the model in the optimization task. Specifically, these algorithms are used to adjust the model weights during training, in other words, these algorithms seek to find the optimal values for the model weights that minimize the loss function, which in turn improves the performance of the model on training data and hopefully on unseen data as well. Among the more common optimization algorithms are the following:

- Gradient Descent (GD): As was explained in the Subsection 4.4.1, this method reduces the loss function by moving it in the opposite direction of the steepest ascent (i.e. the gradient of the loss function). The formula that follows is

$$w_t = w_{t-1} - \eta \cdot \nabla Q_t(w) \tag{4.5}$$

where w_t is the weight for this epoch, w_{t-1} is the weight from the previous epoch, η is the learning rate (step size) and $\nabla Q_t(w)$ is the gradient of the loss function (in this epoch).

To apply this formula, all the data from the training set is necessary to calculate the loss function. This procedure have to be done at each epoch, which can be computationally expensive but it is an effortless method to understand and implement.

- Stochastic Gradient Descent (SGD): In this variant of Gradient Descent, the model updates the parameters one by one: once for each dataset element. In this way, at each epoch the optimizer will randomly take only one element from the training set, which drastically reduces the computational cost. A midpoint between GD and SGD is the so-called Mini-Batch Gradient Descent: instead of a single element per epoch, this algorithm uses a small portion of the training set.

  Nevertheless, the methods explained up to now have a drawback, they are unable to cross positive gradients (since they only move towards strictly the minimum) which could be a problem to overcome small bumps or roughness of the surface. In order to fix this, an element that simulates the inertia of a moving object can be introduced. This additional method is called Momentum and basically, remembering the direction of previous steps the inertia can be induced in the optimization by modifying the equation (4.5) to the following

$$w_t = w_{t-1} - \eta \cdot \nu_t \tag{4.6}$$

  where

$$\nu_t = \gamma \cdot \nu_{t-1} + (1 - \gamma) \cdot \nabla Q_t(w) \tag{4.7}$$

  where ν is the accumulated gradient and γ is the coefficient of momentum (or momentum hyperparameter).

- Adaptative Gradient Descent (AdaGrad): Another strategy in the optimization endeavour is to consider a varying learning rate throughout the training. As mentioned above, this is a very useful strategy for finding an optimization minimum. The AdaGrad algorithm includes a modification of the learning rate in each iteration, so it is no longer a fixed value. With this methodology the resulting equation becomes

$$w_t = w_{t-1} - \eta_t \cdot \nabla Q_t(w) \tag{4.8}$$

  where

$$\eta_t = \frac{\eta_0}{\alpha_t + \varepsilon} \tag{4.9}$$

where ε is a small number, usually 10^{-5}, different to zero (to avoid dividing by 0) and the α_t term follows the equation

$$\alpha_t = \alpha_{t-1} + (\nabla Q_t(w))^2 \tag{4.10}$$

Basically, at each epoch, the learning rate is being divided by the cumulative sum of the squares of the gradients obtained up to now. However, the only risk with this algorithm is if the learning rate decays to values excessively close to zero, which could cause a problem called gradient vanishing: if the points ends up close to an area that has a derivative close to 0 the gradient disappears in this neuron, becoming a 'dead neuron'.

- Root Mean Square Propagation (RMSProp): This optimizer is a special and more widely used version of AdaGrad. In this case the learning rate is updated based on the root mean square of the previous gradients but with a moving average this time, meaning a redefinition of α_t to

$$\alpha_t = \rho \cdot \alpha_{t-1} + (1 - \rho) \cdot (\nabla Q_t(w))^2 \tag{4.11}$$

where ρ is the decay rate parameter (hyperparameter of RMSProp).

- Adaptative Moment Estimation (Adam): Probably the optimizer by excellence. This is because it is able to combine the best of both worlds: it uses the concept of Momentum (with its hyperparameter) together with the RMSProp methodology to reduce the learning rate (with its decay rate parameter). Under this algorithm, the final expression of the weights update at each epoch becomes

$$w_t = w_{t-1} - \eta_t \cdot \nu_t \tag{4.12}$$

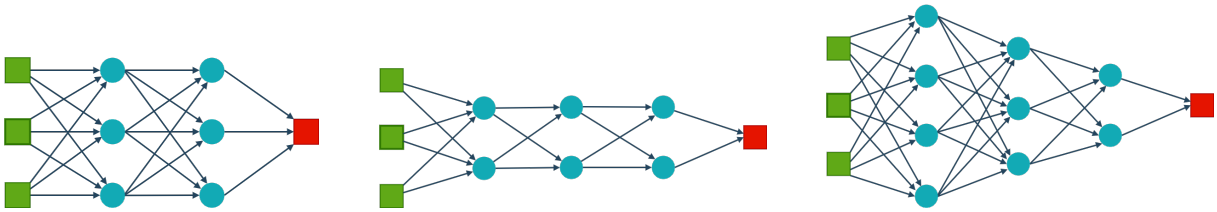
where η_t is the learning rate obtained with the algorithm RMSProp and ν_t the accumulated gradient obtained with the Momentum algorithm (both at each epoch t).

Among the algorithms presented, the most typically used are RMSProp and Adam, probably because of their functionality to reduce the learning rate.

Number of neurons

An important concept in deep learning is the architecture of the model. With hyperparameters intended to define the structure of the neural network, the number of intermediate layers between the input layer and the output layer, called hidden layers, is first defined, and then the

number of neurons in each of these layers is selected. Some examples of model architectures can be seen in Figure 4.4 to visualise these concepts better. The optimal architecture depends



(a) Hidden layer (3,3) (b) Hidden layer (2,2,2) (c) Hidden layer (4,3,2)

Figure 4.4: Different hidden layer architectures (blue dots) for an input layer of three elements (green squares) and an output layer of one element (red square).

essentially on the problem to be tackled, so there are apparently not many guidelines. But in a nutshell, increasing the complexity of the model (large number of hidden layers and/or neurons) the number of training data required will be much higher than for simpler architectures otherwise the risk of overfitting will be considerable.

Activation functions

For the before mentioned activation functions in the Section 4.3, depending on the problem to be tackled, multiple options can be found. Among the main options are the following:

- Linear: Although it has been mentioned that one of the objectives of the activation function is to produce non-linear results, there are situations in which this linearity needs to be preserved. For example, if the output of the last neuron is wanted to be a number associated with the prediction of a parameter, that can take any real value, the activation function of this layer must let the information pass without modifying it, i.e. applying a linear function. Its representation, in Figure 4.5, is very simple, but it can be helpful to understand what activation functions do in a visual way.

  In this case it is simple to see that whatever value is entered, the linear function will return that same value. This could be seen as not applying any activation function, but it should be indicated in the model architecture.

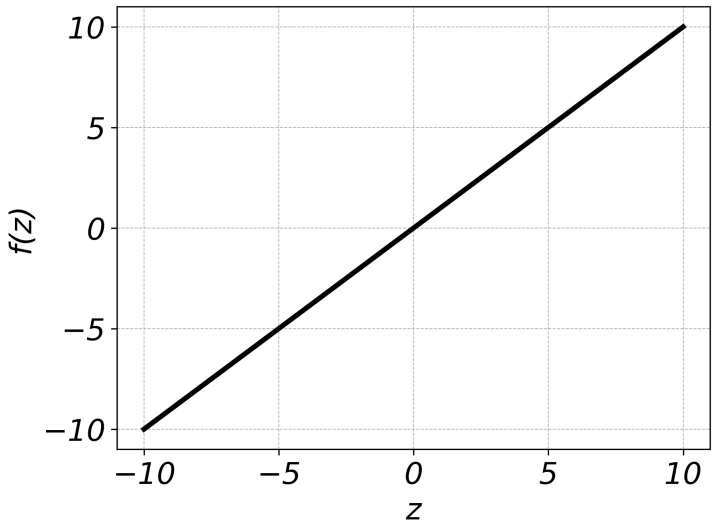


Figure 4.5: Linear activation function

- Sigmoid: One of the first used activation function, which follows the equation

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{4.13}$$

where z is the input value and whose shape can be seen in Figure 4.6.

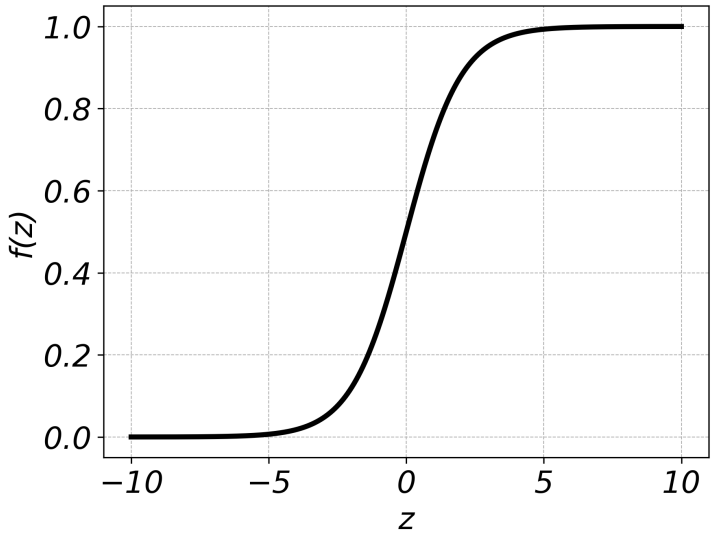


Figure 4.6: Sigmoid activation function

This function generates values between 0 and 1, assigning practically 0 to large negative numbers and practically 1 to large positive numbers and it is commonly used in binary classification problems.

Sigmoid function can present problems of gradient vanishing, as with the AdaGrad optimizer (centered in x=0 in this case).

- Softmax: This function can be understood as a generalisation of the sigmoid function since it is also used for classification tasks, but this time multi-class. To understand how it does this, let's start by looking at the equation which follows:

$$softmax(z_i) = \frac{z_i}{\sum_{j=1}^{n} e^{z_j}} \tag{4.14}$$

where n is the total number of elements.

This equation generates a useful outputs because all of them are normalized (between 0 and 1 as with sigmoid) but also the sum of these elements is always 1. In Figure 4.7 the softmax function for eight equispaced dots can be seen.

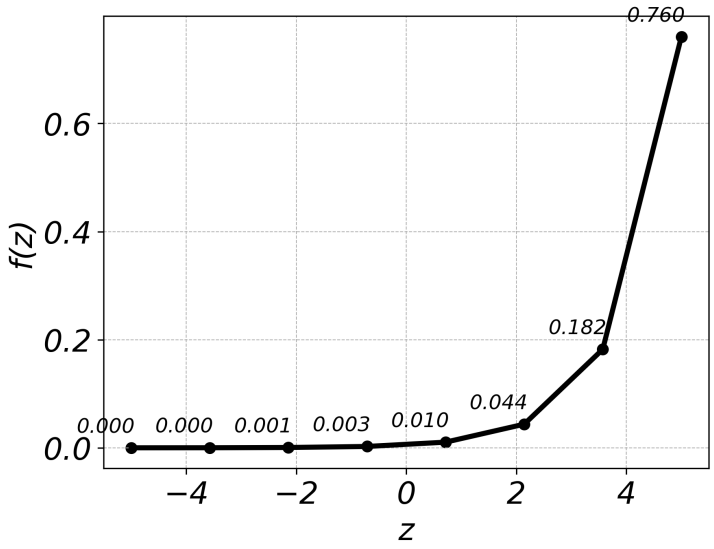


Figure 4.7: Softmax activation function

As can be seen, the fact of using exponential expressions gives it a more restrictive character when classifying (small differences count a lot). To understand how it works, let's imagine that just if before sigmoid generates a single value between 0 and 1 to indicate whether it belonged to one class or another, now with softmax an output vector with values between 0 and 1, with one position for each class, where the higher the value of each

element in the vector the more likely it is that it belongs to that class. Moreover, given that all the positions of the vector add up to 1, they are directly giving probabilities of belonging to each class.

- Hyperbolic tangent (tanh): This is a popular activation function which follows the equation

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4.15}$$

generating the curve in Figure 4.8.

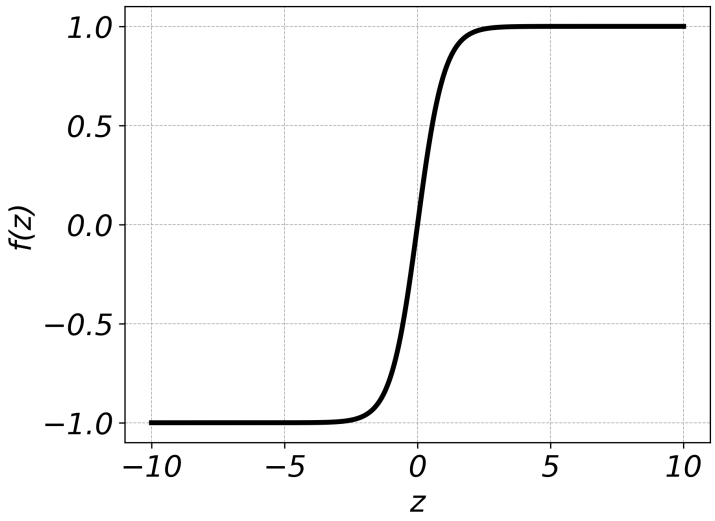


Figure 4.8: Hyperbolic tangent activation function

This function is symmetric because it centers its output values on 0, working in the interval from -1 to 1. This function can present the same problem of gradient vanishing as the sigmoid function for the same reason as the latter.

- Rectified Linear Unit (ReLU): A different approach is to focus only on the positive region to force positive values and discard negative ones. The ReLU function with the equation

$$ReLU(z) = max(0, z) \tag{4.16}$$

where "max" operator choose the higher value among the givens, generates the curve in Figure 4.9.

The greatest advantage of this method is its low computational cost which results in relatively small training times. However, this function can present two problems. The first

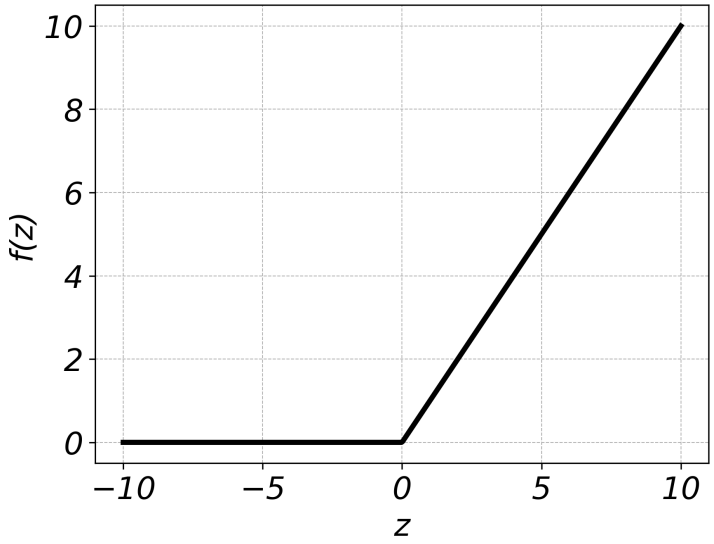


Figure 4.9: Rectified Linear Unit activation function

is, as in the sigmoid function, the vanishing gradient. The second possible problem, less common, is exploding gradients: since the ReLU function has no upper limit, the values passing through it can scale to really large numbers, which generates huge diverging gradients.

- Scaled Exponential Linear Unit (SELU): Since the greatest risks with ReLU come from the drastic elimination of negative values, multiple alternatives appeared to deal less heavily with this region. Leacky ReLU, for example, instead of a horizontal for the negative region opts for a straight line with some slope. The Exponential Linear Unit (ELU) function instead of a straight line for the negative region makes use of an exponential, following the equation

$$\text{selu}(x) = \lambda \times \begin{cases} x & \text{if } x > 0 \\ \alpha \times (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \tag{4.17}$$

where α and λ are parameter to adjust the shape of the exponential. If these parameters are set to concrete values ($\alpha \approx 1.67$ and $\lambda \approx 1.05$), in theory, the network will self-normalize (outputs centred at 0 with standard deviation 1), solving not only the gradient vanishing, but also the explosive gradient problem. This function is known as Scaled Exponential Linear Unit (SELU) and its specific shape can be seen in Figure 4.10.

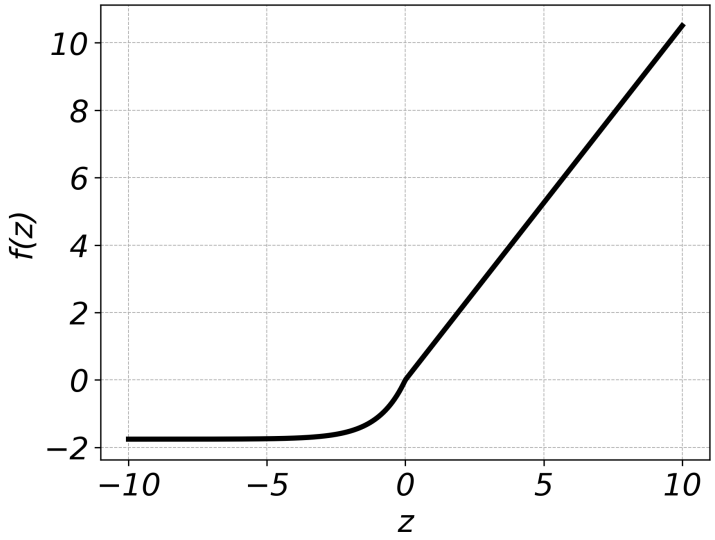


Figure 4.10: Scaled Exponential Linear Unit activation function

The optimal activation function depends a lot on the problem to be tackled and the architecture of the model, so it may be interesting to consider several of them and test which one works best.

Having understood the hyperparameters used to define a model, from the Deep Learning area only one key element remains to be introduced in order to be able to work with coordinate files as input data. For this reason, the following section will explain how to go from a set of geometries to the input layer in the Neural Network.

4.5. Atom-Centered Symmetry Functions

In Section 4.2 it is mentioned that the input data must have a correct format, interpretable by the Neural Network. In the specific case of a database of molecular geometries (as is the case in this work), in order to serve as an input layer, it should satisfy the following properties:

- Invariance to geometrical transformations: the properties should not change if the molecule is shifted in space, rotated or permuted a couple of identical atoms.
- Intensive with the size: the input layer is recommended to have a fixed-size, being consistent for any number of atoms.
- Bijectivity: the input should ensure a unique and precise correspondence between the mathematical representation and the molecular structure

Among the mentioned properties, the geometrical coordinates do not fulfill any of them. The same molecule moved in space changes its coordinates completely (it is not invariant to geometrical transformations). Moreover, the number of atoms directly influences the number of coordinates (it is not size intensive). Finally, the same structure can be reached from multiple different coordinates by geometric transformations (it is not bijective). For this reason, an alternative to conventional coordinates is necessary.

Therefore, within a direct application in Deep Learning but conceived in the area of theoretical chemistry, J. Behler in 2011[58] presented a way to convert a set of coordinate files into usable functions for the training of a Neural Network.[58]

The so-called Atom-Centered Symmetry Functions (ACSFs) capture the local environments of each atom contained in the molecule by means of functions that are invariant to geometrical transformation, intensive with the size and (with some exceptions) bijectives. This makes these functions consistent and generalizable representations of the molecular geometries.

In addition, a great advantage is its efficiency because it works with local environments. Instead of molecular representations, it focuses on atom-centered functions that offer lower computational costs without losing information. To do this, it makes use of cutoff functions that limit what is considered to be the local surroundings of each atom.

There are different types of functions to build these ACSFs but they can mainly be grouped into two groups:

- Radial functions: capture the distribution of atoms around a central atom based on the distances between them (“two-body-dependent” functions).
- Angular functions: capture the angular relationships between a central atom and two neighboring atoms (“three-body-dependent” functions).

Additionally, interactions between larger numbers of atoms can be included, such as dihedrals (“four-body-dependent” functions).But for most situations, the two types mentioned above are sufficient. The specific functions selected for the present work will be developed in more detail later on (in Section 5.6).

Altogether, these functions provide a complete description of the local environment of each atom and can be used to feed a neural network. This allows, for example, the creation of Deep-Learning models capable of learning and predicting physico-chemical properties effectively. In this project they will be used in this way, both with the partial charge for the re-validation of NNAIMQ as well as the localization index to develop the new model.

5 Computational methods

This chapter describes the methodologies proposed to address the objectives of this Master's thesis. It starts with the two ways to generate geometries from an initial structure for the database creation: with Normal Mode Sampling (Section 5.1) and with Molecular Dynamics (Section 5.2). Afterwards, the procedure followed in this project for the calculation of QTAIM properties is detailed (Section 5.3), as well as the procedure used for their prediction with NNAIMGUI (Section 5.4). Furthermore, the keywords used by NNAIMGUI for the different hyperparameters of the training are indicated (Section 5.5). Finally, the procedure used with the MM2SF code for the generation of ACSFs is detailed (Section 5.6).

5.1. Geometry generation with Normal Mode Sampling (NMS)

The first option considered to generate from a few molecules (in 'xyz' file format) a multitude of different geometries with which to generate the database was to use the Normal Mode Sampling methodology.[59]

The NMS technique takes an initial geometry and creates distortions of it using its vibrational levels. These distortions can be seen as random linear combinations of the normal modes of vibration, which will generate different geometries that are feasible at the quantum-mechanical point of view. With this procedure, no matter how many geometries are generated, they will all belong to the same conformer, since in order to pass from one conformer to another, energies higher than the vibrational ones need to be considered. For this reason, a conformer generation technique must be implemented before the NMS. For the conformer generation, the chemical toolbox Open Babel [60] was used, in particular its "confab" function.

This option generates all the permutations of bonds in the molecule that are chemically possible, i.e. it obtains all the possible geometries resulting from bond rotations. For this reason, the distortions generated correspond to major energy changes and are of a mechano-classical nature. Subsequently, an optimisation of the geometry of each possible conformer generated was carried out with Gaussian [61] since it may be that different rotations in the molecule lead to similar conformers. Once filtered, NMS is performed on each of the resulting conformers by

setting the temperature to 900K with a python code that takes the normal modes of vibration and applies them in random proportions to the initial geometry. In this way different geometries will be generated for each molecule.

Figure 5.1 summarizes the methodology used to obtain geometries with NMS.

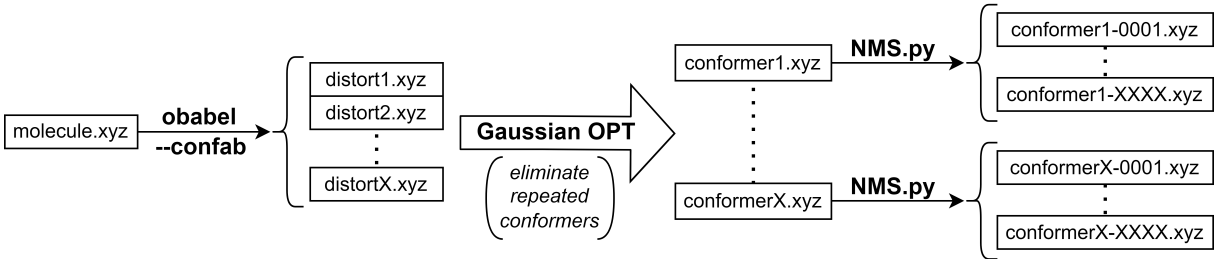


Figure 5.1: Flowchart of the NMS methodology to generate different geometries from a molecule

5.2. Geometry generation with Molecular Dynamics (MD)

As a second option for the generation of geometries from a few molecules, the use of Molecular Dynamics[62] was considered for this project.

The aim of applying MD is to employ a more robust sampling technique which should be able to escape the equilibrium potential associated with the initial geometry, allowing a wide variety of geometries to be visited. In this way, by regulating the temperature conditions and time intervals, the equations of motion of the system are propagated following the energies and forces acting on each nucleus. To run the molecular dynamics the Python library *ASE*[63] was used at a *GFN-xTB*[64] level of theory with the electronic temperature at 900K and with the Langevin thermostat with a friction coefficient of 0.1 fs^{-1} and a temperature set at 900K as well. Each step of the dynamics corresponds to one fs and from those finally performed, 1 out of 5 is selected to ensure differences among the structures. This results in geometries of the molecule where, if the dynamics parameters have been properly chosen and the simulation time was large enough, all the different conformers will be included.

Figure 5.2 summarizes the methodology used to obtain geometries with MD.

Figure 5.2: Flowchart of the MD methodology to generate different geometries from a molecule. In this project an $n = 5$ was always used.

5.3. Atom-in-Molecule property calculation

This section details the methodology used to obtain from an xyz file, containing the geometry of a molecule, a desired QTAIM property for each of its atoms. As will be shown below, the *Gaussian*[61] and *AIMALL*[65] packages are combined with the programming languages *Bash*[66] and *Python3*[67].

5.3.1. (Optional) Geometry optimization

This step is recommended for initial geometries obtained experimentally or of imprecise origin. Basically, it consists of a geometry optimisation with its frequency calculation with Gaussian (*"OPT FREQ"* keywords).

In the output generated, it is checked that all the values of the frequencies are positive, which indicates that there are no imaginary frequencies (since the coefficients are squared), thus confirming that what is obtained is a "real minimum" and not, for example, a transition state.

The level of theory set for this optional step is M06-2X/def2-TZVP (for coherence with the next step) and the final geometry obtained will be the one used as input for the next step.

5.3.2. Obtaining the wave function data file

From the starting geometry, a single-point calculation is performed with Gaussian redirecting the checkpoint file to a "wfn file" where the wave function information is collected. The level of

theory chosen was M06-2X/def2-TZVP because of the proven reliability of M06-2X, the sufficient accuracy provided by a "triple-Z" basis and the fact that this is the level of theory applied in the training of NNAIMQ, the pre-built NNAIMGUI model for partial charge prediction.

If the calculation ends normally, the generated file with 'wfn' extension will be the input for the AIMALL package, responsible for applying QTAIM into the system in the next step.

5.3.3. Performing QTAIM calculations

From the wavefunction data file, the properties of Atoms in Molecules can be calculated. To do this, it was decided to use the AIMALL application for setting up and running most QTAIM calculations called AIMQB. Additionally, in order to minimize the computation time, different bash command-line options from AIMQB were activated. Once the process has been successfully completed, three additional things must be checked in the output in order to ensure a quality database:

- Proof of the Poicaré-Hopf theorem: above-mentioned in Section 3.5 theorem in topology related to the number of different critical points present in the system.
- Absence of non-nuclear attractors: they can be a problem, since they are maxima in the electronic density that cannot be associated with an atomic nucleus.
- Total charge close to zero: since this project only works with neutral molecules, the sum of all partial charges should be zero, or close to it.

Finally, if the three previous conditions are fulfilled, it means that the QTAIM calculations have been carried out correctly on the system and the last step to obtain the desired properties can proceed.

5.3.4. Extracting the desired properties

Within the outputs generated by AIMQB, the file with 'sum' extension contains, among other things, several QTAIM properties for each atom. Since the ones to be extracted in the present work are both the partial charges and the localization indices, a Python script was created to extract them in separate files and in a suitable format for their later interpretation and comparison

with the results predicted with NNAIMGUI. In addition, the script also extracts the delocalization index, although it is not explicitly treated in this work, as it is a complementary variable to the localization index.

5.4. Properties prediction with NNAIMGUI

The methodology followed for the prediction of properties with the NNAIMGUI code is outlined below. For a detailed explanation of how to use this code and all its functions, it is recommended to consult its GitHub repository: `https://github.com/m-gallegos/NNAIMGUI/`.

In essence, either on the command line or via the graphical interface, the following information must be specified:

- The geometry file (in xyz format)
- The model to be used: NNAIMQ (partial charge prediction) or a custom one. If the later is used, then the folder model, the name of the target property and its units have to be indicated.
- Optional settings: to modify the default charge equilibration scheme, used to remain the sum of the partial charges equals to the total charge.

Once NNAIMGUI has been executed, if everything finished normally, a file with extension 'nnaim' will be obtained, containing the prediction of the desired QTAIM properties for each atom in the system.

Additionally, given that the aim is to compare these results with those obtained with AIMQB, a Python script was used to rewrite the results in a new file with the correct format.

5.5. NNAIMGUI hyperparameters

The main step to perform with the NNAIMGUI trainer module is the training of the model itself. To configure the training, the hyperparameters shown in the Section 4.4.3 must be specified, choosing for each one the option that generates the best results. The keywords in NNAIMGUI

for each hyperparameter are grouped in the Table 5.1.

NNAIMGUI keyword	Hyperparameter
ftra	Fraction of data used for training
vsplit	Validation split factor
nepochs	Maximum number of epochs
patnc	Patience (for early-stopping approach)
lr	Learning rate
loss	Loss metric
optimizer	Optimizer algorithm
neurons	Model architecture
activations	Activation functions of each hidden layer

Table 5.1: NNAIMGUI keywords to include the different hyperparameters of the model

Among the options to be taken into account, the best combination of hyperparameter values, in this project, will be decided through a benchmarking.

5.6. Creating ACSFs with MM2SF

In this section the procedure followed in this work to obtain the ACSFs from a set of geometries in xyz format by using the Python code MM2SF will be introduced. For a detailed description of the use and specific functions of this code it is recommended to consult its GitHub repository: `https://github.com/m-gallegos/MM2SF`.

In short, MM2SF creates an optimal collection of ACSFs in an unsupervised way, since it makes use of a Gaussian Mixture Model (mentioned in Section 4.1) which decomposes the charasteristic chemical space of a molecule into delimited clusters. This code works on the radial (two-body) and angular (three-body) functions.

On the one hand, it uses a "standard radial symmetry function" denoted as

$$G_i^{rad} = \sum_{j \neq i}^{N} e^{-\eta(r_{ij}-r_s)^2} \cdot f_c(r_{ij}) \tag{5.1}$$

where r_s and η correspond to the center and width of the exponential curve in the spatial distribution respectively and f_c is the cutoff function used to delimit the local chemical environment. In the Figure 5.3, **taken from the Reference** [68], the effect of this parameters in the spatial distribution of this standard radial symmetry function can be seen.

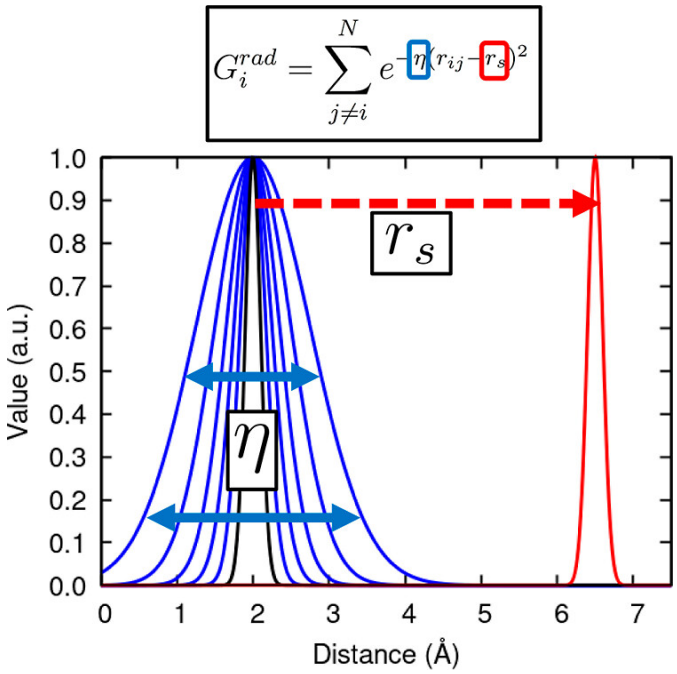


Figure 5.3: Effect of the r_s (red) and η (blue) parameters in the spatial distribution of a standard radial symmetry function. The following values were employed, r_s = 2.0, 6.5 Å and η = 0.90, 2.0, 4.0, 8.0, 17.3, 50.0 Å^{-2}.

On the other hand, it uses a "heavily modified angular symmetry function" which can be expressed as

$$G_i^{ang} = 2^{1-\xi} \sum_{j,k \neq i}^{N} (1 + \cos(\theta_{ijk} - \theta_s))^{\xi} \cdot \exp\left[-\eta \left(\frac{r_{ij} + r_{ik}}{2} - r_s\right)^2\right] \cdot f_c(r_{ij}) \cdot f_c(r_{ik}) \quad (5.2)$$

where θ_{ijk} is the angle formed by the reference atom, i, and the two neighboring atoms, j and k; ξ correspond to the width of the function in angular dimensions.

In order to visualize it, in the Figure 5.4, **also taken from the Reference** [68], the effect of each parameter of the equation on the spatial representations of the heavily modified angular symmetry function (also called "polar plot") can be seen.

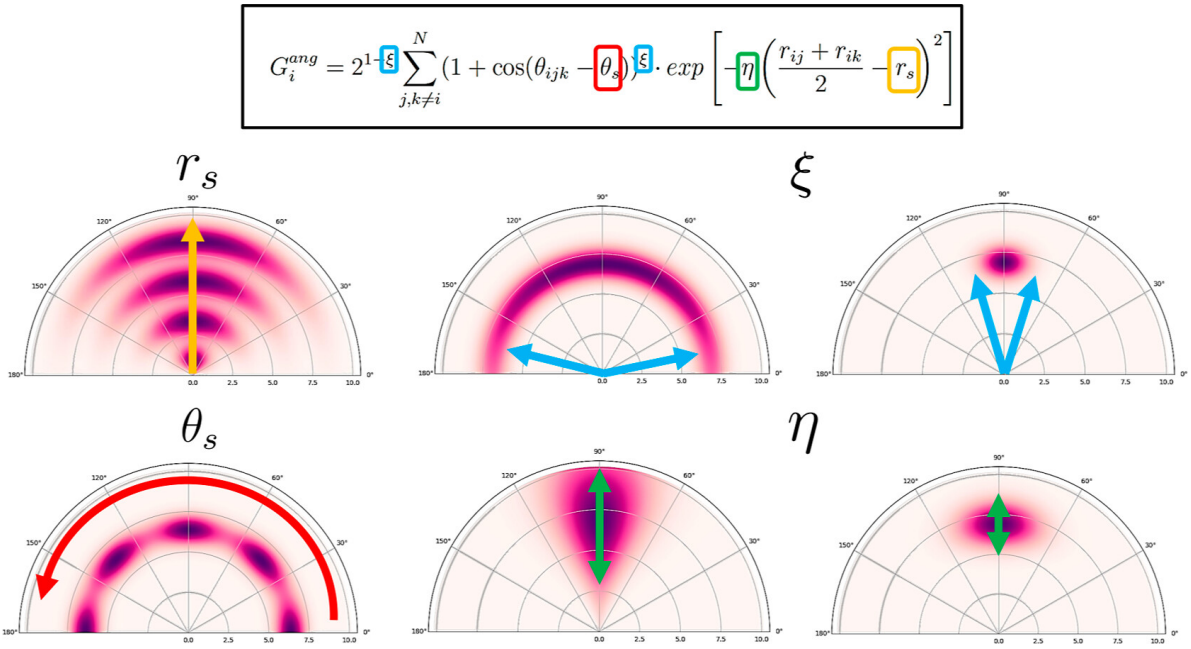


Figure 5.4: Effect of the r_s (orange), θ_s (red), ξ (blue), and η (green) parameters in the spatial distribution of a heavily modified angular symmetry function. The following values were employed r_s = 1.0, 3.5, 6.0, 8.5 Å, ξ = 1, 90, θ_{ijk} = 0, 45, 90, 135, 180°, and η = 0.05, 0.50 Å^{-2}.

In the results chapter are shown figures obtained with this code for several trios of atoms (ijk), which is composed of four sub-figures distributed as follows: top right and bottom left the spacial distributions of the radial functions, shown in Figure 5.3, for the pairs of atoms $i - j$ and $i - k$ respectively, bottom right the "radial dispersion plot" generated by projecting the two graphs mentioned above and top left the polar plot, shown in Figure 5.4, for the trio of atoms $i - jk$ (where the angle is formed between the $i - j$ and $i - k$ unions).

Finally, if the code is correctly executed, this methodology will be applied to each pair and trio of atoms in our set of geometries, generating the ACSFs wanted. Concretely, the code will generate three outputs with the following information:

- *input.type*: this file contains a list with the number of atomic species and its labels.
- *input.rad*: this file contains the resulting parameters for constructing the radial symmetry functions.
- *input.ang*: this file contains the resulting parameters for constructing the angular symmetry functions.

In these files all the information from the chemical environment mapped with the input geometries is condensed into a format suitable for neural network training. And for this reason, these files will form the input layer of the developed Neural Network model.

6 Computational details

The computational calculations carried out in the present project were performed in different clusters of the University of Oviedo. Each specific task, depending on the computational resources or packages required, was carried out with a particular cluster, using a maximum of 16 processors simultaneously (if available). In this way, the following processor models were used:

- ***Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz***: up to 4 of this processors may be used simultaneously for low-demanding tasks as running or training a FFNN models with *NNAIMGUI*.
- ***Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz***: up to 16 of this processors may be used simultaneously for tasks with *Gaussian* or *AIMALL* .
- ***Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz***: up to 16 of this processors may be used simultaneously for high-demanding tasks as running the MD with *ASE* or obtaining the ACSFs with MM2SF.

7 Results and discussion

This chapter presents and discuss the results obtained from the computational methods applied. First, the creation of the database, a key element when working with Neural Networks, is presented (Section 7.1). Then, the re-validation of the NNAIMQ model for partial charge prediction is discussed (Section 7.2). Afterwards, the results of the development process of the FFNN model for the prediction of the Localization Index will be detailed (Section 7.3). Finally, the application of the latter model to 4 systems of biochemical interest are presented and analyzed (Section 7.4).

7.1. Database creation

In order to re-validate the NNAIMQ model and to develop the new model to predict the localization index, a set of geometries whose QTAIM properties are known is needed. It was chosen to use for both cases the same dataset of geometries, using their partial charges for the re-validation and their localization indices for the development of the new neural network.

With the aim of having systems with a biochemical nature, but small enough to be able to build a sufficiently large dataset, it was decided to take as a starting point from the well-known "20 essential amino acids"[69] the 18 that have only C, H, O or N atoms (without S)[1]: Alanine (A), Arginine (R), Aspargine (N), Aspartic acid (D), Glutamic acid (E), Glutamine (Q), Glycine (G), Histidine (H), Isoleucine (I), Leucine (L), Lysine (K), Phenylalanine (F), Proline (P), Serine (S), Threonine (T), Tryptophan (W), Tyrosine (Y) and Valine (V).In the auxiliary Figure 9.1 the general structure of each amino acid is shown.

As it was explained in the Section 4.2, the larger the size of our dataset, in theory, the better results should be obtained. For this reason, as will be discussed later, the number of geometries finally generated was adjusted to be the maximum possible in the time available.

From these 18 initial molecules, multiple geometries must be generated to sample different

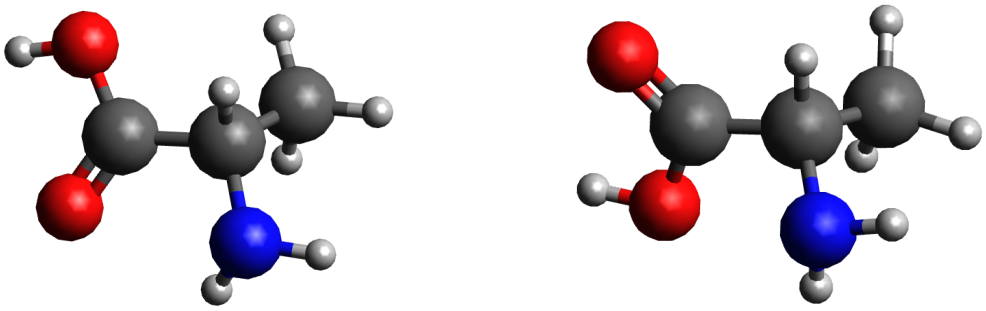
[1] It was decided not to include sulfur because the number of atom types increases the complexity of the chemical space exponentially and, given that in the time available the dataset to be built could be large but not enormous, it was decided to simplify the problem as much as possible.

chemical environments. For this purpose, two strategies were considered: Normal Mode Sampling and Molecular Dynamics.

To see the viability of each, both procedures (explained in Sections 5.1 and 5.2 respectively) were carried out for the alanine molecule. In the next section, the results obtained will be shown and discussed.

7.1.1. Generation of geometries for the alanine molecule

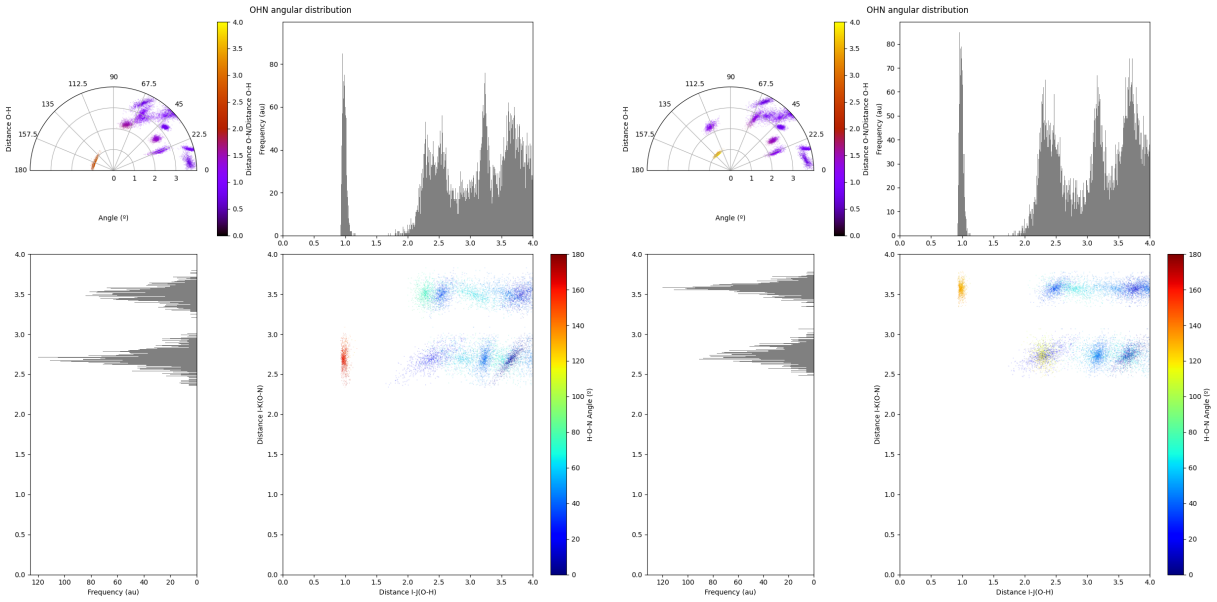
Starting with the NMS method, following the procedure developed in Section 5.1, five geometry distortions were obtained. After the geometry optimization it turned out to be only two different conformers, named conformer A and conformer B (keto and amine groups in "anti" and "syn" respectively) whose geometries are shown in Figure 7.1.



(a) Conformer A (b) Conformer B

Figure 7.1: Conformers obtained for the alanine molecule.

Over both of them, NMS was applied generating 10000 geometries in each case. These structures were the input given to the MM2SF code to generate all the radial and angular distributions of each conformer. In order to give an example of the use of these symmetry functions to condense chemical environments, the OHN trio of atoms was taken for conformer identification. Here, the angular distributions generated place the first atom, oxygen, as the central element by analysing the angle generated with the other two atoms, hydrogen and nitrogen. Together with the O-H and O-N radial distributions, the results grouped in Figure 7.2 were obtained. (full-page figures available in auxiliary Figures 9.2 and 9.3).



(a) Conformer A

(b) Conformer B

Figure 7.2: Observed angular distribution of the O-HN (i-jk) atomic trio of both alanine conformers. In both sub-figures, the radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

The geometry of each conformer can be easily associated to its plot. Starting with conformer A, in its radial dispersion plot a red area representing a trio where the O-N distance is approximately 2.7 Å and the O-H distance is approximately 1 Åcan be found. In this plot, two clearly distinguishable horizontal bands are formed, which can be related to the O-N distances in the "anti" or "syn" position. Since the trio mentioned is in the band with the smaller O-N distance it can be deduced that the alcohol group (due to the O-H distance equal to 1 Å) is in "syn" with the nitrogen, i.e. the keto group and the amine group are in "anti", as expected.

On the other hand, conformer B presents, in its dispersion plot too, an orange zone corresponding to the alcohol group in "anti" to nitrogen, i.e. the keto group and the amine group in "syn". Regarding the polar plots, both are very similar, except for a couple of zones that appear displaced: one at around 1 Å for both conformers and another at a little more than 2 Å at about 112.5º in conformer A which goes to about 70º in conformer B. This again reflects the differences in geometry resulting from the "anti" and "syn" conformations of the keto and amino

groups.

After the NMS methodology, the MD strategy detailed in Section 5.2 was carried out. With this process, 10000 final geometries were obtained, forming the input for the MM2SF code. Taking again the OHN trio, the results obtained can be seen in Figure 7.3. (full-page figure available in auxiliary Figure 9.4)

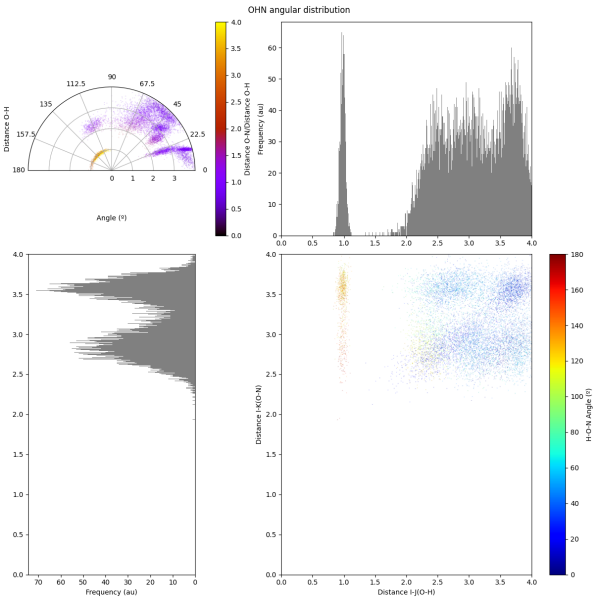


Figure 7.3: Observed angular distribution of the O-HN (i-jk) atomic trio of the selected alanine geometries from the MD. In both sub-figures, the radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

Comparing the results obtained with NMS and MD, the following can be deduced:

- With MD the most dispersed points are obtained, being able to differentiate areas in common with NMS but occupying more space (greater diversity). This is a demonstration of a good data set: occupying as much chemical space as possible but always forming well-defined clusters (associated with specific groups of the molecule).
- With MD, comparing the polar plots, there are geometries of both conformers. In other words, throughout the MD, both conformer A and B have been visited.
- In the dispersion plot obtained with MD it is more difficult to see the different zones. This

is because while with NMS 10000 geometries were obtained for each conformer (20000 in total), with MD just 10000 geometries were used to describe both conformers.

All in all, the MD methodology seems to be more efficient in creating a diverse dataset that includes the different conformers. Additionally, NMS presents a problem not yet mentioned. Whereas for alanine 5 distortions were obtained with Open Babel, for example, for lysine 173 and for arginine 905 were obtained. All these distortions have to be optimized with Gaussian and compared one by one to find out which ones correspond to the same conformer.

For these reasons, **it was decided to use MD as the only method to build the database**, being a more automated and similar procedure for all molecules.

7.1.2. MD over all amino acids

As indicated previously, in this project the aim was to generate the largest possible dataset in the time available. After the time invested in generating the geometries themselves, the QTAIM properties extraction step developed in Section 5.3 must be carried out. The biggest bottleneck in the process, the AIMQB package step, takes about 15 minutes on average, bringing the total process time to about 20-25 minutes per geometry. Therefore, given that the time available for this particular step was estimated to be a maximum of 75 days, it was decided to focus on a total number of geometries in the database around 3500.

On the other hand, it must be borne in mind that a small molecule, for example alanine, needs fewer geometries to define its conformational diversity than a larger molecule with greater conformational variety. For this reason it was decided that the number of geometries to be generated for each amino acid would depend on the number of atoms.

Thus, knowing that the total sum of atoms of the 18 amino acids is 350, for each molecule a number of geometries equal to its number of atoms multiplied by 10 was taken. This adds up to a total of exactly 3500 structures for our dataset. In the auxiliary Table 9.1 the number of geometries taken for each amino acid is indicated.

In order to carry out MDs long enough to allow the molecules to visit their different conformers, 50000 steps of 1 fs were made, as for alanine, of which 1 in 5 was saved. This generated a set

of 10000 structures for each amino acid, from which randomly the corresponding number of geometries were taken following the auxiliary Table 9.1.

On these 3500 xyz files, the procedure to obtain the QTAIM properties indicated in Section 5.3 was carried out.

All calculations ended normally, the Poincare-Hopf theorem was fulfilled in all cases and no Non-Nuclear Attractors were found. But, regarding the reconstruction of the total charge from its atomic components two geometries were found with errors notoriously higher than the average: while 3498 geometries had molecular charges between +0.004 and -0.004, for a histidine frame a total charge of about -0.050 is obtained and for proline frame a total charge of about -0.250. For this reason, these structures were replaced by others in close simulation times (which should be similar in geometry). With this change, the reconstruction of the total charge obtained for each geometry can be seen in auxiliary Figure 9.5.

All these errors in the charge reconstructions are acceptable (thousandth of an electron), so the QTAIM property calculation can be consider as successfully completed.

7.2. NNAIMQ re-validation

In this section, the results obtained with the dataset built for the re-validation of the pre-built model called NNAIMQ, used for the prediction of the partial charges, will be shown.

Once the partial charges have been calculated, the next step is to predict them using the code of *NNAIMGUI*, following the procedure explained in Section 5.4, where the model selected in this case was NNAIMQ. This generates a prediction of the partial charge, which can be compared with the one taken as reference, i.e. the one calculated with *AIMQB*, in different ways.

In Figure 7.4, the dispersion graph where the x-axis corresponds to the reference values and the y-axis corresponds to the predictions can be seen. It can be seen that all the points, independently of the atom, are centred on the diagonal and with a low average deviation from the diagonal (the next figure will show quantitatively how low this deviation is), which means that the predictions are, in general, reasonably close to the reference.

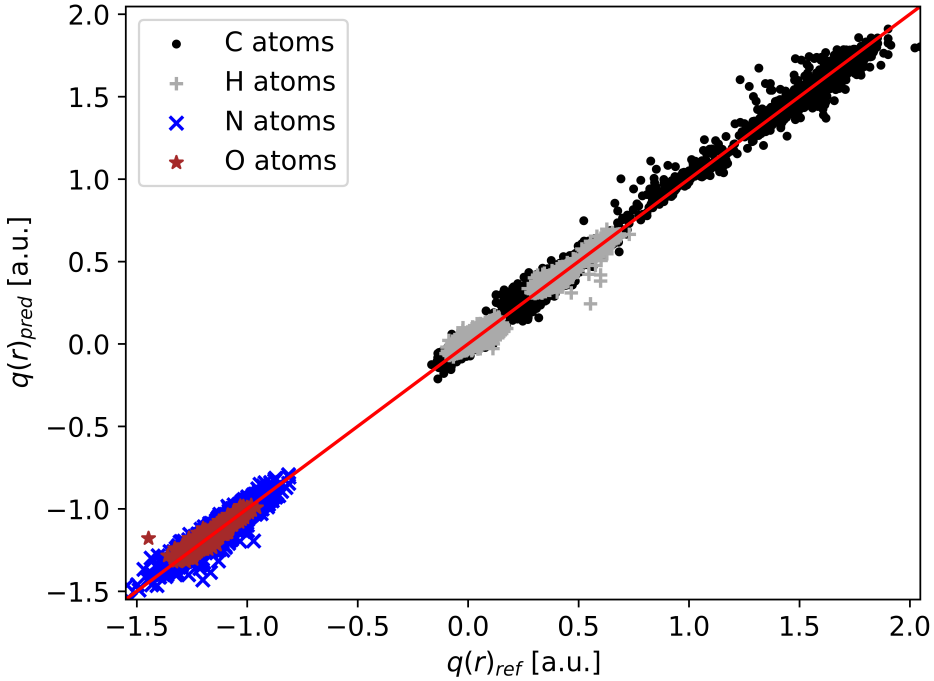


Figure 7.4: Dispersion plot split by atom type in the prediction of partial charges with NNAIMQ (y-axis) using as reference the M06-2X/def2-TZVP level of theory from the package AIMQB (x-axis).

On the other hand, it can be seen the values in the partial charge that different atoms can take: nitrogen and oxygen move in similar zones of negative values (negatively charged), this being a reflection of their tendency to withdraw nearby electron density owing to their large electronegativity. Hydrogen has two clearly defined regions, one closer to 0 and the other centred at +0.5 electrons (positively charged), which may be associated with less or more charge-subtracting environments respectively, the latter being most likely related to the nitrogen and oxygen atoms mentioned above. Finally, in the carbon atom the widest range of possible values in the partial charges, varying from charges close to zero (even somewhat negative) to charges of +2 electrons, can be found. This reflects the great variety of environments that carbon has in the database studied. And it could not be otherwise since the amino acids studied (as in all the compounds of organic chemistry) have carbon as the main skeleton of their structures.

Afterwards, in order to make a quantitative analysis of the performance of the NNAIMQ model for the partial charge prediction, the absolute errors ($|q_{ref} - q_{pred}|$) were calculated for each particle and, once separated by atom type, they were ordered from smallest to largest.

Therefore, Figure 7.5 presents the s-curves for each atom, which show how the error increases as the percentage of accumulated atoms increases (sorted from smallest to largest error).

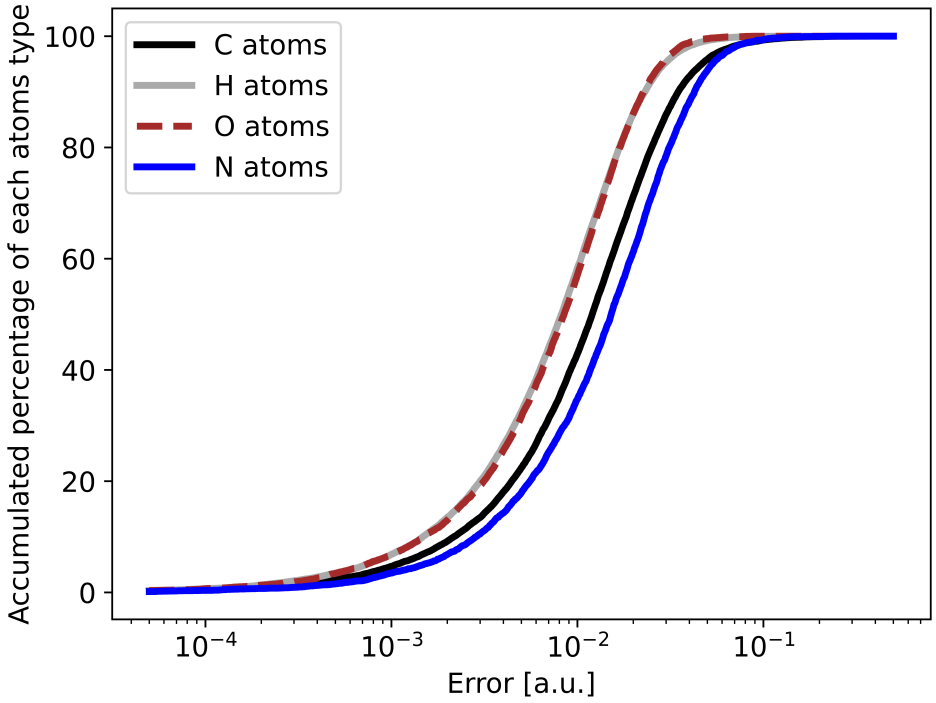


Figure 7.5: S-curves of different atom types for the prediction of partial charges with NNAIMQ using as reference the M06-2X/def2-TZVP level of theory from the package AIMQB.

Consequently, with these s-curves and their displayed data values the following deductions can be made:

- Regardless of the atom type, all the errors obtained are focused in the range of 10^{-2} electrons. This range is consistent with the defined NNAIMQ errors in the Reference[68].
- 80 % of the carbons have errors smaller than 0.026 electrons whereas 95 % have errors smaller than 0.047 electrons.
- 80% of the hydrogen and oxygen atoms have errors lower than 0.017 electrons and 95% have errors under 0.029 electrons.
- 80% of the nitrogen atoms have errors of less than 0.032 electrons and 95% have errors smaller than 0.054 electrons.

All-in-all, considering the interpolation range of the model, for molecules whose chemical environments are similar to those used for the re-validation it can be stated that **with 95% of confidence, the errors committed in the prediction with NNAIMQ will be: for carbon atoms less than 0.047 electrons, for hydrogens and oxygens less than 0.029 and for nitrogens less than 0.054**. Additionally, with the reference data and predictions, the MAEs and RMSEs for each atom type shown in Table 7.1 were calculated.

	C atom	H atom	O atom	N atom
MAE	0.017	0.011	0.011	0.0200
RMSE	0.025	0.015	0.014	0.0273

Table 7.1: MAE and RMSE for the different atom types for the prediction of the partial charges with NNAIMQ using as reference the M06-2X/def2-TZVP level of theory from the package AIMQB.

Therefore, **the obtained MAEs for the prediction of the partial charge are in the 0.011-0.020 electrons range. These results are really close to the 0.007-0.016 electrons range reported in the original work** (Ref.[21]).

7.3. FFNN model development for LI prediction

In this section, the results obtained at each key step of the FFNN model development for the prediction of the loalization index will be presented. In a nutshell, to create a model suitable for *NNAIMGUI*, it is necessary to start by building representative ACSFs of the chemical environment, then to choose the best hyperparameters for training and finally to train and implement them into *NNAIMGUI* for use in the prediction of new systems.

7.3.1. Atom Centred Symmetry Functions creation

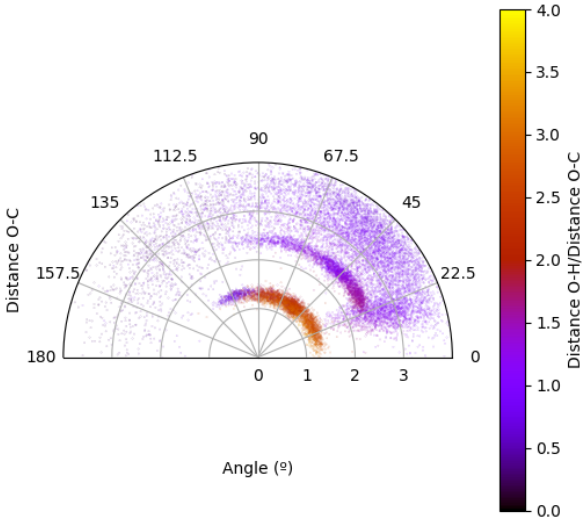
With the database generated, the first step in the development of a FFNN model is the construction of symmetry functions representative of the chemical environments addressed with the database. To do this, the procedure detailed in Section 5.6 was initially applied to the 3500 geometries that made up the database.

Analyzing the results obtained, some issues was noticed. Of the 18 initial amino acids, only 7 (aspargine, aspartic acid, glutamine, glutamic acid, serine, threonine and tyrosine) have three or more oxygen atoms in their structure and only 2 (arginine and histidine) have three or more nitrogen atoms in their structure. This causes a problem when studying the O-OO and N-NN angular distributions, since the number of geometries that contribute to them will be very small. For this reason, it was decided to enhance the aforementioned trios by adding more geometries containing them. Additionally, with the general aim of improving the description of all the chemical environments, the number of geometries of the other amino acids was also increased, although to a lesser extent. Specifically, the number of atom geometries for each amino acid went from 10 to 60, 40 or 20 depending on whether they had 4, 3 or less nitrogen or oxygen atoms respectively. The exact number of geometries taken for the final ACSFs can be found in the auxiliary Table 9.2.

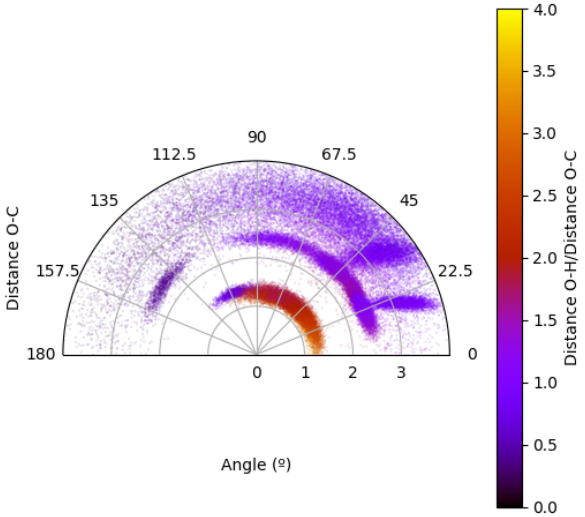
As a comparative example of the effect of these corrections, the polar plots for the O-CH and O-OO trios with and without trio boosting can be seen in Figure 7.6. Additionally, their radial functions and dispersion plots can be found in auxiliary Figures (9.6, 9.7, 9.8, 9.9).
In both trios **there are noticeable improvements while using the boosted database to construct the ACSFs**. First, in the O-CH trio, radial functions with smoother curves can be appreciated, as well as three zones in the polar plots that, at least visually, can only be appreciated in the case of the boosted database (Figure 7.6b): one at O-C distances of around 3 Å, and with an O-CH angle of about 140º and two at O-C distances of 2.5 AA, and approximate angles of 20º and 40º. Secondly, with respect to the O-OO trio, in the radial functions the improvement is noticeable, going from staggered jumps to perceptible curves.

Regarding its polar plot (Figure 7.6d), although it is still a bit scarce, two main areas with the enhanced database can already be visually intuited: one at an O-O distance of just over 2 Å, and covering O-OO angles from 45º to 135º; and another area with an O-O distance of about 3 Å, and with an O-OO angle of around 30º.

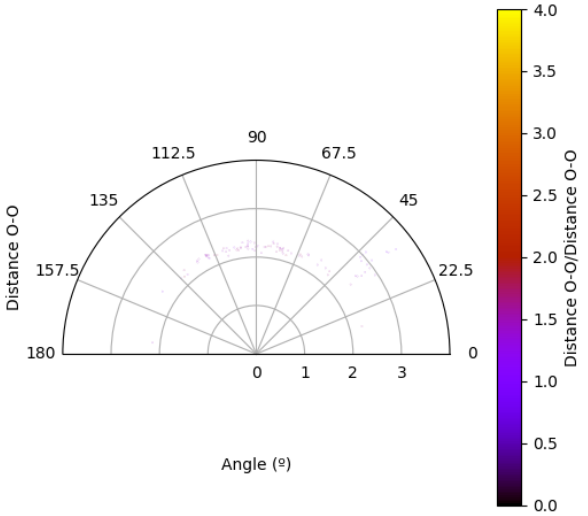
In each zone of the radial dispersion and polar plots of the different trios, relevant information of the chemical environments of the molecules is condensed. Nevertheless, since the aim of the work is not the interpretation of these functions, the relationship of each of these zones with functional groups or patterns in the geometries of the amino acids will not be further explored.



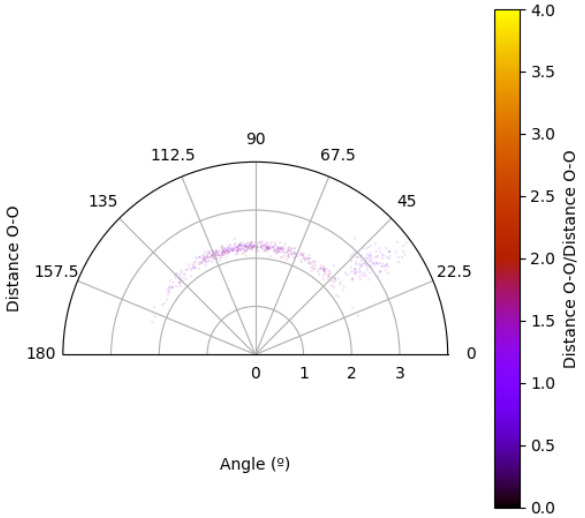
(a) O-CH with training database



(b) O-CH with boosted trios database



(c) O-OO with training database



(d) O-OO with boosted trios database

Figure 7.6: GMM reconstructed polar plots of the O-CH and O-OO atomic trios with two different databases, where it is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

7.3.2. Benchmarking of the hyperparameters

Once the ACSFs to be used for model development have been selected, the next step is to find the best combination of hyperparameters for the specific objective of the model.

To do this, a portion of the complete database, in this case 1000 randomly selected geometries,

is taken in order to reduce the resource consumption of the benchmarking. In this way, a training is carried out with each possible combination of parameters studied.

The possible alternatives considered for each hyperparameter presented in Section 5.5 are the following:

- Fraction of data used for training: fixed to 0.8, i.e. 80% of data for training and 20% for the test step.
- Validation split factor: fixed to 0.2, i.e. 20% of the training for the validation step.
- Maximum number of epochs: fixed to 100000, big enough for allowing convergence.
- Patience: 10 and 20 were considered as candidates.
- Learning rate: 0.01, 0.001 and 0.0001 were considered as candidates for the initial values (all models were varying the learning rate in training).
- Loss metric: fixed to MSE because is more strict with little deviations.
- Optimizer algorithm: RMSProp and Adam were considered as candidates.
- Number of hidder layers (model architecture): 2, 3 and 4 were considered as candidates.
- Number of neurons per layer (model architecture): 5, 10 and 15 were considered as candidates.
- Activation functions of each hidden layer (except the last that must be linear)[2]: ReLU, SELU and tanh were considered as candidates.

For each possible combination of candidates, a different model was taken into account, bringing the number of trained FFNNs to 324 (hence the use of a reduced database). For each of them, errors in the training (with the validation step) and in the test (with the test step) were obtained for each type of atom. By grouping the errors by possible candidates considered for each hyperparameter, the results for each atom type are presented below.

[2] As indicated in the Section 4.4.3, in property predictions the last activation function has to be linear

Carbon atom

For the carbon atoms, the resulting benchmarking for the different hyperparameters generates the results grouped, as boxplots, in the Figure 7.7.

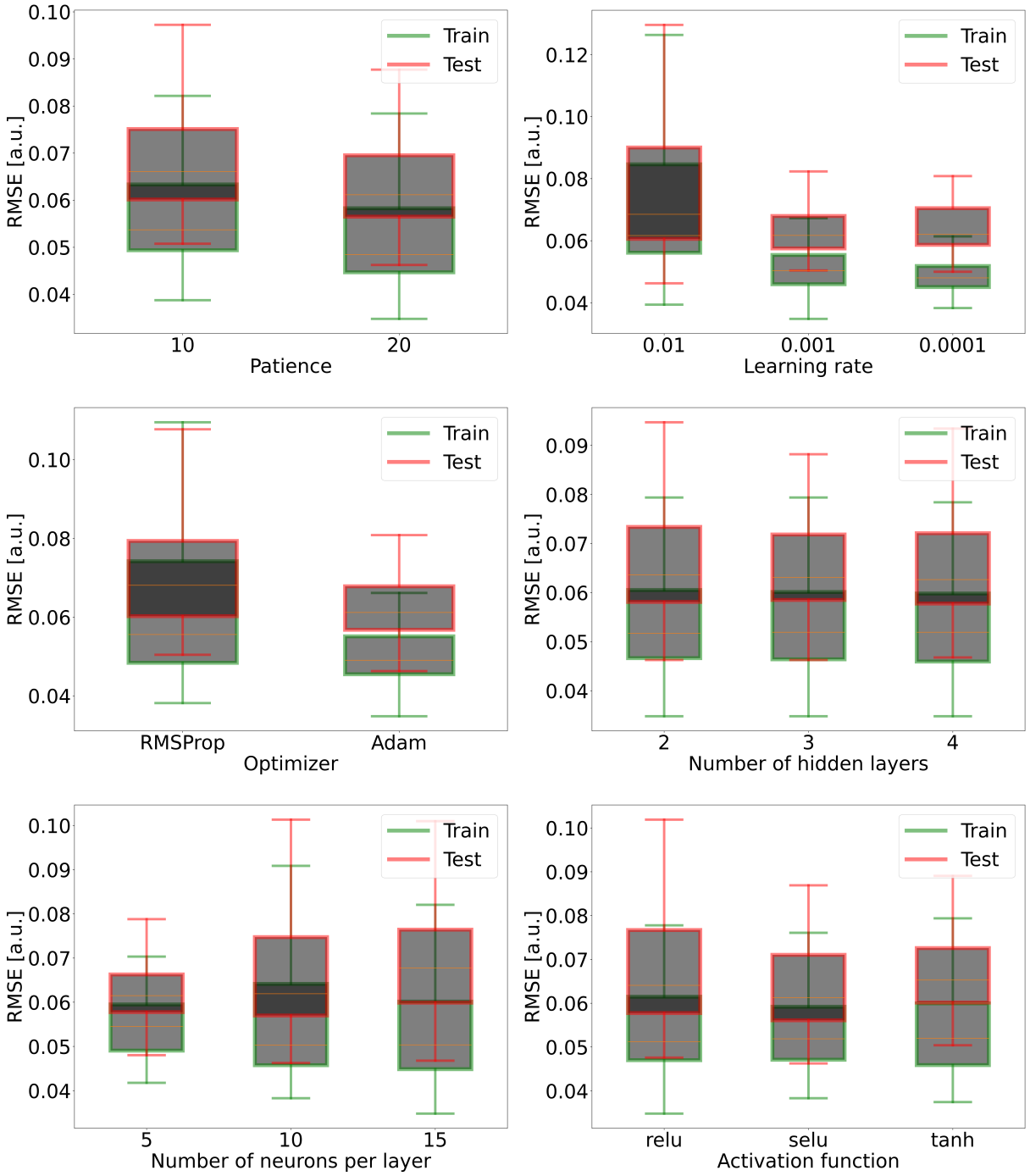


Figure 7.7: Benchmarking for several hyperparameters in the LI prediction for carbon atoms with a boxplot representing the RMSE distribution of each parameter option studied.

For patience, 20 is clearly the most appropriate value (lower errors in both train and test). Regarding the learning rate, 0.001 was taken as it is the lowest median. On the other hand, although with 0.0001 lower extreme values are obtained, there is some risk of overfitting given that there is a greater discrepancy between training and test. In the optimizer, Adam is clearly superior to RMSProp. With the hidden layers, given the similarity of the three boxplots, a direct relationship between the number of hidden layers and the error obtained cannot be found. It was decided 3 hidden layers as this is the intermediate option. Regarding the number of neurons per layer, clearly smaller errors are obtained with 5. And finally, selu was chosen as the best activation function because it generated slightly smaller errors in the test.

Taking into account all these results, **the model with patc=20, lr=0.001, optimizer=Adam, architecture=(5,5,5) and activation function=selu can be considered as the best candidate for carbon atoms**.

Hydrogen atom

The resulting benchmarking for the different hyperparameters in the hydrogen atoms can be seen in the Figure 7.8.

With respect to patience, 20 is once again the most appropriate value. Regarding the learning rate, the same happens as with the carbon: with 0.001 the median is lower and with 0.0001 one can sense some risk of overfitting, given that there is a greater discrepancy between train and test in this second case. Regarding the optimizer, Adam is clearly superior to RMSProp. In the hidden layers, given the similarity of the three boxplots, a direct relationship between the number of hidden layers and the error obtained by the model cannot be found. It was decided to take 3 hidden layers as this was the intermediate option. Regarding the number of neurons per layer, slightly lower results are obtained with 10. And finally, tanh was chosen as the best activation function because it yielded the smallest errors.

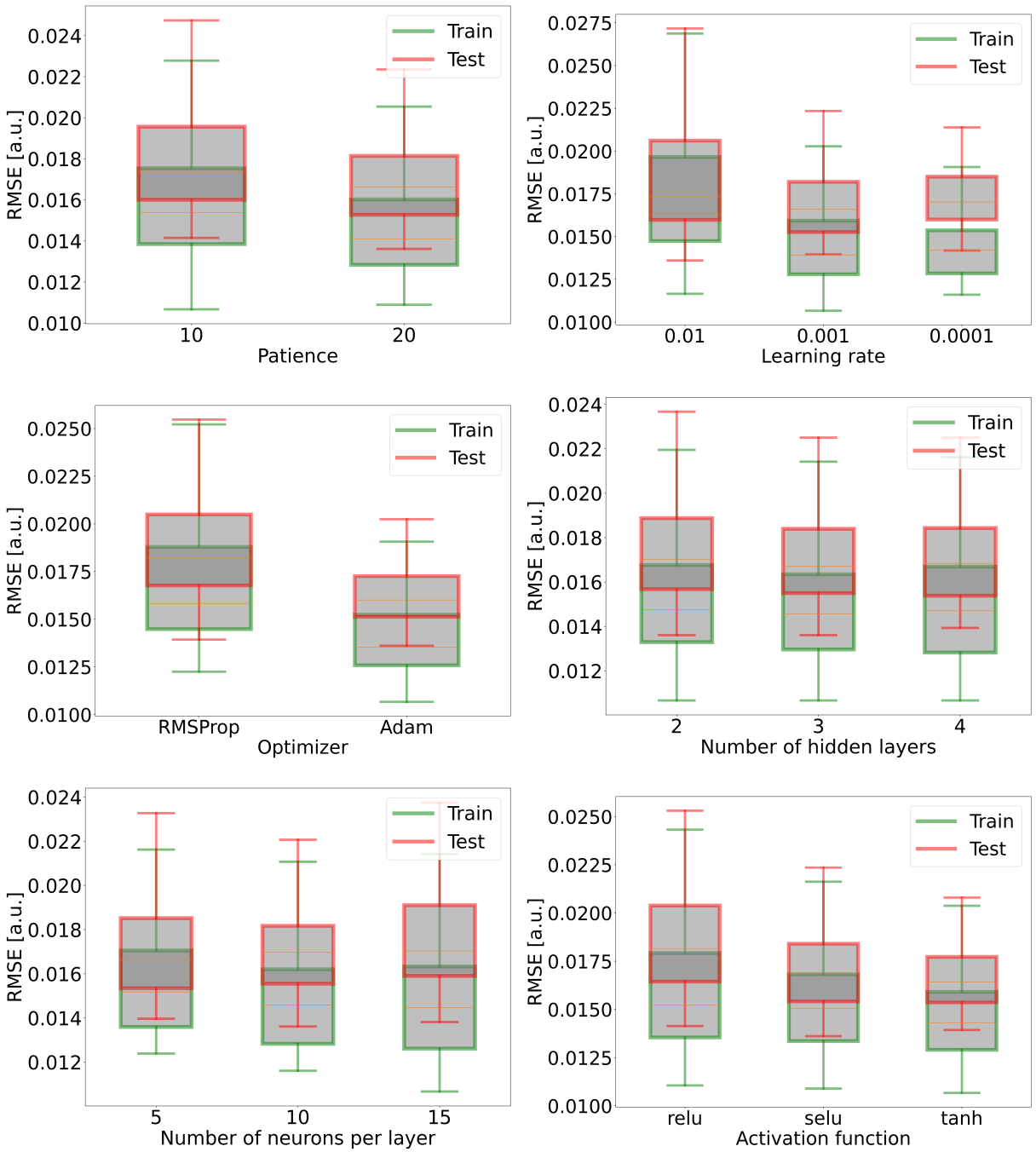


Figure 7.8: Benchmarking for several hyperparameters in the LI prediction for hydrogen atoms with a boxplot representing the RMSE distribution of each parameter option studied.

All in all, **the model with patc=20, lr=0.001, optimizer=Adam, architecture=(10,10,10) and activation function=tanh can be considered as the best candidate for the hydrogen atoms**.

Oxygen atom

In the oxygen atoms, the benchmarking for the different hyperparameters is contain in the Figure 7.9.

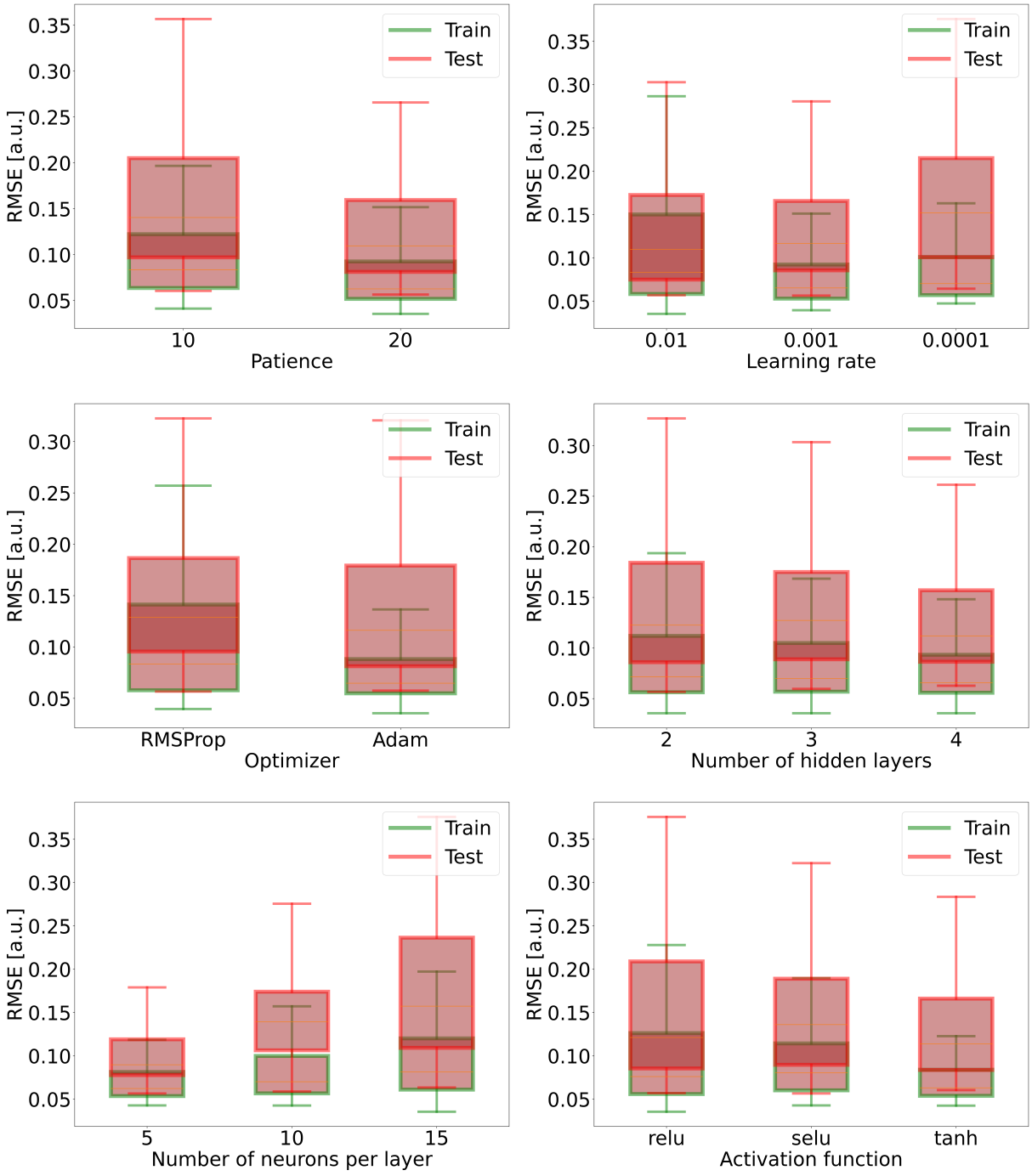


Figure 7.9: Benchmarking for several hyperparameters in the LI prediction for oxygen atoms with a boxplot representing the RMSE distribution of each parameter option studied.

The most appropriate patience is 20 again. With the learning rate, there are doubts as to whether a value of 0.01 or 0.001 would work better, so both options will be evaluated. Regarding the optimizer, the same situation arises, so both alternatives will be checked as well. In the hidden layers, a decreasing tendency of the error as the number of hidden layers increases can be seen, so it was decided to choose 4 layers. The opposite, and more noticeable, happens with the number of neurons per layer: the higher the number of neurons, the higher the errors obtained. For this reason, 5 neurons per layer was chosen as the best option, and finally tanh was chosen as the best activation function because it yields lower errors in average.

With all this, **the model can be considered as fixed hyperparameters patc=20, architecture=(5,5,5,5,5) and activation function=tanh, and as hyperparameters to check lr=0.01 or lr=0.001 and with optimizer=RMSProp or optimizer=Adam as the best candidates for the oxygen atoms**.

This leaves four possible models as best candidates for the oxygen atom.

Nitrogen atom

In the benchmarking for the nitrogen atoms hyperparameters, the results obtained can be seen in the Figure 7.10.
By comparison with Figure 7.9 it can be seen that the morphology of the boxplots in the nitrogen atom is practically the same as for the oxygen atom, except for the learning rate, 0.01 is the more appropriate value in the nitrogen due to its lower median and errors in general in the test.
After the analysis, it is concluded that the best models for the nitrogen atom are exactly the same as those for oxygen except in the learning rate.

It can be translate as that **the model can be considered as fixed hyperparameters patc=20, lr=0.01, architecture=(5,5,5,5,5) and activation function=tanh, and as hyperparameter to check the optimizer=RMSProp or optimizer=Adam as the best candidates for the nitrogen atoms**.

This leaves two possible models as best candidates for the nitrogen atom, also being among the best options for the oxygen atom.

Having determined the potentially best models for each type of atom, a second step in the

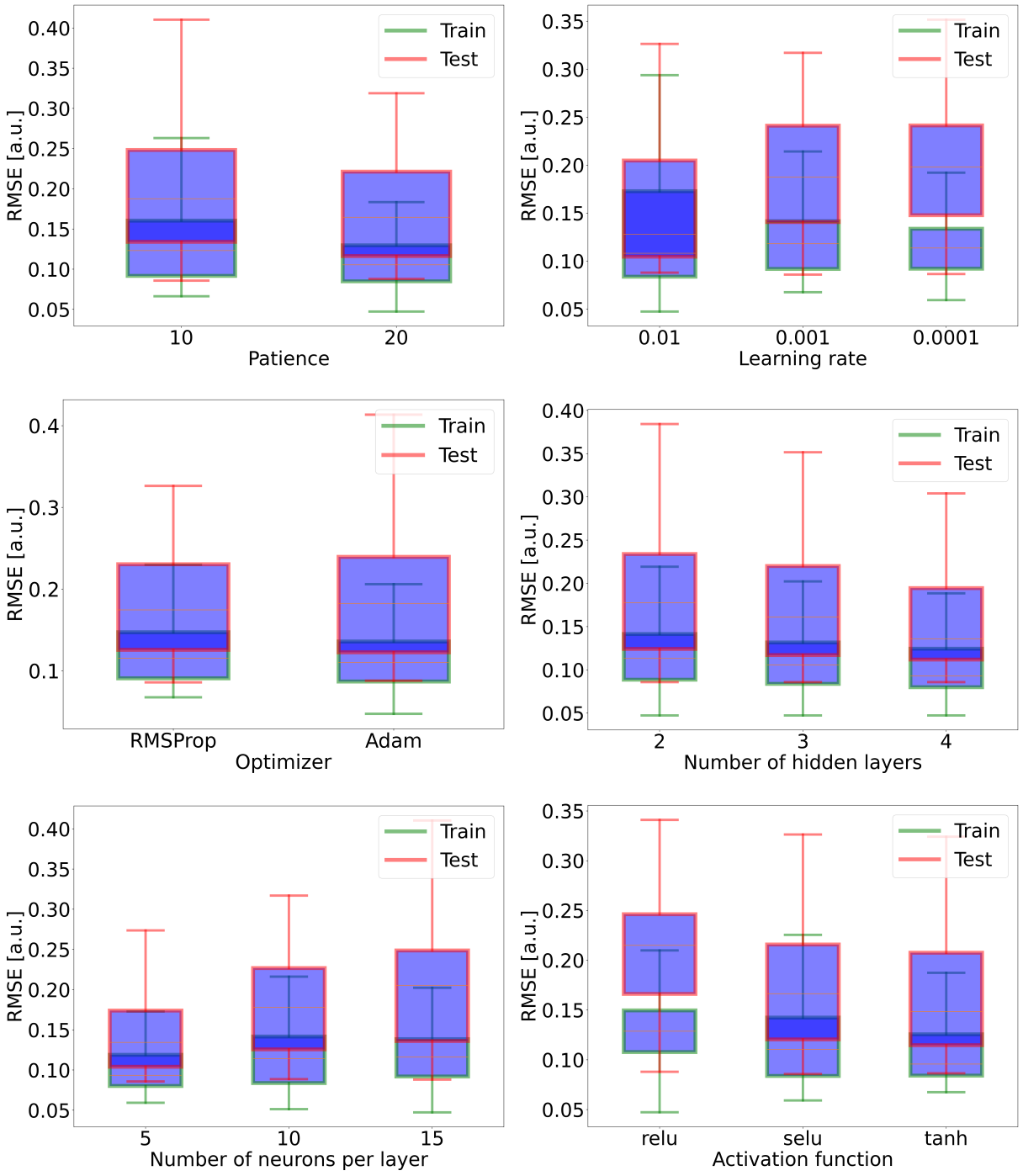


Figure 7.10: Benchmarking for several hyperparameters in the LI prediction for nitrogen atoms with a boxplot representing the RMSE distribution of each parameter option studied.

benchmarking was carried out. On these final candidates, 6 models in total, the training was performed again but this time with the whole database.

Thus, the errors obtained in the training and in the test for each type of atom are shown in the

Table 7.2.

patc	20	20	20	20	20	20
lr	0.001	0.001	0.01	0.01	0.001	0.001
optimizer	Adam	Adam	RMSProp	Adam	RMSProp	Adam
architecture	(5,5,5)	(10,10,10)	(5,5,5,5)	(5,5,5,5)	(5,5,5,5)	(5,5,5,5)
activ.	selu	tanh	tanh	tanh	tanh	tanh
Train C	0.042	0.037	0.066	0.055	0.042	0.037
Test C	0.044	0.041	0.067	0.056	0.047	0.041
Train H	0.014	0.012	0.021	0.016	0.015	0.013
Test H	0.014	0.013	0.021	0.018	0.016	0.013
Train O	0.045	0.039	0.105	0.048	0.056	0.040
Test O	0.063	0.052	0.107	0.053	0.061	0.047
Train N	0.058	0.063	0.152	0.072	0.064	0.057
Test N	0.085	0.087	0.156	0.081	0.075	0.075

Table 7.2: RMSE in [a.u.] for each final model candidate in the validation step (train) and test step for each atom type. For each row of resulting errors, the lowest error is show in green and the worst in red.

From this table, it can be readily realized which is the right choice. One single model is clearly the best in all test sets and in validation sets it is very close to the first place, even surpassing it for the nitrogen atom. For this reason, **the model with patc=20, lr=0.001, optimizer=Adam, architecture=(5,5,5,5,5) and activation function=tanh can be considered as the best for the prediction of the localization index** (over the chemical environments addressed in the database at least), followed closely by the similar model but with architecture=(10,10,10). Additionally, from the results it can be extracted that the biggest influence of error, among the final candidates, is the learning rate, followed by the optimizer. This is because the worst models are found at the same learning rate (0.01) and on the other hand, for similar learning rates, the worst models are found with the same optimizer (RMSProp).

Finally, with the aforementioned model chosen as the best, after training it with *NNAIMGUI* using the whole dataset, it can be analysed further. Therefore, in the next section the behaviour

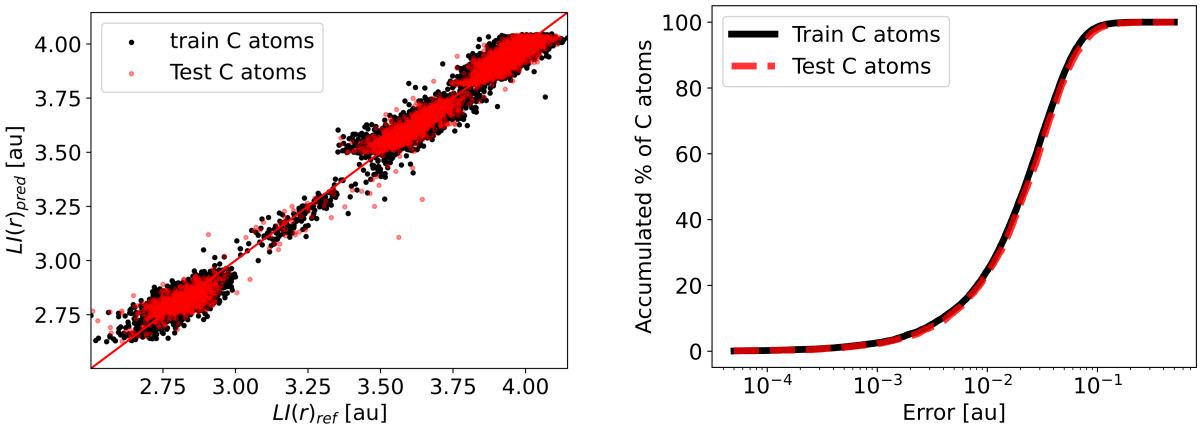
of this model will be studied in more detail, as it was done with the NNAIMQ model.

7.3.3. Performance of the final model

With the final model chosen with bechmarking, training was carried out using the entire database, which generated predictions of the localization indices. Similar to the partial charge, taking the results obtained with *AIMQB*, following the procedure in Section 5.3, as the reference, it is possible to represent the errors made in different ways. In the following, the results obtained following the same procedure as for the partial charge are shown sorted by atom type.

Carbon atom

In the Figure 7.11, both the dispersion plot and the s-curve for the carbon atom can be seen, separated between training set and test set geometries.



(a) Dispersion plot

(b) S-curve

Figure 7.11: Comparison between the reference values and those predicted by the model for the localization index of the carbon atoms in the dataset, separating in both figures the atoms used in the training stage from those used for the test.

Starting with the dispersion plot, as will be seen later, again the largest ranges of values for the property are found. From 2.5 to 4.2 electrons approximately, but with a minor presence in the 3 to 3.5 electron range. This leaves two regions, one at lower LIs where the carbon is more involved in its surroundings reducing its number of localized electrons (possibly bonded with charge-subtracting groups with oxygen or nitrogen atoms) and another at higher LIs where the

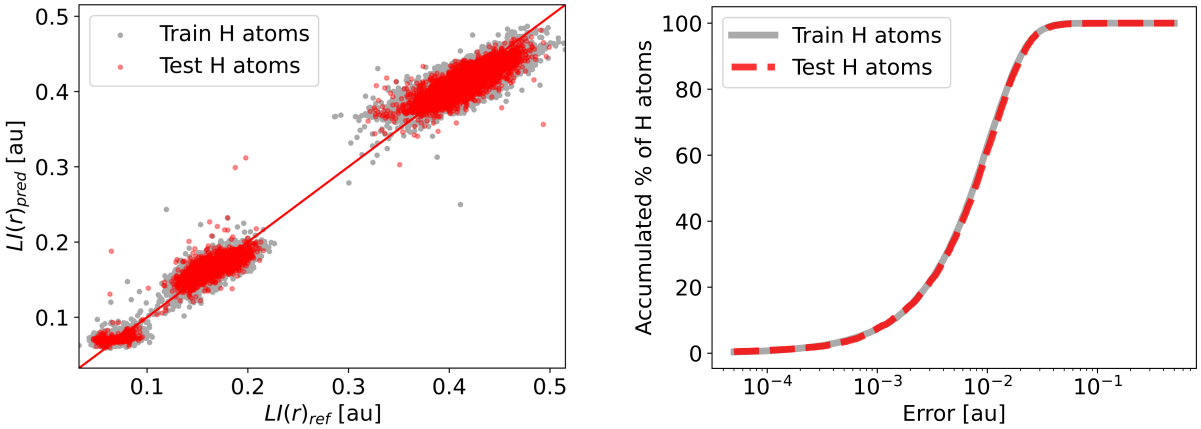
carbon has more fixed electrons in its basin (possibly together with carbon or hydrogen atoms).

By continuing with the s-curve, together with its represented data, results can be derived in the same way as was done for the partial charge:

- Errors in carbon are centred in the 10^{-2}-10^{-1} range, a bit worse than in NNAIMQ.
- 80% of carbon atoms showed errors of less than 0.047 electrons.
- 95% of carbon atoms had errors under 0.082 electrons.

Hydrogen atom

Both the scatter plot and the s-cuve, separated into training and test set for the hydrogen atom, are shown in Figure 7.12.



(a) Dispersion plot

(b) S-curve

Figure 7.12: Comparison between the reference values and those predicted by the model for the localization index of the hydrogen atoms in the dataset, separating in both figures the atoms used in the training stage from those used for the test.

Regarding the dispersion graph, a zone of two clusters in the range from 0 to about 0.2 and another homogeneous zone from 0.35 to 0.5 electrons is observed. This corresponds to two cases, one to smaller LIs where the initial hydrogen electrons are almost entirely involved in bonds or atoms next to it (possibly together with charge-subtracting groups with oxygen or nitrogen atoms involved) and another to larger LIs where the hydrogen has a little more electron

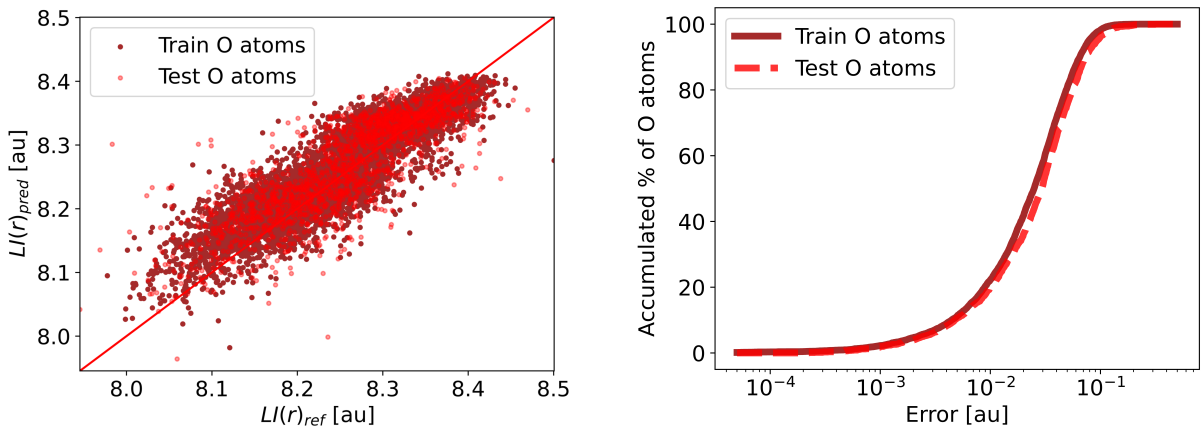
localisation, but remaining very scarce (possibly bonded with carbon atoms).

From the s-curve, together with its represented data, the following can be deduced:

- Hydrogen errors are centred in the 10^{-2} range, similar to those obtained with NNAIMQ.
- 80% of hydrogen atoms presented errors smaller than 0.016 electrons.
- 95% of hydrogen atoms showed errors of less than 0.026 electrons.

Oxygen atom

Continuing with the oxygen atom, its dispersion plot and s-curve, both divided into training and test sets, are shown in Figure 7.13.



(a) Dispersion plot

(b) S-curve

Figure 7.13: Comparison between the reference values and those predicted by the model for the localization index of the oxygen atoms in the dataset, separating in both figures the atoms used in the training stage from those used for the test.

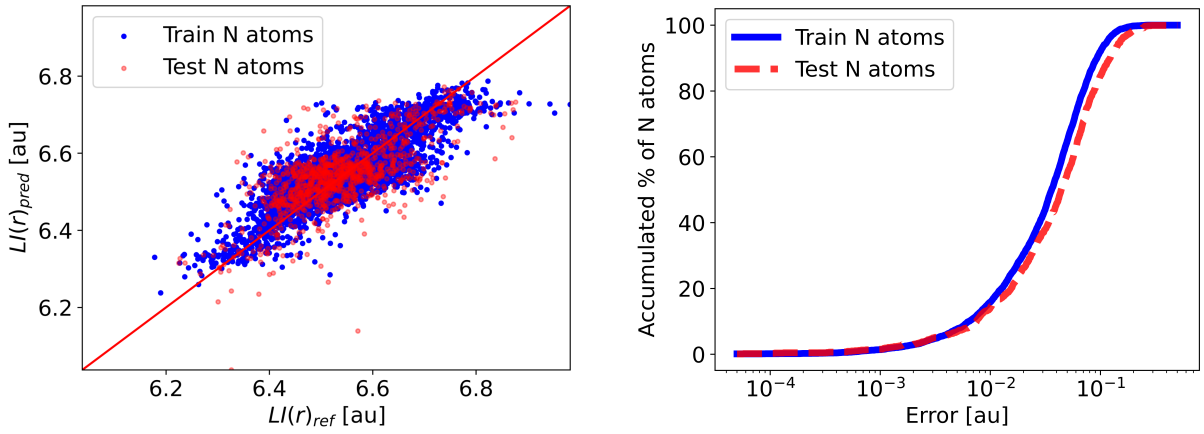
Analyzing the scatter plot, it is visually observed a greater dispersion from the main diagonal, which can be related to a greater error in the prediction. In this atom a single region is found between the 8 and 8.5 electrons. This can be translated into a greater homogeneity in the functionality of oxygen in the systems considered. Moreover, knowing that isolated oxygen has 8 electrons in its structure, it can be affirmed that it is a strong electronegative atom, since it manages in all cases to maintain at least this number located at its basin.

Taking the s-curve, together with its plotted data, allows the deduction of the following results:

- Errors for oxygen in the 10^{-2}-10^{-1} range are obtained.
- 80% of oxygen atoms had errors smaller than 0.057 electrons.
- 95% of oxygen atoms presented errors lower than 0.094 electrons.

Nitrogen atom

Finally, for the nitrogen atom, separating between train and test set, the dispersion plot and s-curve are shown in Figure 7.14.



(a) Dispersion plot

(b) S-curve

Figure 7.14: Comparison between the reference values and those predicted by the model for the localization index of the nitrogen atoms in the dataset, separating in both figures the atoms used in the training stage from those used for the test.

Starting with the dispersion graph, as with oxygen, a unique homogeneous region is found, this time centred between 6.2 and 6.9 electrons. As before, reasoning that isolated nitrogen has 7 electrons in its structure, it can be deduced that it is also a good charge-subtracting group, but less so than oxygen, since it does not have those 7 initial electrons fixed in its basin in any case (contrary to oxygen with its 8 initial electrons). It is noteworthy here that at high values of the LI the same effect of cut-off explained in the carbon atom is experienced.

Using the s-curve, together with its represented data, allows the derivation of the following results:

- Errors for nitrogen centred in the 10^{-2}-10^{-1} range are obtained, slightly worse than with NNAIMQ. In addition, slight differences between the training and test errors are perceived, which could be associated with a very slight overfitting of the model, but without reaching significant levels.

- 80% of nitrogen atoms had errors of less than 0.088 electrons.

- 95% of nitrogen atoms showed errors under 0.153 electrons.

In addition, with the reference data and the predictions, both in the training and in the test, the MAEs and RMSEs for each type of atom considered in the datbase shown in Table 7.3 were calculated.

	Training				Test			
	C atom	H atom	O atom	N atom	C atom	H atom	O atom	N atom
MAE	0.028	0.010	0.031	0.044	0.030	0.010	0.036	0.056
RMSE	0.037	0.013	0.040	0.057	0.041	0.013	0.047	0.075

Table 7.3: MAE and RMSE for each atom type in the training and test of the model for the LI prediction using as reference the M06-2X/def2-TZVP level of theory from the package *AIMQB*.

Hence, the MAEs obtained for the prediction of the localization index in the test set are in the range of 0.010-0.056. These errors are higher than those obtained with NNAIMQ, but still within the same orders of magnitude.

7.4. Application of the developed model in biosystems

As a demonstration of the applicability of the model and in order to further analyse its usefulness, it was decided to study molecules not included in the database. It should be noted that the model should not work well for extrapolating information. However, if the atoms of the molecules are in chemical environments well-defined by the model, they should be predictable with it. And this, regardless of the size of the system they form, even if the FFNN has been trained with very small biosystems (amino acids).

In order to test this, molecules of biochemical interest, of different sizes and biological functions, were explored. As a result, the molecules finally chosen for the study were the following:

- L-hydroxyproline: derived from proline (included in the database) and an important component of collagen (the most abundant protein in animals). Its size and general structure is similar to average amino acids in the database.
- L-ornithine: key intermediate in the urea cycle (ammonia elimination process in all mammals). It is similar in size and structure to average amino acids in the database.
- L-DOPA: precursor of dopamine and other important neurotransmitters. Larger than average amino acids in the database.
- Folic acid: essential for DNA synthesis and repair, cell division and amino acid synthesis. It is significantly larger in size and different in structure than all amino acids in the database.

For the prediction of molecules with the developed model, the procedure of Section 5.4 was followed by choosing a "Custom" model, from which the path of its directory, the name of the property to be calculated (LI) and its units (electrons) were specified. *NNAIMGUI* could be executed without problems, which generated the prediction of the localization index for the atoms of the proposed systems.

In order to study the performance of the model in these new systems, it was decided to carry out the procedure of Section 5.3 where with *AIMQB* these localization indices were calculated by applying QTAIM. Taking the latter as the reference method, the errors made in the prediction of the LI with the model could be studied.

In Figure 7.15, the absolute errors committed in each atom of the L-hydroxyproline, L-ornithine and L-DOPA molecules (ordered from left to right) are shown together with a reference image of their optmized geometries (used in both methods of obtaining the LI). In Figure **??** the same can be seen but for the acid folic molecule.

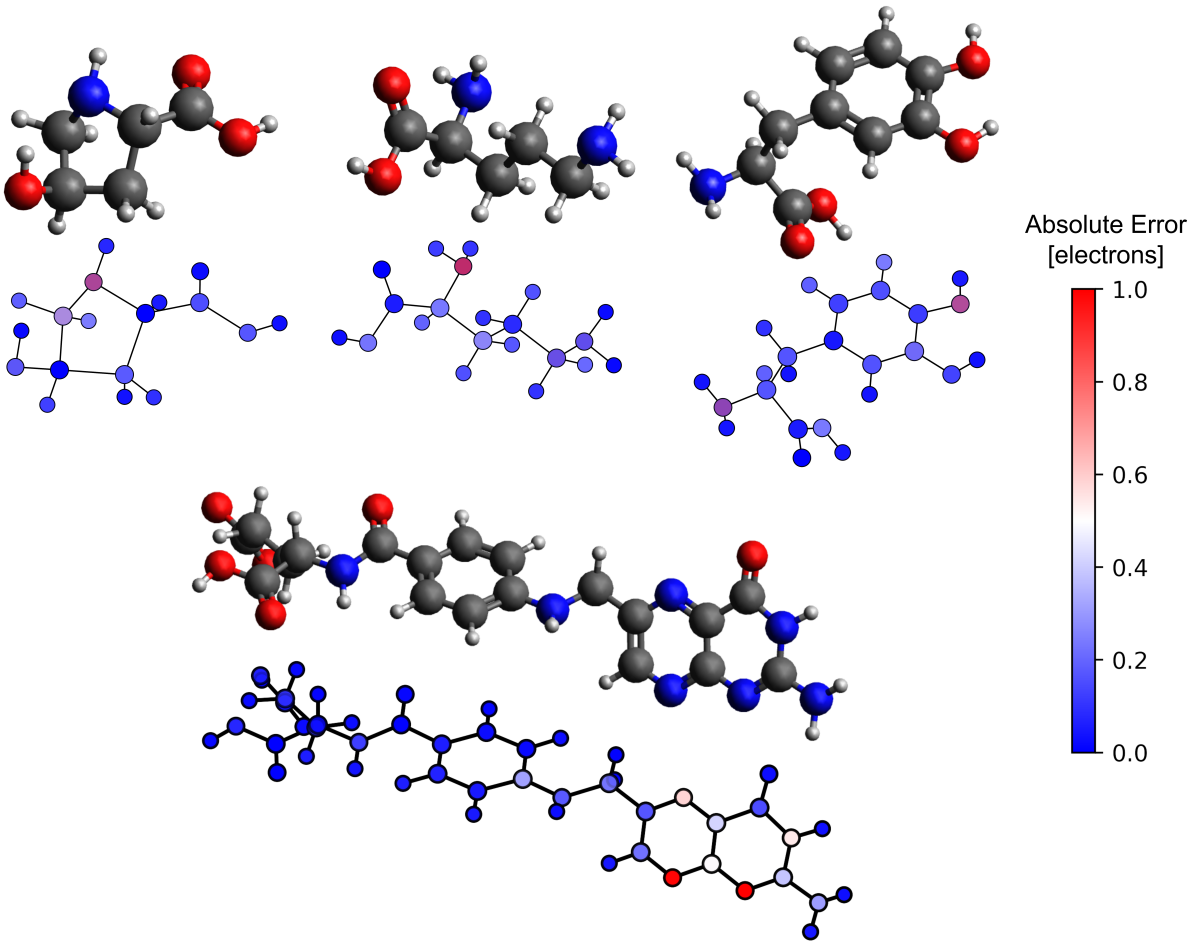


Figure 7.15: Subfigures with molecular structures colored in CPK (top) and absolute errors, in atomic units (electrons), committed in the prediction of each atom (bottom) for the molecules: L-hydroxyproline, L-ornithine and L-DOPA from left to right in the first row and folic acid in the second row.

Additionally, to quantify the accuracy of the model for each system and atom type, in Table 7.4 it can be found the MAEs and RMSEs for each molecule studied, split by atom type.
Before discussing the results, an important factor must be taken into account. The interpolation range has been mentioned before, since it is the area in which a model can be applied without extrapolating the results, which is not recommended in neural networks (as explained at the end of Subsection 4.4.2). Therefore, it would be advisable to be able to know if the systems on which the model is being applied (the proposed molecules in this case) are within this range of interpolation.

	L-hydroxyproline		L-ornithine		L-DOPA		Folic acid	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
C atom	0.063	0.084	0.066	0.075	0.058	0.064	0.155	0.216
H atom	0.043	0.047	0.041	0.046	0.037	0.041	0.032	0.036
O atom	0.076	0.092	0.043	0.060	0.076	0.094	0.040	0.049
N atom	0.229	0.229	0.164	0.164	0.188	0.187	0.556	0.662

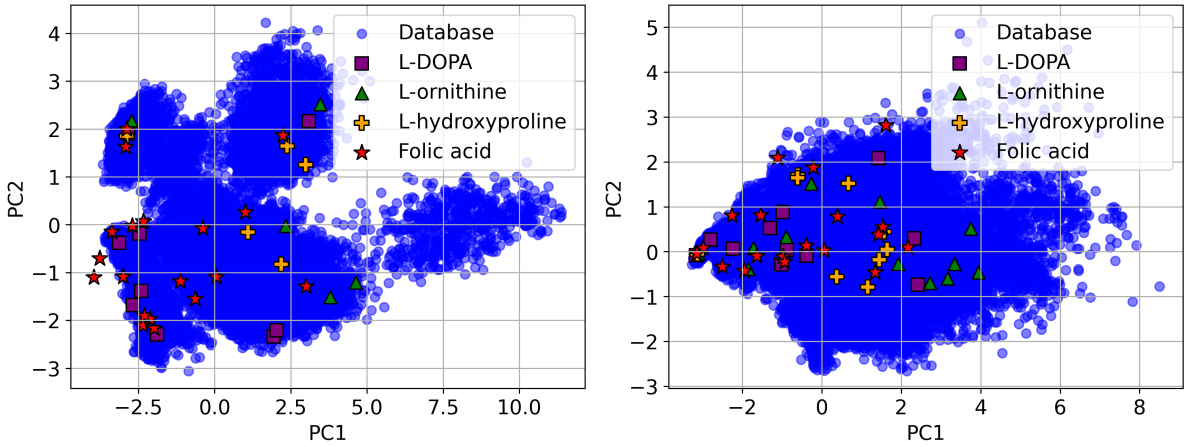
Table 7.4: MAE and RMSE committed in the prediction of the different atoms types in the selected systems.

However, given that the coordinates of each atom have been transformed into specific parameters of the symmetry functions, the interpolation area will be determined according to these parameters, i.e. it will be a region of multiple dimensions (even more than a hundred). Such a region should be impossible to represent, but by making use of a well-known Unsupervised Learning technique called Principal Component Analysis (ML technique mentioned in Section 4.1) this may be possible.

Without going into technical details, this dimensionality reduction technique looks for complex patterns among the data allowing to compact their dimensions into linear combinations of these. In this way, it is possible to impose that the multidimensional function corresponding to the parameters of the symmetry functions be condensed into only two dimensions, called Principal Components (PC1 and PC2).

Applying this same transformation to the training data and to the new molecule data, the PCAs for each atom are grouped in Figure 7.16, where the interpolation zone is defined by the region occupied by the database.

Finally, with all the results presented, these will be discussed molecule by molecule in order to study the applicability and limitations of the developed model .



(a) C atom

(b) H atom

(c) O atom

(d) N atom

Figure 7.16: PCA on each type of atom of the geometries of the dataset and the studied biosystems

L-hydroxyproline

In L-hydroxyproline, analyzing the position of its atoms in the different PCAs, shown as yellow crosses in Figure 7.16, it can be seen that clearly one atom is outside the interpolation zone: the only nitrogen atom of the structure. This fact, surely explains that in the Table 7.4 for this molecule the nitrogen atom doubles and even triples the error obtained with any other atom. Additionally, looking at the absolute errors for each atom in Figure 7.15, it can be found a localization of the error, apart from the nitrogen, in an oxygen (on the left) and in a carbon bonded to the aforementioned nitrogen. These additional errors may come from the distortion

generated by the nitrogen extrapolation case or because they themselves are being somewhat extrapolated (since in PCAs an oxygen and a carbon can be located at the boundary of the interpolation zone).

All in all, **the average errors (both MAEs and RMSEs) for the L-hydroxyproline atoms are smaller than 10^{-1} electrons except for the nitrogen atom, which is 0.23 electrons**.

L-ornithine

In the case of L-ornithine, by observing the position of its atoms in the different PCAs, shown as green traingles in Figure 7.16, several atoms can be found close to the boundary of the interpolation zone. The most worrying are a nitrogen (around PC1=0.9 and PC2=-1.5) and an oxygen (around PC1=-1.5 and PC2=-0.6). Of these two atoms, in Figure 7.15 the nitrogen with the highest error reliably corresponds to the one mentioned above. In addition, the other nitrogen in the structure also has a slightly high error. Nevertheless, looking at the table, it can be affirmed that it is the biosystem whose nitrogen atoms are being best predicted. Altogether, **the average errors (both MAEs and RMSEs) in the L-ornithine are under 10^{-1} electrons except for the nitrogen atoms, which is 0.16 electrons**.

L-DOPA

Regarding L-DOPA, by examining the placement of its atoms in the different PCAs, shown as purple squares in Figure 7.16, two atoms can be found especially at the border of the interpolation zone. These are a nitrogen (around PC1=-0.1 and PC2=-1.6) and an oxygen (around PC1=-1.5 and PC2=-0.4). This fact possibly justifies that in Figure 7.15 the two atoms with the highest absolute error are precisely a nitrogen and an oxygen. It may also be the reason for such relatively high errors in the Table 7.4 for the oxygen atoms.

All in all, **the average errors (both MAEs and RMSEs) for the L-DOPA atoms are less than 10 electrons except for the nitrogen atom, which is 0.19 electrons.**

Folic acid

For folic acid, analysing the location of its atoms in the different PCAs, shown as red stars in Figure 7.16, multiple atoms can be seen outside the interpolation zone or near the border of it. Starting with the ones that are clearly outside, there are two nitrogen atoms (around PC1=-1.8 and PC2=2.7 one and, PC1=3.0 and PC2=-1.5) and two carbon atoms (around PC1=-4 and PC2=-1 both). In addition, two other nitrogens, an oxygen and a pair of carbons can be found at the boundary of this interpolation zone. By looking at Figure 7.15 all these extrapolations can be associated to the same region of the molecule. This can be justified by the lack in the database of complex aromatic systems that include nitrogens in their rings, being the imidazole group of histidine the only example included. This derives, as can be seen in Table 7.4, in the worst errors for carbon and nitrogen atoms among the biosystems studied. Altogether, **the average errors (both MAEs and RMSEs) for the folic acid are less than 0.05 electrons for the carbon and hydrogen atoms, while for the oxygen atoms they ascend to 0.22 electrons and for the nitrogen atoms to 0.66 electrons**.

Additionally, another important fact should be noted. While with the traditional procedure of using Gaussian and AIMQB it took about half a day of computation on the cluster mentioned in Chapter 6 to obtain the QTAIM properties of all the studied biosystems, with *NNAIMGUI* and the trained model it was a question of seconds, on a conventional computer without any clusters, to predict the localization index. This shows the great potential of the model to, for example, track the LI over a whole molecular dynamics (thousands of frames).

8 Conclusions

This Master's Thesis uses Neural Networks to radically reduce the computational cost of making predictions of QTAIM properties without losing accuracy. The great advantage of QTAIM is the transferability of its properties, which in the area of Deep Learning can be translated into the fact that knowing the internal patterns that govern a QTAIM property, it can be predicted in new systems not seen by the model. This, as long as its atoms have chemical environments already considered in the training (within the interpolation range). By merging these two disciplines, new applications previously unfeasible in QTAIM could be obtained with FFNN models, such as: the tracking of a particular property in a live MD, the immediate prediction of a QTAIM property in huge biosystems such as proteins, or even the study of these properties in systems such as large materials. All in all, the advantages of using FFNN models are evident. And as proof of this, the results obtained for the scientific objectives of this project.

Starting with the re-validation of the NNAIMQ model, the range of errors obtained (using MAEs) in the constructed database is only 0.004 higher than the one reported in the original work. For this reason, it can be confirmed with the methodology followed in this project that NNAIMQ is a valid model for the prediction of partial charges in simple biosystems.

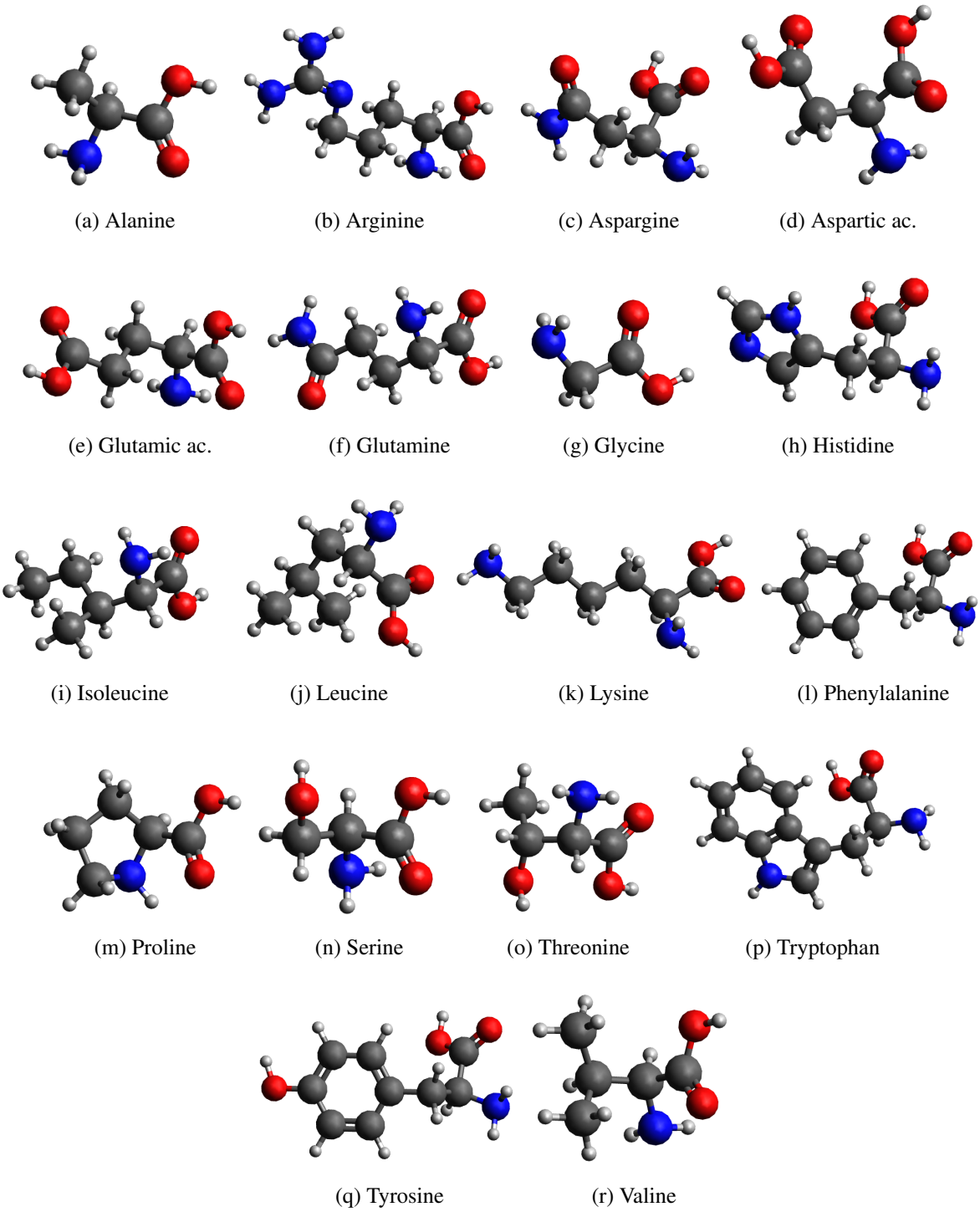
Regarding the second objective, the model for localization index prediction was successfully developed. The resulting hyperparameters proved to be the best for this task, with the database used, were: a patience of 20, a learning rate of 0.001, the Adam optimizer, a neural architecture of 4 hidden layers of 5 neurons each and whose activation functions were tanh for all of them, except the last one which had to be linear. In terms of performance in the test step carried out, an error range (using MAEs) of 0.010-0.056 electrons can be found, remaining in the order of hundredths of electrons.

For the third objective, the developed model was applied to four new molecules of biochemical interest. With them, the relationship between exceeding the interpolation limits and the increase in error was demonstrated. In particular, the atom with the greatest increase in errors with respect to the tests of the previous objective was nitrogen. This could indicate the necessity to increase the interpolation region of this atom in particular, considering more diverse chemical

environments for this atom in the database.

As a general conclusion, it has been possible to demonstrate the great potential of combining artificial intelligence with theoretical chemistry. While in the traditional way one depends on a cluster to calculate the properties of a single geometry in reasonable times (even on the scale of minutes and hours), with NNAIMGUI it is possible to obtain predictions for hundreds of geometries in a matter of seconds. However, for real applications in different biosystems, some improvements with respect to the model developed in this work would be required. First, the sulphur atom, not included in the model, is an indispensable element in the area of biochemistry and it would be advisable to include it despite its increased computational cost. Then, for a more accurate model, it would be recommendable to have a larger database. And finally, in order to be applicable to a variety of scenarios, the database should be more diverse than the one used. However, in order to achieve this, it would be necessary to consider a project with the same methodology, but over a longer period of time, in order to generate a more complete database with the characteristics described above.

9 Supporting material



(a) Alanine (b) Arginine (c) Aspargine (d) Aspartic ac.

(e) Glutamic ac. (f) Glutamine (g) Glycine (h) Histidine

(i) Isoleucine (j) Leucine (k) Lysine (l) Phenylalanine

(m) Proline (n) Serine (o) Threonine (p) Tryptophan

(q) Tyrosine (r) Valine

Figure 9.1: Selected amino acids to generate geometries that form the database of the work.

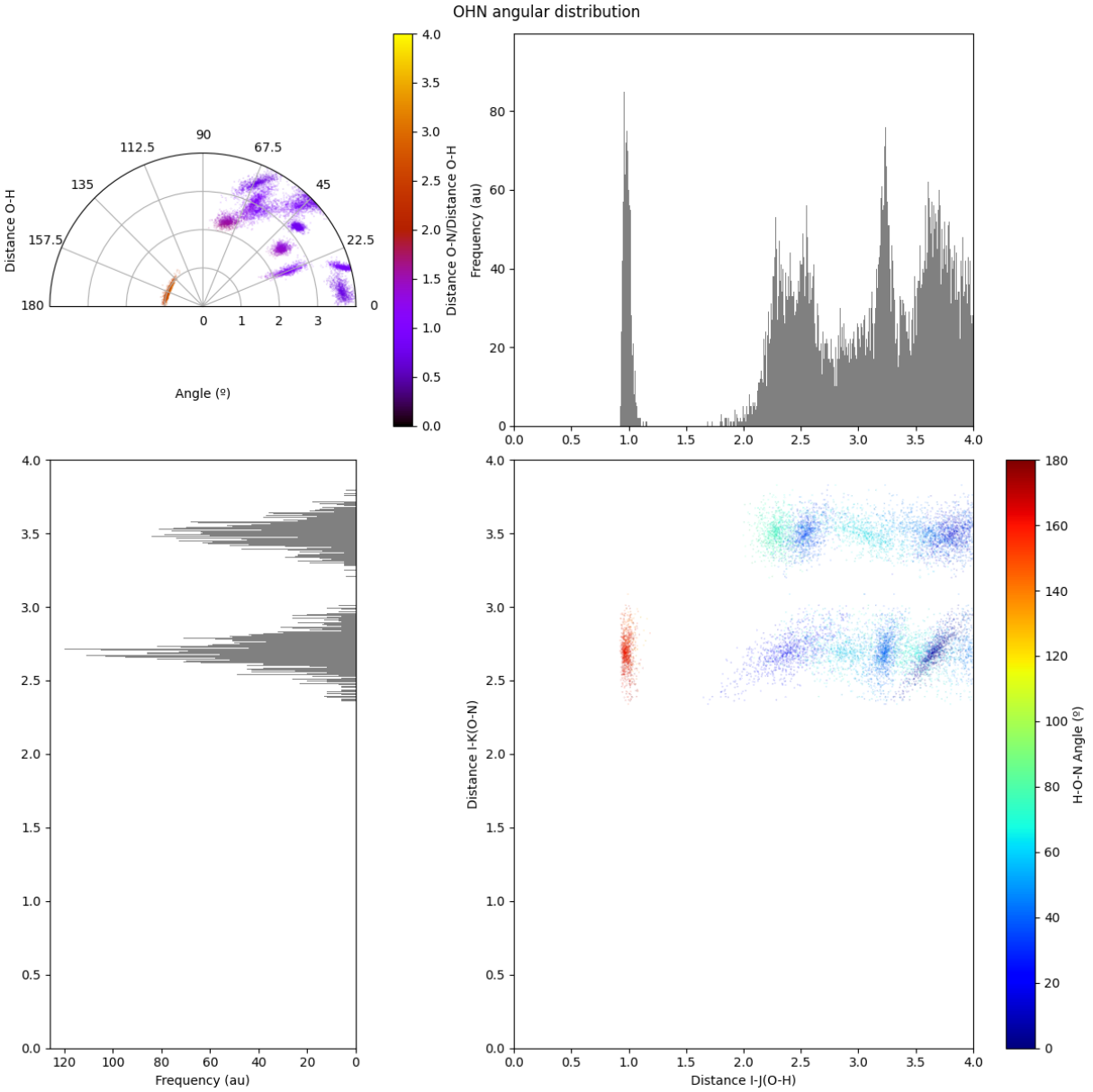


Figure 9.2: Observed angular distribution of the O-HN (i-jk) atomic trio of the alanine conformer A. In the sub-figures, the radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

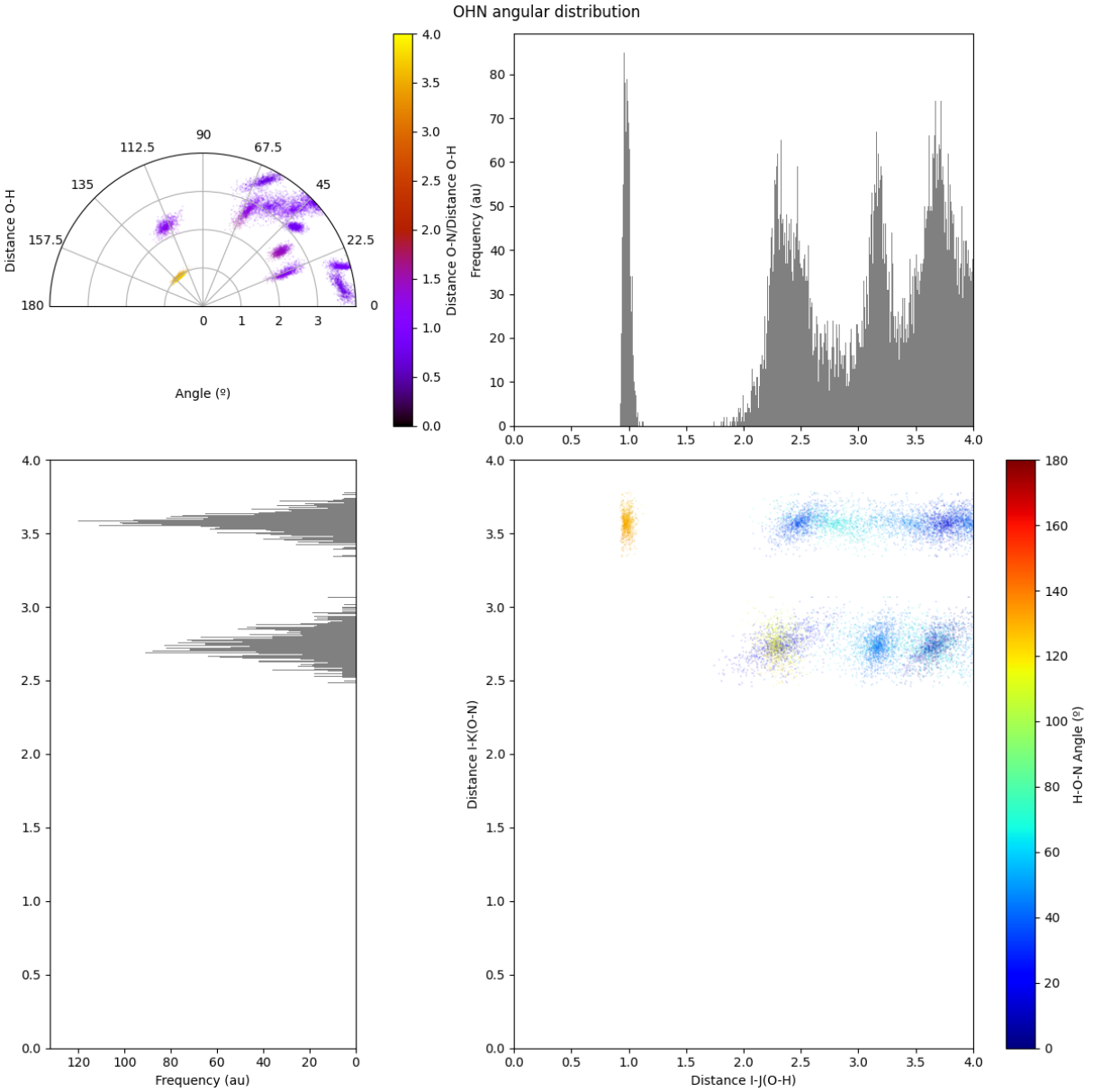


Figure 9.3: Observed angular distribution of the O-HN (i-jk) atomic trio of the alanine conformer B. In the sub-figures, the radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

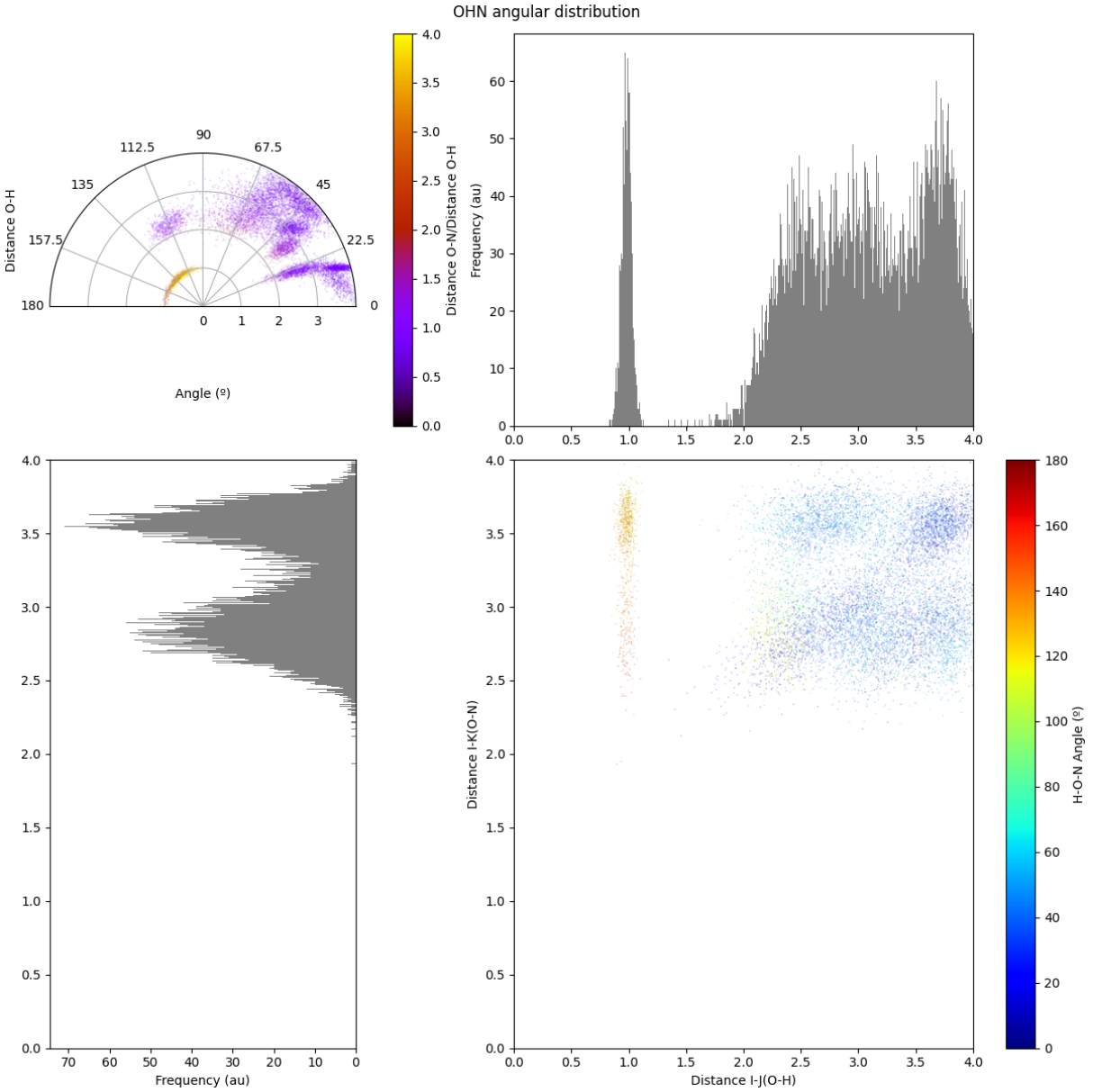


Figure 9.4: Observed angular distribution of the O-HN (i-jk) atomic trio of the selected alanine geometries from the MD. In both sub-figures, the radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

Amino acid	nº atoms	nº geometries
Alanine	13	130
Arginine	26	260
Aspargine	17	170
Aspartic acid	16	160
Glutamic acid	19	190
Glutamine	20	200
Glycine	10	100
Histidine	20	200
Isoleucine	22	220
Leucine	22	220
Lysine	24	240
Phenylalanine	23	230
Proline	17	170
Serine	14	140
Threonine	17	170
Tryptophan	27	270
Tyrosine	24	240
Valine	19	190
TOTAL:	350	3500

Table 9.1: Number of atoms and geometries generated for each amino acid to form the database.

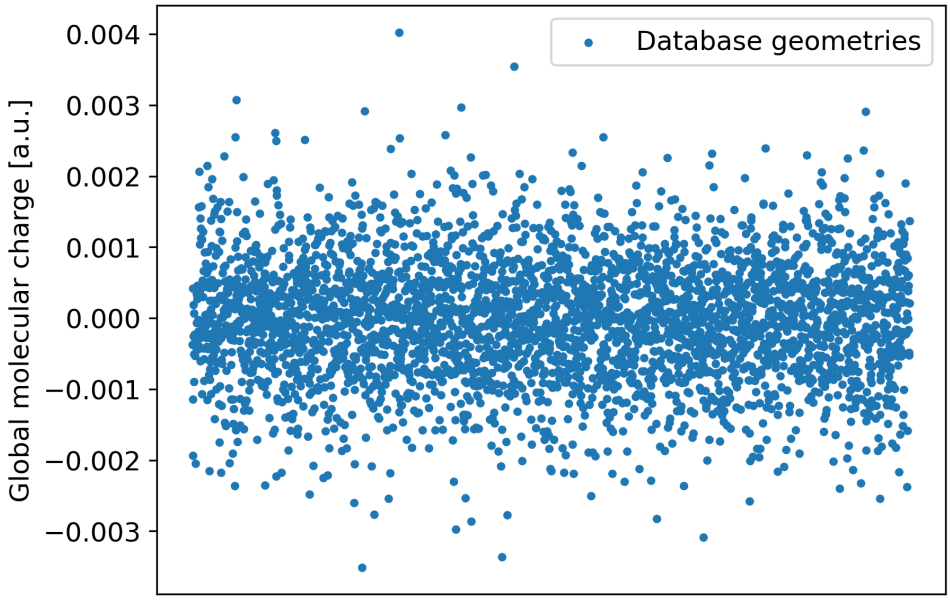


Figure 9.5: Global molecular charges obtained with QTAIM for all the geometries in the database.

Amino acid	nº atoms	nº geometries
Alanine	13	260
Arginine	26	1040
Aspargine	17	680
Aspartic acid	16	960
Glutamic acid	19	1140
Glutamine	20	800
Glycine	10	200
Histidine	20	1200
Isoleucine	22	440
Leucine	22	440
Lysine	24	480
Phenylalanine	23	460
Proline	17	340
Serine	14	560
Threonine	17	680
Tryptophan	27	540
Tyrosine	24	960
Valine	19	380
TOTAL:	350	10720

Table 9.2: Number of atoms and geometries generated for each amino acid to form the boosted database for the ACSFs generation.

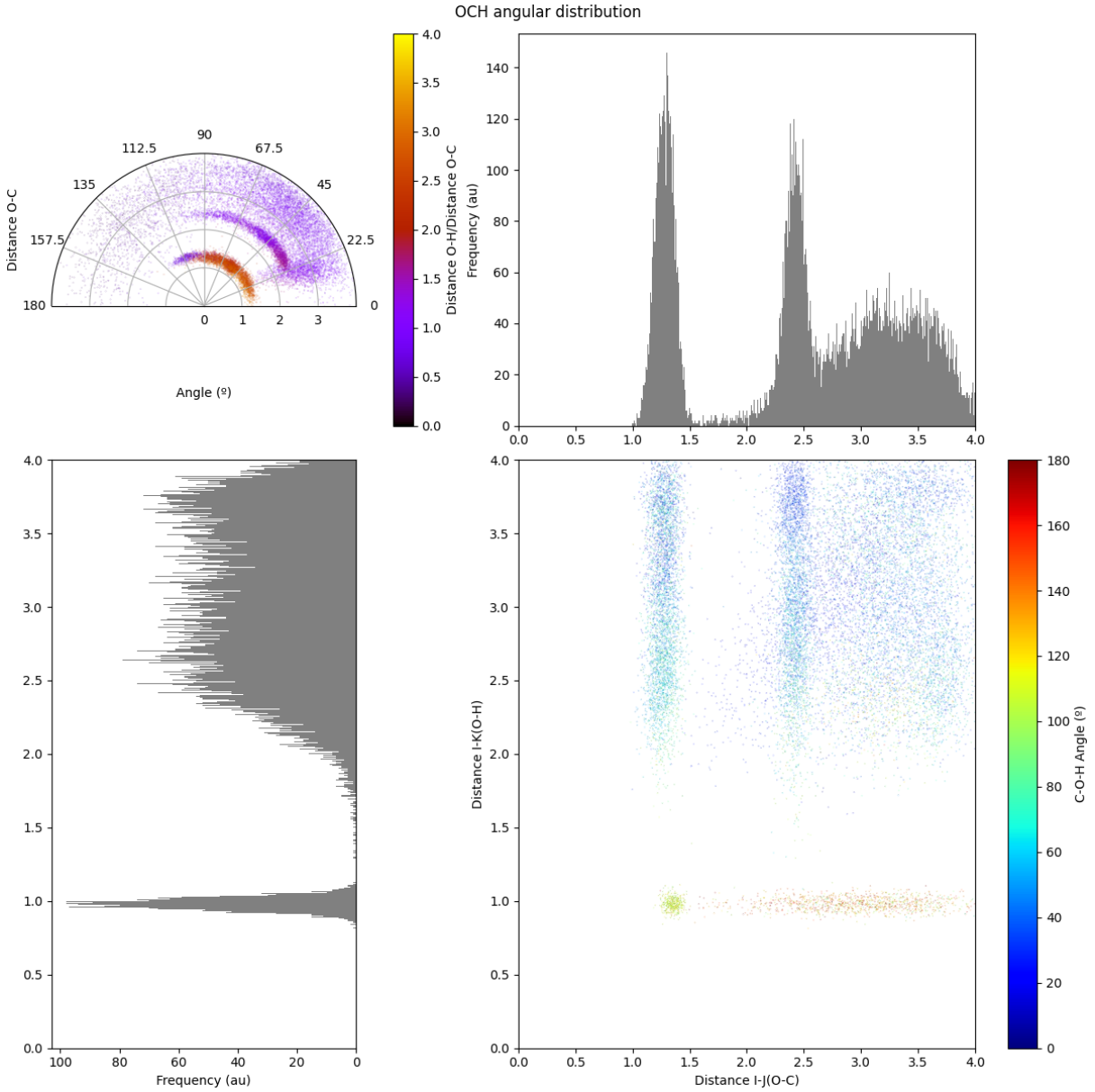


Figure 9.6: GMM reconstructed angular distribution of the O-CH atomic trio for the same database used for the training. The radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

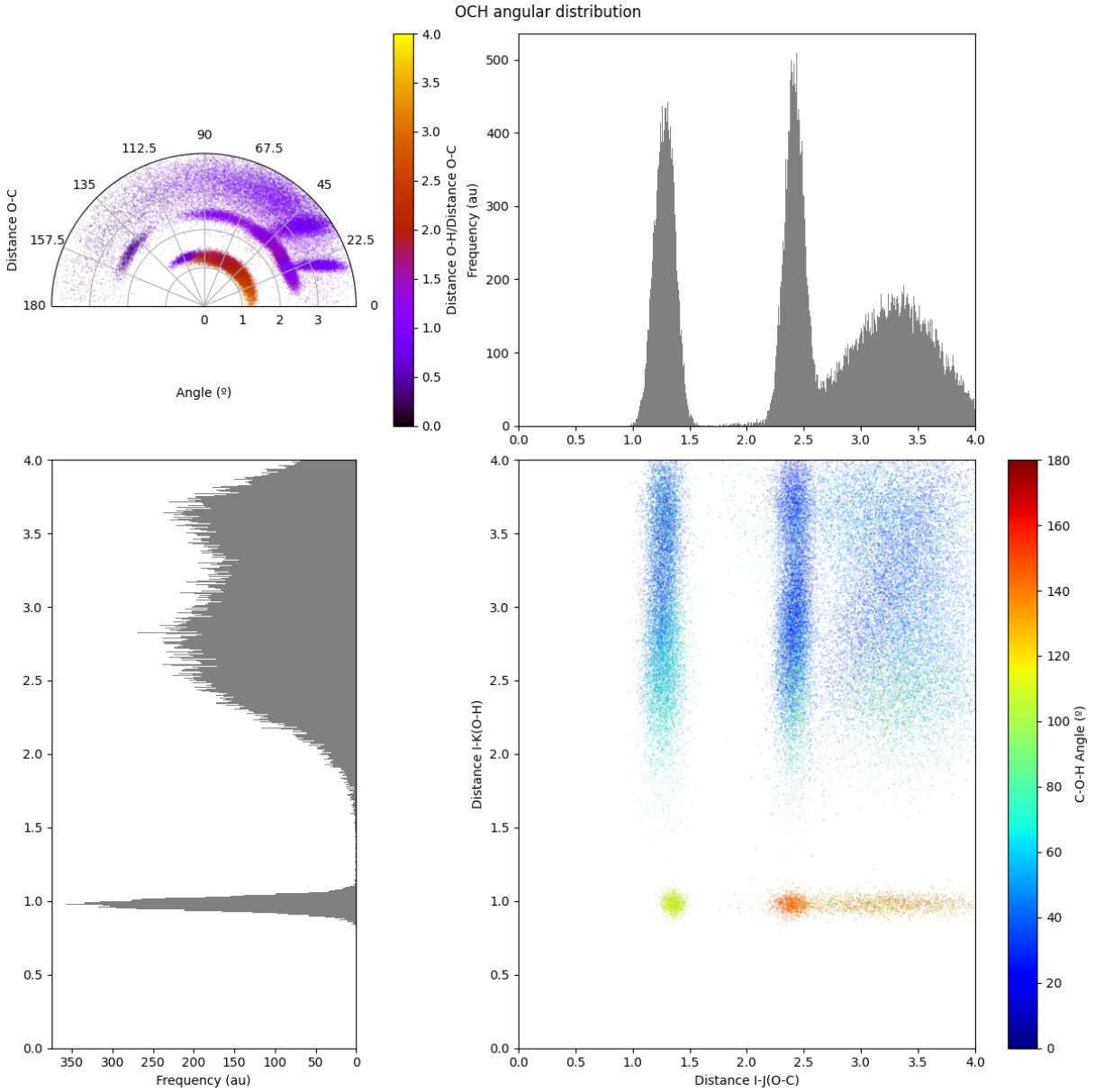


Figure 9.7: GMM reconstructed angular distribution of the O-CH atomic trio for the final boosted database. The radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

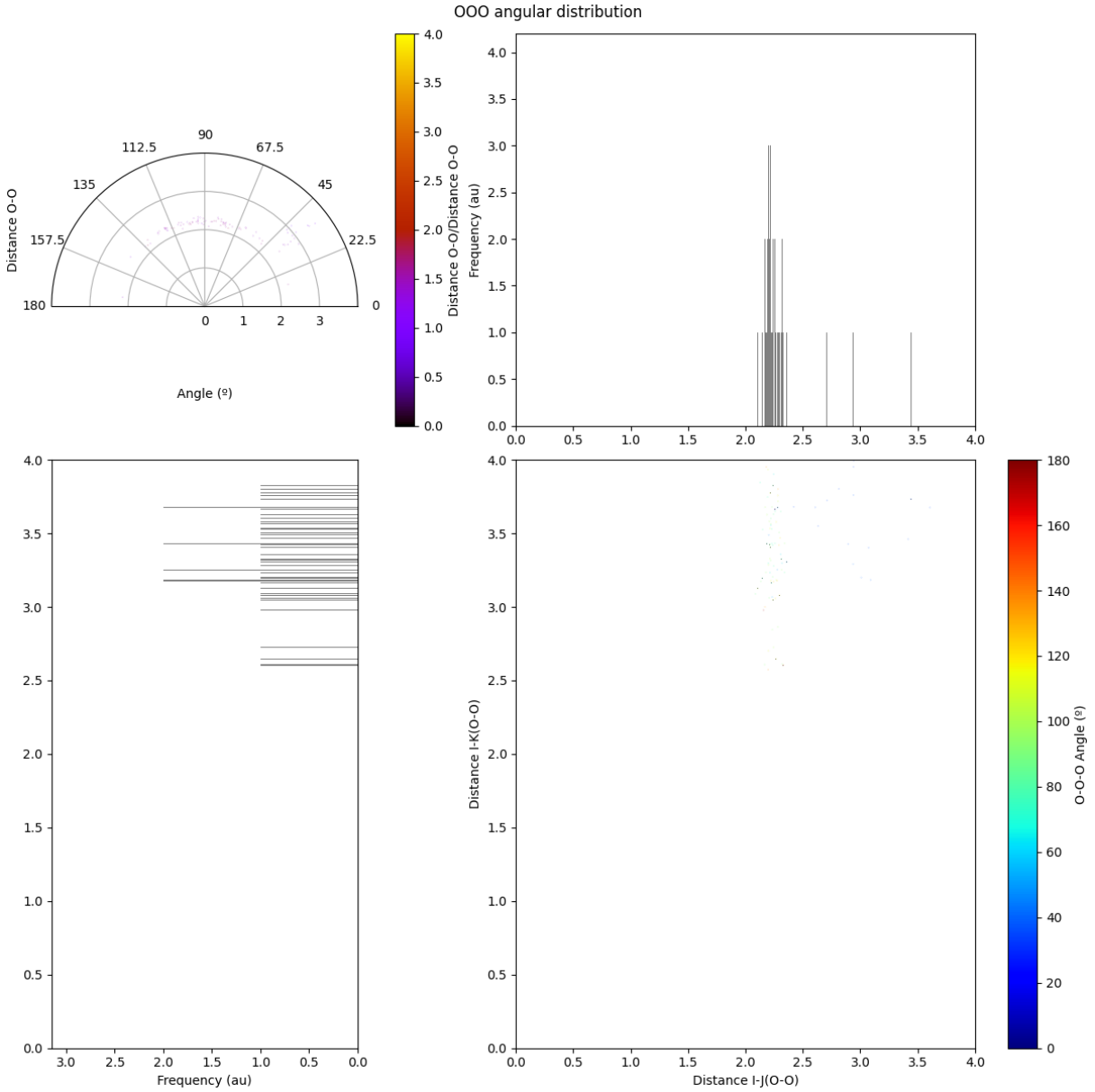


Figure 9.8: GMM reconstructed angular distribution of the O-CH atomic trio for the same database used for the training. The radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

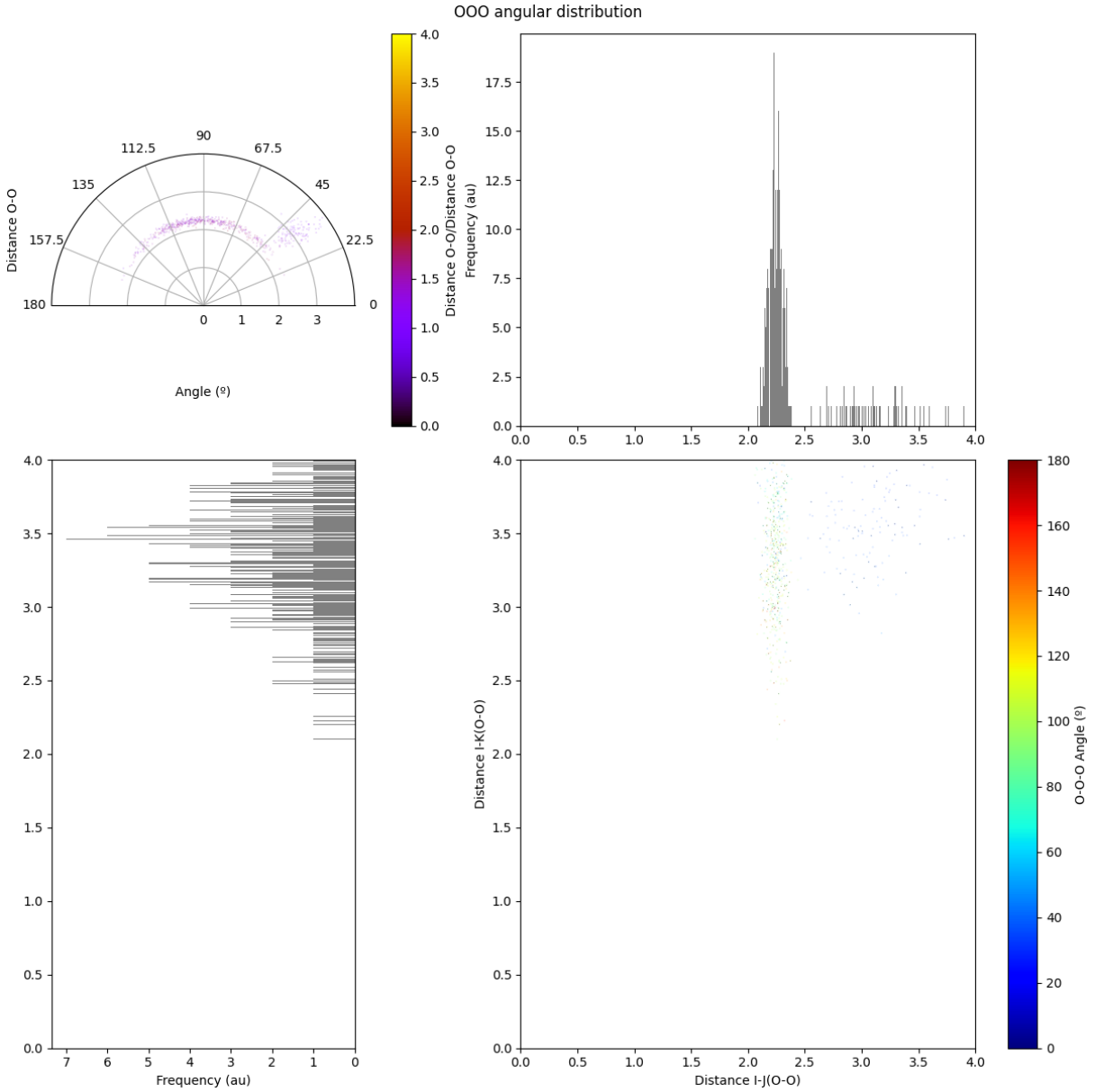


Figure 9.9: GMM reconstructed angular distribution of the O-OO atomic trio for the final boosted database. The radial dispersion (bottom right) plot is colored according to the θ_{ijk} values, whereas the polar plot (top left) is colored according to the r_{ik}/r_{ij} ratio to show the heterogeneity in neighboring pairwise distances.

Bibliography

[1] M. Castells, *City*, 1997, **2**(7), 6–16.

[2] A. G. Gross and J. E. Harmon, *The Internet revolution in the sciences and humanities*, Oxford University Press, 2016.

[3] M. Islam, G. Chen, and S. Jin, *American Journal of Neural Networks and Applications*, 2019, **5**(1), 7–11.

[4] B. Mahesh, *International Journal of Science and Research (IJSR).[Internet]*, 2020, **9**(1), 381–386.

[5] B. G. Sumpter and D. W. Noid, *Macromolecular theory and simulations*, 1994, **3**(2), 363–378.

[6] M. Grifoni and P. Hänggi, *Physics Reports*, 1998, **304**(5-6), 229–354.

[7] S. Fujita and S. Godoy, *Quantum statistical theory of superconductivity*, Springer Science & Business Media, 1996.

[8] D. Canet and D. Canet, *Nuclear magnetic resonance: concepts and methods*, Wiley Chichester, 1996.

[9] R. Wong and W.-L. Chang, *Journal of Parallel and Distributed Computing*, 2022, **164**, 178–190.

[10] D. Vollath, *Environmental Engineering and Management Journal*, 2008, **7**(6), 865–870.

[11] A. Khodavirdipour, M. Mehregan, A. Rajabi, and Y. Shiri, *Avicenna Journal of Clinical Microbiology and Infection*, 2019, **6**(4), 133–137.

[12] S. H. Park and K. Han, *Radiology*, 2018, **286**(3), 800–809.

[13] G. Luo, J. Long, B. Zhang, C. Liu, S. Ji, J. Xu, X. Yu, and Q. Ni, *Expert opinion on drug delivery*, 2012, **9**(1), 47–58.

[14] W.-Q. Deng and W. A. Goddard, *The Journal of Physical Chemistry B*, 2004, **108**(25), 8614–8621.

[15] A. Steane, *Reports on Progress in Physics*, 1998, **61**(2), 117.

[16] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, *Reviews of Modern Physics*, 2020, **92**(1), 015003.

[17] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya, et al., *Chemical reviews*, 2019, **119**(19), 10856–10915.

[18] W. Thiel, *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2014, **4**(2), 145–157.

[19] G. B. Goh, N. O. Hodas, and A. Vishnu, *Journal of computational chemistry*, 2017, **38**(16), 1291–1307.

[20] M. Gallegos and Á. Martín Pendás, *Journal of Chemical Information and Modeling*, 2023, **63**(13), 4100–4114.

[21] M. Gallegos, J. M. Guevara-Vela, and Á. M. Pendás, *The Journal of Chemical Physics*, 2022, **156**(1).

[22] P. Ehrenfest, *Annalen der Physik*, 1911, **341**(11), 91–118.

[23] M. Rodríguez-Meza and J. L. Cervantes-Cota, *CIENCIA ergo-sum, Revista Científica Multidisciplinaria de Prospectiva*, 2006, **13**(3), 303–311.

[24] L. De Broglie, *Nobel lecture*, 1929, **12**, 244–256.

[25] A. D. Vlasov, *Physics-Uspekhi*, 1993, **36**(2), 94.

[26] J. Mehra, *The Solvay conferences on physics: aspects of the development of physics since 1911*, Springer Science & Business Media, 2012.

[27] F. A. Berezin and M. Shubin, *The Schrödinger Equation*, Vol. 66, Springer Science & Business Media, 2012.

[28] E. G. Lewars, *Introduction to the theory and applications of molecular and quantum mechanics*, 2011, **318**.

[29] A. Szabo and N. S. Ostlund, *Modern quantum chemistry: introduction to advanced electronic structure theory*, Courier Corporation, 2012.

[30] R. Mulliken, *Reviews of Modern Physics*, 1960, **32**(2), 232.

[31] J. Townsend, J. K. Kirkland, and K. D. Vogiatzis in *Mathematical Physics in Theoretical Chemistry;* Elsevier, 2019; pp. 63–117.

[32] J. Binkley and J. Pople, *International Journal of Quantum Chemistry*, 1975, **9**(2), 229–236.

[33] I. Shavitt in *Methods of electronic structure theory;* Springer, 1977; pp. 189–275.

[34] R. J. Bartlett and M. Musiał, *Reviews of Modern Physics*, 2007, **79**(1), 291.

[35] E. Engel, *Density functional theory*, Springer, 2011.

[36] K. Burke, J. P. Perdew, and Y. Wang in *Electronic Density Functional Theory: recent progress and new directions;* Springer, 1998; pp. 81–111.

[37] J. P. Perdew, M. Ernzerhof, and K. Burke, *The Journal of chemical physics*, 1996, **105**(22), 9982–9985.

[38] B. Miehlich, A. Savin, H. Stoll, and H. Preuss, *Chemical Physics Letters*, 1989, **157**(3), 200–206.

[39] J. Tao, J. P. Perdew, V. N. Staroverov, and G. E. Scuseria, *Physical review letters*, 2003, **91**(14), 146401.

[40] Y. Zhao and D. G. Truhlar, *Theoretical chemistry accounts*, 2008, **120**, 215–241.

[41] F. Tran, J. Stelzl, and P. Blaha, *The Journal of chemical physics*, 2016, **144**(20).

[42] A. D. Becke, *The Journal of chemical physics*, 1997, **107**(20), 8554–8560.

[43] Y. Zhao and D. G. Truhlar, *Theoretical chemistry accounts*, 2008, **120**, 215–241.

[44] Á. M. Pendás and J. Contreras-García, *Topological Approaches to the Chemical Bond*, Springer Nature, 2023.

[45] C. F. Matta and R. J. Boyd, *The quantum theory of atoms in molecules: from solid state to DNA and drug design*, 2007.

[46] S. Jenkins, Z. Liu, and S. R. Kirk, *Molecular Physics*, 2013, **111**(20), 3104–3116.

[47] C. L. Firme, O. A. C. Antunes, and P. M. Esteves, *Chemical Physics Letters*, 2009, **468**(4-6), 129–133.

[48] C. F. Matta, *Journal of Computational Chemistry*, 2014, **35**(16), 1165–1198.

[49] F. Fuster and S. J. Grabowski, *The Journal of Physical Chemistry A*, 2011, **115**(35), 10078–10086.

[50] E. C. Brown, R. F. Bader, and N. H. Werstiuk, *The Journal of Physical Chemistry A*, 2009, **113**(13), 3254–3265.

[51] A. I. Baranov and M. Kohout, *Journal of computational chemistry*, 2011, **32**(10), 2064–2076.

[52] A. M. Turing, *Computing machinery and intelligence*, Springer, 2009.

[53] D. A. Reynolds et al., *Encyclopedia of biometrics*, 2009, **741**(659-663).

[54] S. Wold, K. Esbensen, and P. Geladi, *Chemometrics and intelligent laboratory systems*, 1987, **2**(1-3), 37–52.

[55] G. Cybenko, *Mathematics of control, signals and systems*, 1989, **2**(4), 303–314.

[56] K. Hornik, *Neural networks*, 1991, **4**(2), 251–257.

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[58] J. Behler, *The Journal of chemical physics*, 2011, **134**(7).

[59] T. Nagy, A. Vikár, and G. Lendvay, *The Journal of Chemical Physics*, 2016, **144**(1).

[60] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, *Journal of cheminformatics*, 2011, **3**, 1–14.

[61] M. e. Frisch, G. Trucks, H. B. Schlegel, G. Scuseria, M. Robb, J. Cheeseman, G. Scalmani, V. Barone, G. Petersson, H. Nakatsuji, et al., Gaussian 16, 2016.

[62] M. Karplus and G. A. Petsko, *Nature*, 1990, **347**(6294), 631–639.

[63] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, et al., *Journal of Physics: Condensed Matter*, 2017, **29**(27), 273002.

[64] S. Grimme, C. Bannwarth, and P. Shushkov, *Journal of chemical theory and computation*, 2017, **13**(5), 1989–2009.

[65] T. A. Keith, *TK Gristmill Software: Overland Park, KS, USA*, 2017.

[66] P. GNU, Free software foundation. bash (3.2. 48)[unix shell program], 2007.

[67] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA, 2009.

[68] M. Gallegos, B. K. Isamura, P. L. Popelier, and Á. Martín Pendás, *Journal of Chemical Information and Modeling*, 2024, **64**(8), 3059–3079.

[69] M. J. Lopez and S. S. Mohiuddin in *StatPearls [Internet];* StatPearls Publishing, 2023.