



Universidad de Valladolid

Facultad de Ciencias

TRABAJO FIN DE MASTER

European Master in Theoretical Chemistry and
Computational Modelling

HIGH DIMENSIONAL NEURAL NETWORK POTENTIALS: SOFTWARE DEVELOPMENT AND APPLICATION TO NITROGEN CENTERED RADICALS

Author:

Javier Domínguez Calvo

Advisors:

Victor Rayón Rico

Matti Hellström

Acknowledgements

[Marvin the Paranoid Android, an Artificial Intelligence, said:]

“I am at a rough estimate thirty billion times more intelligent than you. Let me give you an example. Think of a number, any number.”

“Er, five,” said the mattress.

“Wrong,” said Marvin. “You see?”

Douglas Adams

Life, the Universe and Everything

I want to thank my family and friends for being my support during these Master Studies, especially given the difficult situation of these years due to the pandemic. So thank you all: Carmen, Nacho, Mario, Clara, Carlos, David, Bego and more.

Also, I want to thank the whole SCM team for their help during my internship there and for the opportunity to work in a different environment and develop code, all of this in a different country. In particular to Matti Hellström, my advisor during the internship who has taught me a lot.

Finally, I have to acknowledge Victor Rayon, my advisor at the University of Valladolid, for having the patience for allowing me to program and automatize the code in the way I wanted to, and being very helpful every time I need it.

Contents

List of Figures	v
List of Tables	vi
Acronyms	vii
I Density Functional Theory for Radical Stabilization Energies of Nitrogen Centered Radicals: Automatization of the Process and Comparison with Published Data	1
1 Introduction	4
1.1 Density Functional Theory	4
1.1.1 Origins of DFT	4
1.1.2 DFT without dispersion	5
1.1.3 DFT with dispersion	6
1.2 Basis set	7
1.3 Resolution of the Identity	8
1.4 Molecules	8
2 Goals	10
2.1 Languages	10
2.1.1 Python	10

2.1.2	mySQL	11
2.1.3	Shell	11
2.1.4	Compatibility between languages	12
2.2	Obtaining RSE	12
3	Results	14
4	Conclusions	19
II	High Dimensional Neural Network Potentials: Description, Development within the SCM Code and an Example of Application	20
1	Introduction	23
1.1	Machine Learning	23
1.1.1	Regression	24
1.2	Neural networks	26
1.3	Neural Networks for chemistry	29
1.4	High Dimensional Neural Network Potentials	31
1.4.1	Output	32
1.5	Input	33
1.5.1	Invarational Input	33
1.5.2	Symmetry Functions	34
1.5.3	Forces	37
1.6	Energy and Force Calculation	37
2	Goals	40
3	Results	44

3.1 Training of the model	44
3.2 Testing the model	46
4 Conclusions	49
A Molecules	50
B Python	52
C MySQL Database	59
D Bash	61
Index	65
Bibliography	68

List of Figures

3.1	RSE	16
3.2	MUE	16
1.1	A simple neural network	27
3.1	Training of the Networks	46
3.2	HDNNP vs DFT	47
C.1	The MySQL diagram	59

List of Tables

3.1	DFT calculations results. All values are in Eh.	17
3.1	Number of Symmetry Functions	45
3.2	HDNNP results	47
A.1	Molecules used in this thesis. Only the radicals are shown, but the given name is that of the non radical species.	50

Acronyms

G Symmetry Functions. 34

OOP Object Oriented Programming. 40

AI Artificial Intelligence. 10, 23

AMS Amsterdam Modelling Suite. 40–42, 45

B3LYP Becke, 3-parameter, Lee–Yang–Parr. 5

BDE Bond Dissociation Energy. 9

DFTB Density Functional based Tight Binding. 44

DFT Density Functional Theory. 4, 30

DL Deep Learning. 26

DS Data Science. 40

GGA Generalized Gradient Approximation. 6

HDNNP High Dimensional Neural Network Potentials. 23, 30, 32, 40, 41, 44

HF Hartree-Fock. 30

M06-2X Minnesota 2006 density functional with 54% Hartree-Fock exchange. 5

M06-L Minnesota 2006 local functional. 5

ML Machine Learning. 23, 24, 26, 30, 40

MP2 Møller–Plesset. 7

MUAE Mean Unsigned Average Error. 14

MUE Mean Unsigned Error. 14

NN Neural Networks. 26–28, 30, 31, 41

PES Potential Energy Surface. 30, 32, 48

RI-JCOSX chain-of-spheres exchange. 8

RI Resolution of the Identity. 8

RSE Radical Stabilization Energy. 8, 9, 14

RS Range Separated. 7

SCM Software for Chemistry and Materials. 40

SQL Structured Query Language. 11

About this work

This Master Thesis was originally going to be done locally in the University of Valladolid. However, I had the opportunity of having an internship at the SCM, in Amsterdam, the Netherlands, developing high Dimensional Neural Network Potentials. So I spent there 3 months doing this internship. As a result, the Master Thesis is divided in two parts.

Part 1 comprises everything done in Valladolid. The goal was to compare different DFT methods for some molecules extracted from a publication. But the main difficulty and what took the most time was to develop different scripts and software to automatize the process: from creating inputs to extracting the results.

Part 2 is the biggest part of the work. It tries to reflect what was done during my stay in Amsterdam. However, the goal there was to develop HDNNP in Fortran 2003 inside the code of the company. Due to license issues the big length of the code it would be useless to describe the code itself. So, what is done here is to describe these potentials and some little calculations were done to test them.

Unfortunately the biggest and hardest part of thesis, writing the code, is not well reflected in this thesis. So, please, keep in mind while reading, that the results are just there to illustrate the mechanics and test the code, but they are not the goal of this thesis.

This work has been drafted using Overleaf, in the \LaTeX language.

Part I

Density Functional Theory for Radical Stabilization Energies of Nitrogen Centered Radicals: Automatization of the Process and Comparison with Published Data

Abstract

In this part we try to study different DFT functionals and comparing with some reference data for some selected molecules. For that purpose, we are describing DFT, and different forms that exist of this theory and everything needed for the calculation. Then, the hardest task of this part was to build scripts that allow to automatize the whole process, and building a clever and relational database to store all relevant information. Finally, some calculations were performed using those scripts and where we compare several DFT functionals with published information.

1 Introduction

Due to the high number of calculations that can be done in a regular physical chemistry research, the need for some degree of automatization in calculation arises. In this part of the work, we tried to develop some way to automatize several DFT calculations, for several different functionals and in many different nitrogen radical compounds. In order to achieve that, several computational languages were used.

1.1 Density Functional Theory

Obtaining electronic energies for molecules is one of the main focus of computational chemistry. In order to achieve that, many different methods exist, such as Hartree-Fock (HF), post-Hartree-Fock, One of the most extended groups of calculations to perform these kinds of calculations is called Density Functional Theory (DFT). In these theory the properties of a many-electron system can be determined by using functionals, i.e. functions of another function. There are many different types of DFT functionals, and in the following lines we are describing some of the most modern and relevant ones.

1.1.1 Origins of DFT

In DFT, the most important variable is the electron density, which is related with the wave function with the following relation:

$$n(\mathbf{r}) = N \int d^3\mathbf{r}_2 \cdots \int d^3\mathbf{r}_N \Psi^*(\mathbf{r}, \mathbf{r}_2, \dots, \mathbf{r}_N) \Psi(\mathbf{r}, \mathbf{r}_2, \dots, \mathbf{r}_N) \quad (1.1)$$

As it can be derived, we can use this functional to obtain the effective single-particle potential as:

$$V_s(\mathbf{r}) = V(\mathbf{r}) + \int \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d^3\mathbf{r}' + V_{XC}[n(\mathbf{r})] \quad (1.2)$$

Of that equation, the most tricky component is the $+V_{XC}$, this is, the Exchange-Correlation potential, because it includes the many particle interactions, and the exact formula is not known, as we have to introduce approximations.

1.1.2 DFT without dispersion

There are many ways of calculating this potential. We are not going to describe the primitive and simple ways, but focusing on the important for the current work.

Hybrid Functionals

In Hybrid Functionals part of the exact exchange is given from Hartree-Fock Theory, while the rest of the exchange- correlation energy is obtained from other ways, usually empirical.

Of these functionals, one of the most widely known is Becke, 3-parameter, Lee-Yang-Parr (B3LYP) [1],

Minnesota Functionals

Very related to the former are the Minnesota Functionals known as M06. These functionals are constructed by empirically fitting their parameters, while being constrained to a uniform electron gas.

Two M06 functionals [2] are used in this work: Minnesota 2006 local functional (M06-L) and Minnesota 2006 density functional with 54% Hartree-Fock exchange (M06-2X). The former uses no HF exchange, so it is not really hybrid, and the latter uses a 54% of HF

exchange, as the name says.

1.1.3 DFT with dispersion

All of the former functionals do not include dispersion in their calculations. Dispersion (van der Waals) interaction plays a crucial role in the formation, stability, and function of molecules and materials. However, long-range correlation, which is the physical root of dispersion interaction, is absent in popular local or semi-local exchange-correlation functionals. Thus, proper dispersion corrections are necessary for DFT calculations on realistic systems.

The actual details of how this is included is not important for the current purpose, but many functionals can add this dispersion by adding one energy term. The most widely known dispersion functionals include DFT-D3 and DFT-D3BJ [3], where DFT is the chosen functional. Some functionals include by default this term and in others we have to specify it. In all the rest of functionals, that are going to be described in the following lines, this term was included or already appears. For instance, we also use B3LYP D3BJ, this means, the already described one with dispersion.

Generalized Gradient Approximation

The first of the subcategories to be described is Generalized Gradient Approximation (GGA) are a quite simple form of DFT. They are defined for using the gradient of the electron density as part of their calculations.

Two examples of these functionals used in the current work are PBE D3BJ [4] and BLYP D3BJ .

meta-GGA

A meta-GGA [5]functional uses the Laplacian (second derivative) of the electron density in addition to the density and the magnitude of the gradient of the density. Of all the possibilities, we chose B97M-D3BJ [6] and SCAN-D3BJ [7] (referred in this work as

SCANfunc-D3BJ, since it's the Orca input) to represent these functionals

Range Separated

Range Separated (RS) functionals vary the percentage of Hartree-Fock and DFT exchange for long-range and short-range interactions and are used in cases involving charge transfer excitation[8]. We wanted to include some functionals of these types, of the family of ω B97: wB97M-D3BJ [9] and wB97X-D3BJ [10].

Double Hybrid

Finally, the last category of functional to use are Double Hybrid. These functionals are developed by mixing a part of second-order Møller–Plesset (MP2) correlation energy to the exchange correlation potential[11]. Of these, we have included DSD-PBEP86 D3BJ[12] and B2PLYP-D3 [13]. In order to make this last functional work, it needed a correlation basis for the only case of the B2PLYP functional, so we had to include the keyword Def2-TZVPP/C in that case.

Many different functionals have been listed, as we need many different functionals to create an appropriate benchmark.

1.2 Basis set

Apart from choosing a functional, we must choose a proper basis set to describe the calculation. Quantum chemical calculations are performed using a finite set of basis functions, that must be chosen prior to the calculation. For simplicity we are only going to use one in the current work, although it would be possible to change it with the developed code. This is def2-TZVPP[14]. It is a triple zeta basis with valence polarization. It's a relatively good basis, so we should expect good results with it.

1.3 Resolution of the Identity

Resolution of the Identity (RI) [15]. Within the RI formulation, the electron density is approximated by a linear combination of auxiliary basis functions.

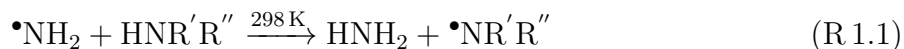
This basis should be in accordance with the basis set, so we are going to use def2. We use a basis only for the Coulomb (J) part, pr for both the Coulomb and the Exchange (JK). Most of the time with just the Coulomb is enough (def2/J). But we chose to use the def2/JK in one case: when dealing with hybrid functionals in molecules with less than 20 atoms. As we see, this require some degree of programming in order to choose the correct RI, one of the goals we want to achieve.

We also need to choose a function for the RI itself. DFT calculations that do not require the HF exchange to be calculated (non-hybrid DFT) can be very efficiently executed with the RI-J approximation.

In the case of the hybrid ones, due to slower calculations different approaches need to be done. For instance an algorithm that is called the chain-of-spheres exchange (RI-JCOSX) [16], that includes a semi-numerical integration. That for the Coulomb, but when we have to include the exchange as well we have to use RI-JK, as the former cannot do the former calculation.

1.4 Molecules

In this work we have chosen to work with the molecules from the article of Zipse [17]. Here he studies many nitrogen radical molecules. He defines the Radical Stabilization Energy (RSE) as in reaction (R 1.1).



Or mathematically in equation (1.3):

$$\begin{aligned} \text{RSE}(\bullet\text{NR}'\text{R}'') &= \text{BDE}(\text{HNR}'\text{R}'') - \text{BDE}(\text{H}\text{N}\text{H}_2) = \\ & (E(\bullet\text{NR}'\text{R}'') - E(\text{HNR}'\text{R}'')) - (E(\bullet\text{N}\text{H}_2) - E(\text{H}\text{N}\text{H}_2)) \end{aligned} \quad (1.3)$$

Here we introduce the Bond Dissociation Energy (BDE), this is the difference between the full molecule and the radical and hydrogen separately; the hydrogen is going to be canceled with itself obtaining the RSE, so we do not need to obtain it.

Of all the molecules described in the paper, we are going to focus only on the group A, which appear in [appendix A on page 50](#). This paper makes its calculations with other functional: G3B3. The goal will be to obtain different results with the different functionals and compare with the paper one

2 Goals

The goal of the experimental part would be easy; just to obtain the RSE of the different described radicals and comparing between methods and discussing the results obtained.

However, the main difficulty of this part of the thesis has been to develop computational methods to automatize the generation of input, extracting results and storing all data in an efficient way.

2.1 Languages

In order to achieve so, three different programming languages have been used.

2.1.1 Python

Python is one of the most broadly known computer languages and it has many uses. In particular, it has many applications in computer science, including Artificial Intelligence and data science. It's probably the most suited language for making short calculations and analyzing results. To use Python for data science, one should import the following 3 packages:

- Numpy: For algebra and mathematical operations
- Pandas: For table usage
- Matplotlib: For plots and graphics

In the current work, Python has been used for preparing the data prior to store it, as well as to analyze and represent the results. Some of the python files developed by the student appear in [appendix B on page 52](#)

The main drawback of Python is that for complex operations it is very slow. But is a very general tool that has many uses for simple problems.

2.1.2 *mySQL*

But Python lacks the ability to store efficiently big pieces of data, unless a csv file or similar is created each time some data is modified. To overcome that weakness we have used MySQL, one language for relational databases. In relational databases millions of pieces of data can be stored in form of tables, where each row has a unique id. We can build tables for the functionals, molecules or calculations, storing all important information in different fields of the rows and then calling several tables at the same time to merge the contents.

MySQL is just one of the many Structured Query Language (SQL) variants that exist. They all have in common that they are very useful for handling relational and structured data.

In [appendix C on page 59](#) appears the scheme of the database used in this piece of work. As you can see there appear many different tables related by many fields.

2.1.3 *Shell*

Last, we have used Shell, in particular Bash for building commands inside the cluster to launch the calculations. It takes the data from the mySQL and builds the input files, launch every calculation in an ordered way. Besides, when the calculations are done, we use bash for extracting the energies and any other information if needed from the output files in an automated way, and for storing the labeled data inside the results mySQL table.

This part could theoretically be done with python, but for file handling, folder creation and keyword finding, Shell is the most useful tool. Unfortunately, this language is not useful

at all for mathematical operations, even a single real number multiplication is hard to get, so that is why we use Python for numerical analysis.

2.1.4 *Compatibility between languages*

In order to be able to use those languages they have to be able to communicate with each other.

MySQL has its own syntax to get results from its tables, for instance the following query:

```
SELECT name FROM molecule WHERE atoms < 20 AND charge = 0
```

Would return the name of all the neutral molecules with less than 20 atoms. We can call this from a bash script (Shell) using the command :

```
mysql -u USER -pPASSWORD -D DATABASE -N -B -e  
"SELECT name FROM molecule WHERE atoms < \${MAXATOMS} AND charge = 0"
```

So we can call the query including some variables.

For python we have to use the mysql-connector package, and it allows to get MySQL tables as Pandas dataframes and then use a lot of existing code to modify the tables, add columns or plot the results for instance.

The goal is to have everything important stored in MySQL and then access to it by either side of it.

2.2 Obtaining RSE

Once we have such procedures created, launch and obtaining results is the easy part. In fact the databases are ready to work with a lot more molecules and functionals and basis; we would just have to define them with Python for instance. But in order to test it we can just compare the results of some molecules, the set of A molecules.

We will obtain the RSE and compare with the paper values, and describe which functionals are better in which cases.

3 Results

Apart from the coding itself, that can be found in the various appendix, we will discuss in the following pages some results encountered with the calculation. There is no actual need to describe a lot of molecules, since we are just interested to obtain differences between calculation methods. We have chosen the subset of molecules found in [A on page 50](#).

So, we have to launch the calculations for every functional and every molecule, either radical or not. This way we can obtain the RSE. All calculations were made with Orca[18, 19].

In order to obtain the goodness of each calculation we have obtained the Mean Unsigned Error (MUE) , this is, the difference (in absolute value) between our result and the paper one using G3B3.

Finally, we get the Mean Unsigned Average Error (MUAE) obtained from the average between all molecules. This is the value used to compare results.

We can find all results in [table 3.1 on page 17](#) and in [figures 3.1 to 3.2 on page 16](#).

In [table 3.1a](#) we find the functionals without dispersion, pure and hybrids. And we include B3LYP D3BJ for comparison purposes. Indeed, dispersion seems to improve the MUAE in about 1.7 kJ/mol for this functional approximation. Thus, taking into account dispersion is important for the calculation of RSEs. M06-2X is clearly the best performer. It is worth pointing out that M06-2X has been also parametrized to take into account non-covalent interactions.

Then, in [table 3.1b on page 17](#) we have pure functionals with dispersion. We can observe that B97M and SCAN are the best performers but still have large MUAE, For the other

two, we can see that the MUAE is even higher than without dispersion, so they are not describing well the molecules.

Finally, in table 3.1c we have hybrid and double hybrid functionals. We can observe a much better performance. Range separated hybrids perform better than B3LYP and double hybrids perform slightly better than range-separated.

So, in conclusion, for this set of molecules, the best performers are double hybrids (plus dispersion) and M06-2X.

In the figures we can easily find the same results, we find the G3B3 in black in all the figures, and it has one of the highest energy values. The double hybrid and M06-2X are the only functionals that seem to be close to the black line; in some cases, for instance B3LYP, the difference is pretty high.

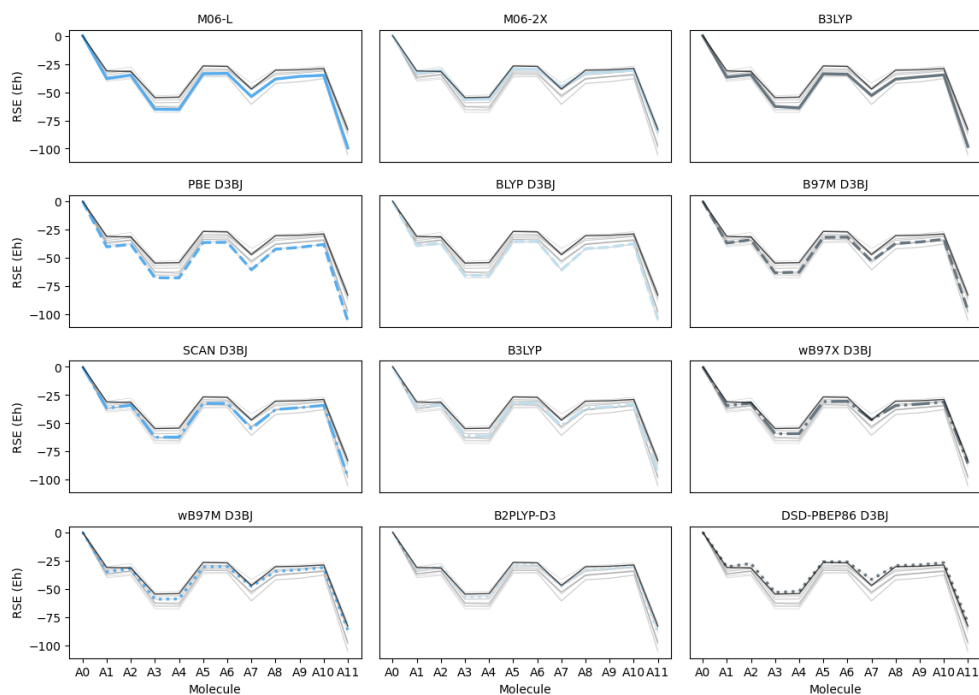


Figure 3.1: RSE of all functionals. In every plot, the G3B3 is given as a full black line, and the rest appear in light grey. The named functional in each graph is colored.

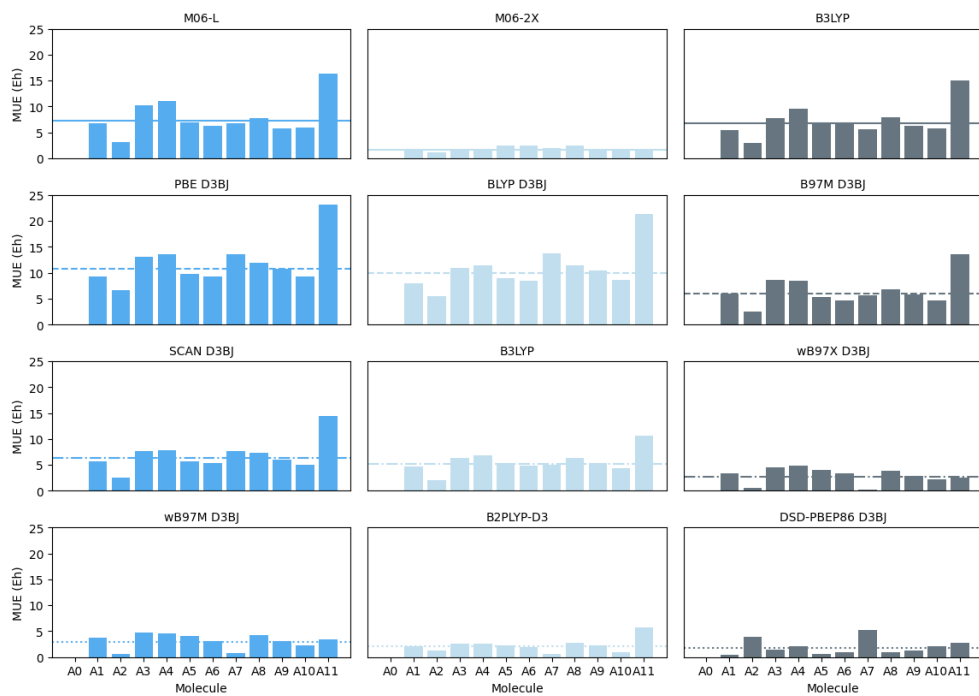


Figure 3.2: MUE between each functional and the G3B3. The horizontal line is the average value.

Table 3.1: DFT calculations results. All values are in Eh.

(a) Pure and hybrid without dispersion. B3LYP D3BJ for comparison.

	G3B3	M06-L		M06-2X		B3LYP		B3LYP D3BJ	
		pure		hybrid		hybrid		hybrid	
	RSE	RSE	MUE	RSE	MUE	RSE	MUE	RSE	MUE
A0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A1	-31.2	-38.0	6.8	-32.9	1.7	-36.7	5.5	-35.9	4.7
A2	-31.7	-34.9	3.2	-30.6	1.1	-34.5	2.9	-33.6	2.0
A3	-54.8	-65.1	10.3	-56.4	1.6	-62.6	7.8	-61.2	6.4
A4	-54.4	-65.4	11.0	-56.0	1.6	-64.0	9.6	-61.3	6.9
A5	-26.8	-33.7	6.9	-29.2	2.4	-33.8	7.0	-32.2	5.4
A6	-27.2	-33.4	6.2	-29.7	2.5	-34.2	7.0	-32.2	4.9
A7	-47.2	-53.9	6.7	-45.3	1.9	-52.9	5.6	-52.2	5.0
A8	-30.6	-38.4	7.8	-33.1	2.5	-38.5	7.9	-37.0	6.4
A9	-30.2	-36.0	5.8	-31.9	1.7	-36.5	6.3	-35.6	5.4
A10	-29.0	-35.1	6.0	-30.7	1.7	-34.7	5.7	-33.4	4.4
A11	-83.1	-99.4	16.3	-84.7	1.6	-98.1	15.0	-93.7	10.6
MUAE			7.9		1.8		7.3		5.6

(b) Pure functionals with dispersion.

	G3B3	PBE D3BJ		BLYP D3BJ		B97M D3BJ		SCAN D3BJ	
		pure		pure		pure		pure	
	RSE	RSE	MUE	RSE	MUE	RSE	MUE	RSE	MUE
A0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A1	-31.2	-40.4	9.2	-39.1	7.9	-37.0	5.9	-36.8	5.7
A2	-31.7	-38.3	6.6	-37.2	5.5	-34.2	2.5	-34.1	2.5
A3	-54.8	-67.8	13.0	-65.7	10.9	-63.4	8.6	-62.5	7.7
A4	-54.4	-67.9	13.5	-66.0	11.5	-62.8	8.4	-62.3	7.9
A5	-26.8	-36.6	9.8	-35.7	8.9	-32.0	5.3	-32.5	5.7
A6	-27.2	-36.4	9.2	-35.6	8.4	-31.8	4.6	-32.5	5.3
A7	-47.2	-60.7	13.5	-60.9	13.7	-52.9	5.7	-54.9	7.7
A8	-30.6	-42.5	11.9	-42.0	11.4	-37.4	6.8	-37.9	7.3
A9	-30.2	-40.9	10.7	-40.7	10.5	-36.1	5.8	-36.3	6.1
A10	-29.0	-38.3	9.3	-37.7	8.6	-33.7	4.7	-34.2	5.1
A11	-83.1	-106.2	23.1	-104.5	21.4	-96.6	13.5	-97.5	14.4
MUAE			11.8		10.8		6.5		6.9

(c) Hybrids and double hybrids including dispersion.

	G3B3	B3LYP		wB97X D3BJ		wB97M D3BJ		B2PLYP-D3		DSD-PBEP86 D3BJ	
		hybrid		hybrid		hybrid		double-hybrid		double-hybrid	
	<u>RSE</u>	<u>RSE</u>	<u>MUE</u>	<u>RSE</u>	<u>MUE</u>	<u>RSE</u>	<u>MUE</u>	<u>RSE</u>	<u>MUE</u>	<u>RSE</u>	<u>MUE</u>
A0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A1	-31.2	-35.9	4.7	-34.6	3.4	-34.9	3.7	-33.3	2.1	-30.7	0.4
A2	-31.7	-33.6	2.0	-32.2	0.5	-32.3	0.6	-30.5	1.2	-27.7	3.9
A3	-54.8	-61.2	6.4	-59.4	4.6	-59.4	4.7	-57.3	2.6	-53.3	1.5
A4	-54.4	-61.3	6.9	-59.3	4.9	-59.1	4.6	-57.0	2.6	-52.4	2.1
A5	-26.8	-32.2	5.4	-30.9	4.1	-30.7	4.0	-29.1	2.3	-26.2	0.6
A6	-27.2	-32.2	4.9	-30.6	3.4	-30.3	3.1	-29.2	2.0	-26.3	0.9
A7	-47.2	-52.2	5.0	-47.0	0.2	-48.0	0.8	-47.8	0.6	-41.9	5.3
A8	-30.6	-37.0	6.4	-34.5	3.9	-34.8	4.2	-33.3	2.7	-29.6	1.0
A9	-30.2	-35.6	5.4	-33.1	2.9	-33.3	3.1	-32.4	2.2	-29.0	1.2
A10	-29	-33.4	4.4	-31.3	2.3	-31.3	2.2	-30.1	1.0	-27.0	2.1
A11	-83.1	-93.7	10.6	-85.6	2.5	-86.5	3.4	-88.8	5.7	-80.4	2.7
MUAE			5.6		3.0		3.1		2.3		2.0

4 Conclusions

It has been achieved in this part the following:

- We have developed Python code to be able to operate numerically with computational chemistry problems.
- We have created a MySQL database to store all relevant information, as well as communicate itself to be able to respond to any conditions we want it to have.
- We have also developed Shell code that is able to create and launch working calculation inputs and to retrieve results
- We have achieved that all of those languages may communicate between them in a smooth way, and to get the information when needed
- We also have used all of that to get some calculations where we have proven that some functionals perform better than others

In the future, this work could be expanded by automatizing even more the process, and making it even more general, including different basis sets or calculation types.

Part II

High Dimensional Neural Network Potentials: Description, Development within the SCM Code and an Example of Application

Abstract

For this part the student developed software for building High Dimensional Neural Network Potentials in the language Fortran 2003 and the integration in the AMS code of the SCM company. In this work, these type of potentials are described from scratch, and compared to traditional linear regressions. Then, the goals that were fulfilled (and those that were not) are widely described. Instead of showing the code, finally some results obtained using the code developed by the student are shown and discussed, and compared with the DFT results from Part 1.

1 Introduction

Machine Learning (ML) is a branch of Artificial Intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. ML can be used for computational chemistry in many ways, being one of those High Dimensional Neural Network Potentials (HDNNP) [20–24]. They can be used to obtain molecular properties such as the energy, but in a complete different fashion than that of traditional physical chemistry calculations such as DFT or Ab Initio

1.1 Machine Learning

Machine Learning (ML) involves computers discovering how they can perform tasks without being explicitly programmed to do so. [25] It involves computers learning from data provided so that they carry out certain tasks.

In traditional programming we create a program, which after is feed by some input data, will get some results. ML can use a complete different approach. Here we feed the input and the output that results from the input, and the computer builds a program that generates such an output from the input. When we have the program, we can feed different input and obtain (allegedly good) output. This kind of Machine Learning is called Supervised Machine Learning, where we have the output it should be, as opposed to Unsupervised Machine Learning, where we don't know the result. This supervised ML is usually divided in two steps. First, training involves having input and the correspondent output to train the

program and testing where we obtain useful data of other input cases with the program. The bigger the first phase is, the better results we will have in the latter. The way of checking the goodness of the method is to save some of the data for the training and use it in the testing, this way we can analyze the difference between the real result and the obtained one. Notice also that the method can only be as good as the quality of the training data.

1.1.1 Regression

In the present case we are going to use as input the geometry of one atom, and as output the energy of that atom. Since the result is a quantitative result, the specific type of ML we are dealing with is called regression. If the result were categorical (such as identifying if a mail is spam or not), we would be talking about classification.

Simple Linear Regression

Regression is actually a widely known method, in particular linear regression.

Simple Linear Regression

Simple linear regressions are widely used in a lot of scientific problems and despite it's simplicity they can be very useful. The equation to be solved in this case is described in (1.1):

$$y = a \cdot x + b \tag{1.1}$$

In this case, we know some x and y values before hand, and with those we can obtain a and b , which, if they are trained well, can be used to obtain the corresponding y for a different x . As we can see there appear the two steps that defined Supervised machine Learning, training and testing.

Multiple Linear Regression

There are many other types of regression. We can use more than one x value, so we would have multiple linear regression, with an equation such as (1.2), for a total number of i x values. This way we make the regression in not one, but multiple input values that summed give one single output value, not forgetting b

$$y = a_1x_1 + a_2x_2 + \cdots + a_ix_i + b = \begin{pmatrix} a_1 & a_2 & \cdots & a_i \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \end{pmatrix} + b \quad (1.2)$$

Many Simple Linear Regressions

It can also be done something inspired on the above for multiple y and a single x . In this case we would have many simple linear regressions, a total of j , the same as y values, where we always input the same x , and the equations can be expressed separately or in matrix form (1.3)

$$\left. \begin{array}{l} y_1 = a_1x + b_1 \\ y_2 = a_2x + b_2 \\ \vdots \\ y_j = a_jx + b_j \end{array} \right\} \rightarrow \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_j \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_j \end{pmatrix} x + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \end{pmatrix} \quad (1.3)$$

Matrix Linear Regressions

Finally, we can combine the two methods above. This means, make a linear regression for any number of either input or output values, i and j respectively. The equation in this case would be the one described in equation (1.4)

$$\left. \begin{aligned}
 y_1 &= a_{11}x_1 + a_{12}x_2 + \dots + a_{1i}x_i + b_1 \\
 y_2 &= a_{21}x_1 + a_{22}x_2 + \dots + a_{2i}x_i + b_2 \\
 y_3 &= a_{31}x_1 + a_{32}x_2 + \dots + a_{3i}x_i + b_3 \\
 &\quad \vdots \\
 y_j &= a_{j1}x_1 + a_{j2}x_2 + \dots + a_{ji}x_i + b_j
 \end{aligned} \right\} \rightarrow \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_j \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1i} \\ a_{21} & a_{22} & \dots & a_{2i} \\ a_{31} & a_{32} & \dots & a_{3i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{j1} & a_{j2} & \dots & a_{ji} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_j \end{pmatrix}$$

$$\rightarrow \vec{y} = \mathbf{A} \vec{x} + \mathbf{b} \tag{1.4}$$

We have been able to obtain a very simple matrix equation, where both the inputs and the output are expressed as a vector. Notice that i and j can have different values and if one or both of them are equal to 1, we have the previous cases, that can be described by this equation as well. The matrix \mathbf{A} that contains all the slopes is going to be called in the current work the matrix of the weights and the individual values, the weights. On the other hand the components of \mathbf{b} are going to be called the bias. Even if this last part is a vector, we are going to be described in bold letters, to differentiate \mathbf{A} and \mathbf{b} from the input and output values.

1.2 Neural networks

Neural Networks (NN) are a subset of Machine Learning or even called Deep Learning (DL), because the machine learns with its own data processing, and not just the input and output values we give to train. Neural Networks are called like this, because theoretically they behave as neurons do, by making connections between them, although this field has expanded in a great way in the last decades, and this name is not as correct as it were in the beginnings.

The first thing to know about a NN is that they are made of layers. There is an input layer and an output layer, just as in regression we had input and output values. Besides

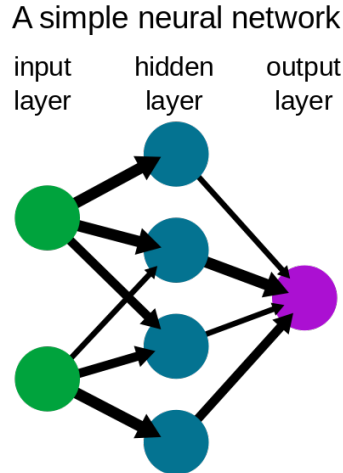


Figure 1.1: A simple neural network. Image obtained from [26]

that, we have in between a certain number of hidden layers . Each layer is made of a number of nodes . The number of nodes in each layer can be different.

Activation Function

Let's picture a 2 layer NN, made only of input and output. The nodes of the input and of the output have to be related by a mathematical expression, such as that of (1.4). But this is not everything. Probably we would not want the output in just this linear way, but we may want some sophistication. So we can apply what is called the activation function . It is applied as in equation (1.5):

$$\vec{y}' = f(\vec{y}) = f(\mathbf{A}\vec{x} + \mathbf{b}) \tag{1.5}$$

The actual nature of function f depends a lot in the case. We might want the result to have only two values, as in classification, so we could have $f(x) = 1$ if $y >$ some value or $f(x) = 0$ if not. For regression we might want some continuous and derivable function, such as $f(x) = \tanh(x)$, the hyperbolic tangent or $f(x) = \frac{1}{1+e^{-x}}$, the sigmoid; respectively to get values between -1 and 1 for the first and 0 and 1n for the latter. Or we can be not worried about this activation function and just use the linear, which is like not applying anything:

$f(x) = x$. The choice of the activation function depends on which value are we interested, so for energy it can be any value, so linear is fine, but for a probability we might want to use the sigmoid one.

But, what if we wanted to have one or more hidden layers? In this case the output in one layer becomes the input for the next one. This way we concatenate a series of weight and bias calculation with activation functions until we arrive to the end of the network. For the hidden layers, the activation function should be one like the hyperbolic tangent or sigmoid; we are not interested on the values to be very disperse since they are going to be inputted again. Besides if we used linear activation for the hidden layers, we would have two linear matrices, that always can be combined in only one, so the advantage on using several layers is lost

Number of Layers

The number of hidden layers to use is arguable. If we use none the result is just a linear regression, modified by the activation function to get the output in some way, but really the input can only be transformed with the output in a linear way. If we have, on the contrary one hidden layer, it is stated in the Universal Approximation Theorem [27, 28] that these NN can describe almost any continuous function. So, with one hidden layer we can get a numerical regression of any function.

What if we add more layers? Adding layers makes the calculation more computationally costly. But there are also great advantages on using more hidden layers [29]. Even if we can represent any function with one hidden layer, adding a second allows to create a layer with more complex data, in a higher level of abstraction. For instance, if the input are some pixels of a photography, the first layer can calculate some lines and edges, the next layer can figure out what is a nose or an eye, and the output would be if we have a face or not. Adding more layers can make it easier to recognize complex data, splitting into different functions, and the network can work more efficiently. Now we can talk about deep learning,

because the networks learns from itself; in the hidden layers all the information is created by the network itself from previous layers without human interaction.

We have to introduce here another crucial concept of machine learning that is overfitting. This occurs when we train too well a network, so the training data is very well explained by the network, but when we test it with the testing data we find that it does not actually give good regression in general, just for the data used to train. This can be a problem for instance if we use too many hidden layers. the number of those to use can depend, but usually one or two hidden layers are enough.

Number of nodes

Besides this, we have to look at the number of nodes. Usually the number of nodes in the input and the output nodes is given by the problem, but we can play with the hidden ones. There are some rules, but in practice it can only be determined by trial and error. To few nodes leads to underfitting, the result is not good enough to fit the training and too many can lead to overfitting again. So when building a neural network this is one of the things to take into account.

1.3 Neural Networks for chemistry

For now we have discussed what neural networks are for, and how they are made of, but now we can look how to apply them to chemistry. One of the oldest problems in physical chemistry is to obtain the energy of a molecule. In chemistry, this problem is solved with the Schrödinger equation, of shape:

$$\hat{H}\Psi = \left(\sum^N \hat{T}_N + \sum^e \hat{T}_e + \sum^N \sum^N V_{NN} + \sum^e \sum^e V_{ee} + \sum^N \sum^e V_{Ne} \right) \Psi = E\Psi \quad (1.6)$$

Some of those terms can be calculated easily, others require small approximations, and

others have not known shape and there are a lot of different methods to describe them: Density Functional Theory (DFT) , Hartree-Fock (HF) and post-HF, and more, with many different approaches in them.

But ML and Neural Networks method is completely different. We don't need to know the shape of any of those functions. We just need to know that the energy is going to be the result of applying some function to the geometry of the molecule, and that is the Schrödinger equation about. So we can build some neural network that does this regression, in this elaborate linear regression fashion that we have created.

So the input is going to be the geometry, and the output the energy. We see that the output layer is going to be made of just one node, because there is only one energy. But for the input we will have many numbers because we cannot describe the geometry with just one number.

Now, it's time to take a look on our problem. If we constructed a neural network for a molecule, it would be trained for that. If we accept that we can obtain in this fashion, the energy from the geometry, we could obtain different energy values from different geometries, and this could be useful for obtaining a Potential Energy Surface (PES) for instance, where we aim for different conformations of the same molecule. But if we changed the molecule, even the slightest, the network would be of no use, because the energy would be entirely different; having a lower number of atoms for instance, the energy to expect would be smaller in absolute value, but the training network is going to give always the same range of values.

We are trying to obtain networks that are valid for different cases. There are many different approaches, but we are going to focus on High Dimensional Neural Network Potentials (HDNNP)[30].

1.4 High Dimensional Neural Network Potentials

So instead of training the molecules as a whole, we can try to study the individual atoms. The atoms are going to be in very similar environments. For instance, a carbon atom can be in a small range of possibilities: single bond, a double, an aromatic environment, end of chain, electrophile due to electronegative bond, with steric repulsion due to neighbors, ... We can picture a carbon atom as described by a sum of all of these effects, and this is exactly how NN work.

So we can build a NN for carbon, training all the weights and biases. And we can think that it describes accurately any carbon atom similar to those trained. We can use then those network for calculation of a carbon atom. This atom is described by certain geometry, for instance three iodine atoms bonded to it. This geometry, distances and angles when inputted in the network would be multiplied by the first weight matrix to obtain some values, that, for instance, we can think that is the kind of bonds that it has. Then we have another layer in the network that could mean the chemical structure around the atom, in this case we would have high numbers in a node that meant steric repulsion and electrophilic. And then a last combination of all those numbers can lead to a single value of atomic energy.

Then we would have a different carbon atom, for instance an aromatic one. This one has thereby a different geometry, so the weights make other hidden nodes with higher numbers, and after passing through the network, a different value for the energy is obtained. As we see, we need good trained networks and also optimal network shape. And for hydrogen we would build a different network, perhaps with a different number of layers or nodes; and definitely with different weights because its environment would be very different. This way we have many different outputs for each time an atom has gone through the network, so let's now study what we do with them.

Notice that the explanation of what happens in each layer of the network is entirely made up, it is just to show, how it can be trained in order of growing complexity, but the network can find patterns different than those. After all, everything that happens inside the

network is just mathematics; there is no actual physics or chemistry working here, no wave function or density functional, just some numbers that happen to describe well the problem.

1.4.1 Output

In the case of HDNNP, the post network treatment of the data is very simple. We just add all atomic energies to get the molecular total energy. No further modifications need to be done; we train the network this way and everything will be done inside it.

Also we can notice that with the same network we can get more information: the gradients of the energy. This is the derivative of the energy to a movement in any of the 3 dimensions of any of the N atoms. This way we get another physical magnitude after the network, that can be used to avoid any bias that happen for using the energy alone. For this to be possible every step has to be differentiable, so the activation functions have to be carefully chosen.

Another thing to notice about the output is that it depends entirely on the trained data. Specifically, as we need computational resources to train the network, there is no experimental molecular energy, the results can only be as good as the trained ones. This is crucial. We are not going to get better results than those used for training. But if the training is good enough, we can have acceptable results, in a fraction of the original time, and this is the key of the usage of the potentials, for instance to obtain a lot of PES conformations in a fraction of the time that were would be used using the training method. But of course, the more training data it has, the better the results, so we need to spend a lot of time making the calculations for training. At the end some equilibrium needs to be found.

1.5 Input

The real difficulty in this method is not the output or the network itself, but the input data. This is due to the need of modification of the data in some ways, as we are going to see.

1.5.1 Invarational Input

We can start thinking the easiest case. Why don't we input the Cartesian coordinates inside the network, and let it work it out. We can do it, and we would obtain a energy value. And after that we can take the same molecule, and move every atom at once 1 to the right. We know the energy should be the same, because the energy would not change with translation. But know the Cartesian coordinates have changed, so the numbers inside the network change as well as the result. In fact there are three things the energy should be invariant to:

1. Translation: Moving the whole molecule.
2. Rotation: Turning the whole molecule.
3. Labelling of atoms: Changing the order in which the atoms appear.

If we use the Cartesian coordinates, all three premises are invalid. So we need to find a different way of expressing the input, and modifying it accordingly.

Another possibility can be to subtract the Cartesian coordinate of the atom that is going to be inputted in each case. This way the translation is solved, but not the others.

We can think about the z-matrix. Using distances, angles and dihedrals instead of Cartesian coordinates. This way we do not have to worry any longer about translation or rotation. But still the problem of the order of the atoms arises. Which one should we choose first? There is no actual way we can order the atoms of a molecule, besides the element they belong.

The solution to this can be adding the numbers. Adding all the distances as a single node. But making this has its issues as well. We could have two different scenarios: one distance between the central (inputted in the network) atom and a different one; and two distances that happen to add the same as the former. And if we just use the sum, the number that we are getting is the same, and we are going to obtain the same energy for two different environments.

In order to solve this issues we use Symmetry Functions (G) .

1.5.2 Symmetry Functions

When using a symmetry function, we do two things: first, solve the problem, described above, and second, create a more useful representation of the chemical environment.

We have said that we can use the distances. But in fact, we are more worried about what atoms are close than those far away. If we input the distance, the values are going to be proportional to it, when we want inversely proportional dependency. In fact we may want to discard any interaction beyond some distance, the cutoff radius . This way, if atoms are too far to interact we save computational resources. This cutoff can be implemented through some function such as (1.7).

$$f_c(R) = \begin{cases} \frac{1}{2} \left[\cos \left(\frac{\pi R}{R_c} \right) + 1 \right] & R \leq R_c \\ 0 & R > R_c \end{cases} \quad (1.7)$$

$$\frac{\partial f_c(R_{ij})}{\partial \alpha_l} = \begin{cases} -\frac{1}{2} \sin \left(\frac{\pi R_{ij}}{R_c} \right) \frac{\pi}{R_c} \frac{\partial R_{ij}}{\partial \alpha_l} & R_{ij} \leq R_c \\ 0 & R_{ij} > R_c \end{cases} \quad (1.8)$$

This is just an example, many other functions can be used. But using these function alone is not enough. As we said, we can introduce some order in the elements, but not for the atoms of each element, so we have to sum them. This way we would obtain just one symmetry function for each element. But we can multiply this cutoff function by a Gaussian,

obtaining something like equation (1.9):

$$G_n^{I:J}(i \in I; \eta(n), R_s(n), R_c(n)) = \varphi_n^{I:J} \left(\sum_{\substack{j \in J \\ j \neq i}} e^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij}) \right) \quad (1.9)$$

$$\frac{\partial G_n^{I:J}(i)}{\partial \alpha_l} = \sum_{\substack{j \in J \\ j \neq i}} e^{-\eta(R_{ij}-R_s)^2} \left(-2\eta(R_{ij} - R_s) f_c(R_{ij}) \frac{\partial R_{ij}}{\partial \alpha_l} + \frac{\partial f_c(R_{ij})}{\partial \alpha_l} \right) \quad (1.10)$$

$$\frac{\partial R_{ij}}{\partial \alpha_l} = \begin{cases} \frac{\alpha_i - \alpha_j}{R_{ij}} & l = i \\ \frac{\alpha_j - \alpha_i}{R_{ij}} & l = j \\ 0 & l \neq i \text{ and } l \neq j \end{cases} \quad (1.11)$$

In this case some parameters appear. R_s takes into account some displacement of the Gaussian. This way we can focus on values at different distances. If we create different values for, let's say $R_c = 0, 2$ and 4 around the central atom, each function is going to focus on atoms more close or far away the central one.

And the other parameter is η that takes into consideration the width of the Gaussian, some to take into account many different atoms, or others will be more selective. This way, we make many symmetry functions for many different parameters, so the input layer will be made of these N symmetry functions, where N can be chosen at will.

In equation (1.9), capital I and J refer to the elements and small caps i and j refer to the atoms itself. So we can see that the sum is all over the atoms of a certain element, but not others. This is because due to the very different chemical nature of different elemental atoms, we probably want to have them as separate input to the network.

Also, we can take into account the angular part of the problem, as in equation (1.12). Now, different parameters appear, but the goal is the same: obtaining symmetry functions that take into account the symmetry, and making sure that different chemical environments

have different symmetry functions.

$$\begin{aligned}
& G_n^{I:JK} (i \in I; \eta(n), \zeta(n), \lambda(n), R_c(n)) \\
&= \varphi_n^{I:JK} \left(2^{1-\zeta} \sum_{\substack{j \in J, k \in K \\ j \neq i, k \neq i, k \neq j}} (1 + \lambda \cos \theta_{jik})^\zeta \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}) \right)
\end{aligned} \tag{1.12}$$

$$\begin{aligned}
\frac{\partial G_n^{I:JK}(i)}{\alpha_l} &= 2^{1-\zeta} \sum_{\substack{j \in J \\ k \in K, i \neq j, i \neq k}} e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \left[\right. \\
& \quad \zeta (1 + \lambda \cos \theta_{jik})^{\zeta-1} \lambda \frac{\partial \cos \theta_{jik}}{\partial \alpha_l} f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}) \\
& \quad - (1 + \lambda \cos \theta_{jik})^\zeta 2\eta \left(R_{ij} \frac{\partial R_{ij}}{\partial \alpha_l} + R_{ik} \frac{\partial R_{ik}}{\partial \alpha_l} + R_{jk} \frac{\partial R_{jk}}{\partial \alpha_l} \right) f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}) \\
& \quad + (1 + \lambda \cos \theta_{jik})^\zeta \frac{\partial f_c(R_{ij})}{\partial \alpha_l} f_c(R_{ik}) f_c(R_{jk}) \\
& \quad + (1 + \lambda \cos \theta_{jik})^\zeta f_c(R_{ij}) \frac{\partial f_c(R_{ik})}{\partial \alpha_l} f_c(R_{jk}) \\
& \quad \left. + (1 + \lambda \cos \theta_{jik})^\zeta f_c(R_{ij}) f_c(R_{ik}) \frac{\partial f_c(R_{jk})}{\partial \alpha_l} \right]
\end{aligned} \tag{1.13}$$

$$\frac{\partial \cos \theta_{jik}}{\partial \alpha_l} = \begin{cases} \frac{(\alpha_j - \alpha_i)(R_{ik}^2 \cos \theta_{jik} - R_{ij} R_{ik}) + (\alpha_k - \alpha_i)(R_{ij}^2 \cos \theta_{jik} - R_{ij} R_{ik})}{R_{ij}^2 R_{ik}^2} & l = i \\ \frac{(\alpha_k - \alpha_i) R_{ij} - (\alpha_j - \alpha_i) R_{ik} \cos \theta_{jik}}{R_{ij}^2 R_{ik}} & l = j \\ \frac{(\alpha_j - \alpha_i) R_{ik} - (\alpha_k - \alpha_i) R_{ij} \cos \theta_{jik}}{R_{ik}^2 R_{ij}} & l = k \\ 0 & \text{other} \end{cases} \tag{1.14}$$

One last consideration on this functions. The symbol ϕ appearing in front of the functions is the scaling function. This function is included so the actual value of each symmetry function ranges between the same values, so it can be more trusted, deleting outliers. The max and min found in (1.15) are that maximum and minimum values found during the training and can be used after that, if stored properly.

$$\varphi_n(x) = \frac{2(x - \min G_n^\circ)}{\max G_n^\circ - \min G_n^\circ} - 1 \quad (1.15)$$

Also we can see that the symmetry functions still have to be inputted in some order, but now it's easier to get some protocol, such as increasing parameter value in some order of the parameters.

Notice that these are just some examples, but other functions may appear as well. The optimization of the functions themselves and the parameters used is another part to optimize within this method.

1.5.3 Forces

Next to those equations, the gradients appear. They are used to obtain the forces, this is, the minus gradients of the energy. Using the equation (1.19).

$$F_{\alpha_l} = -\frac{\partial E}{\partial \alpha_l} = -\sum_{i=1}^{N^{\text{at}}} \frac{\partial E_i}{\partial \alpha_l} = -\sum_I \sum_{i \in I} \sum_{n=1}^{N^{\text{sym}}(I)} \frac{\partial E_i}{\partial G_n^I(i)} \cdot \frac{\partial G_n^I(i)}{\partial \alpha_l} \quad (1.16)$$

In the preceding equations, the formulas for the gradients of the symmetry functions have been obtained. We only need the derivative of the energy with respect to the symmetry functions, and this can be achieved by performing the chain rule all over the layers of the network.

1.6 Energy and Force Calculation

Now that all elements have been introduced, we can make a short test of how the calculation is done.

Let's say that we have obtained all the symmetry functions for an atom i and we have it as a vector $\vec{G}(i)$. This atom belongs to element I . Let's suppose we have a two hidden layer long neural network. The equation to obtain the energy will be as follows:

$$E_i = \mathbf{A}_{23}^I f_2 \left[\mathbf{A}_{12}^I f_1 \left(\mathbf{A}_{01}^I \vec{G}(i) + \mathbf{b}_{01}^I \right) + \mathbf{b}_{12}^I \right] + b_{23}^I \quad (1.17)$$

In equation (1.17) we have named the weights and the biases of each element with two numbers: the layer we are coming from and the layer we are going to, so always the second number is the first plus one. We start counting from zero, the input layer made of symmetry functions is given number zero and the output layer of the energy is given number three. Notice that the output layer does not have activation function because it is considered to be linear. Notice that, as stated, all the time we have a relation between the geometry, that appears makes the atom i of $G(i)$ and the energy, but we do not need to know the actual shape of it, we just approximate it through several matrix multiplications; neural networks imitate the actual and unknown function.

The equation for the force comes by applying the chain rule through the entire equation:

$$F_{\alpha_l} = - \sum_I \sum_{i \in I} \frac{\partial E_i}{\partial \alpha_l} \quad (1.18)$$

$$\begin{aligned} \frac{\partial E_i}{\partial \alpha_l} &= \mathbf{A}_{23}^I f_2' \left[\mathbf{A}_{12}^I f_1 \left(\mathbf{A}_{01}^I \vec{G}(i) + \mathbf{b}_{01}^I \right) + \mathbf{b}_{12}^I \right] \mathbf{A}_{12}^I f_1' \left(\mathbf{A}_{01}^I \vec{G}(i) + \mathbf{b}_{01}^I \right) \mathbf{A}_{01}^I \frac{\partial \vec{G}(i)}{\partial \alpha_l} \\ &= \mathbf{A}_{23}^I f_2' [\vec{a}_2] \mathbf{A}_{12}^I f_1' (\vec{a}_1) \mathbf{A}_{01}^I \frac{\partial \vec{G}(i)}{\partial \alpha_l} \end{aligned} \quad (1.19)$$

Where the vectors \vec{a}_n can be calculated just once in the energy calculation and then stored for the forces, reducing the calculation effort. Remember that even if those are the full formulas, the way the computer calculates it is just sequential, applying functions and matrix multiplication where it needs it.

For the forces we can see that the biases disappear as they appear just in an additive way, the weights stay and now we have to apply the derivative of the activation functions for the original arguments. Also notice that we are going to have 3N forces, but their differences will be only due to changes in the symmetry functions, this is, the geometry, as the neural

network is always the same for all atoms of a given element.

2 Goals

As a part of this Master thesis, the student spent 3 months at the Software for Chemistry and Materials (SCM) * in an internship that could be called Development of HDNNP in Fortran 2003 in Amsterdam Modelling Suite (AMS) [31], under the supervision of Matti Hellström.

AMS is the main product of SCM, that allow many different chemical calculations. It does indeed have some implementations of HDNNP, such as the ANI-1x , ANI-1ccx and ANI-2x. All of these are implementations of HDNNP where the weights, biases, symmetry functions, and almost everything are known before hand, just like most calculation programs, the user just have to input the geometry of their chemical system and run the calculation.

The parametrization, of course, has to be done taking into account some elements and specific chemical environments, usually organic molecules. The advantage of that method is that the user has little to worry about, but on the other hand, there possibilities of usage are little.

Most of the code of AMS is done in Fortran 2003, in a Object Oriented Programming (OOP) , which is the modern standard for programming. However, tools for ML in Fortran are not too well developed. The easiest way to get Machine Learning to work is with Python. Python has a lot of tools and packages to deal with Data Science (DS) and ML, such as Keras and TensorFlow. But Python has a great disadvantage: the efficiency for long

*Dr. S. J. A. van Gisbergen, CEO
Software for Chemistry Materials BV
De Boelelaan 1083
1081 HV Amsterdam
The Netherlands

calculations is very poor. That is why the NNs in these packages are actually programmed in other languages, and Python only calls them.

If we take a look to everything said about HDNNP in the introduction, one can guess that the process is divided in two areas: calculations that depend on the number of atoms, and calculations that depend on the number of elements. Since in a typical calculation it's very uncommon that we are going to have more than 6 or 8 different elements in the same calculation, the computational cost is going to be on the atomic part. And of all the atomic calculations, the most costly part will be obtaining the distances and angles, that depend on the number of atoms with N^2 and N^3 respectively. This means that the most computational costly part is obtaining the Symmetry Functions, not the neural Network itself. And currently the Symmetry Functions were calculated in Python.

So there was an interest in building HDNNP for Fortran 2003 for these two reasons: accordance with the rest of the code and computer efficiency. Furthermore, we wanted to build code that we could use to build any parametrization, instead of depending on the already parametrized data.

Actually there is a piece of code that can do almost all of the above. RuNNer[20–24] is a Fortran tool for the development of any HDNNP. The code is freely available under the GPL3 license. But because of this license this code cannot be used for commercial use inside AMS. Besides, the code is in Fortran 90 , not optimal.

The goal in this internship was to build working HDNNP code in Fortran 2003. In order to develop it, the student followed and implemented the information found in this paper [30].

By the end of the internship the student had written more than 3000 lines of code. The code has all of the following features:

- Reading an input file containing the element information, neural network shape and all of the symmetry functions parameters, among others.
- Ordering the Symmetry Functions by some internal criteria and sorting them using

bubble sort.

- Getting the chemical system and obtaining the distances and angles, with and without periodic boundary conditions.
- Getting the cutoff functions, and gradients of those, with different functions available, at will of the user.
- Getting radial and angular symmetry functions, for all the atoms in the simulation.
- Getting as well the gradients of the symmetry functions, in an efficient way.
- Scaling or not, at will of the user, and with different functions, the obtained Symmetry Functions, as it allows better calculations, using data from an external file.
- Reading from an external file the weights and biases of a Network.
- Building a Neural Network with the geometry and activation functions as desired, and applying those weights and biases.
- Introducing the symmetry functions in the networks and obtaining the energies.
- Introducing as well the Symmetry Functions Gradients and obtaining the forces.
- Summing over all the atoms of the system to obtain total energies and forces.
- Integrating all of the above in the AMS style, so the code is written in the same way as the rest of it.
- Allowing my code to be run smoothly when the correct keyword (hdnnp) is used in the AMS input.
- Building examples and documentation to use my code.

Those are some of all the things achieved in the internship. However, some things could not be done, due to lack of time, such as training my own weights and optimize and paralellize the calculation time. However, even if those things could not be achieved, a lot of work was done and we can obtain energies if we have a pretrained model.

Even if the internship consisted in programming, due to license issues, it would be difficult to show here the code itself. Furthermore, it would have no sense to show 3000 lines of code. so instead of that, some calculations were done with the student's code to show the results obtained and to discuss this kind of potentials, and it's applications.

3 Results

Now we are going to put into practice the knowledge on HDNNP that we have acquired so far. As said, the code made by the student, doesn't have the ability to train it's own Neural Network yet, So we are going to use RuNNer for that use. Keep in mind that this section is just to put in practice the code, but the real goal was always to program the code itself.

3.1 Training of the model

In order to build a Neural Network Potential that works with some of the molecules of the previous part, we need to choose which structures are going to be used for the training. Of course, the set of structures to train the network has to be smaller than the total one, or the training would have no sense at all; we want to obtain new information. In order to keep it simple, we are going to use only the A molecules from the other part of the work. Furthermore, we will focus only on the neutral radicals, enough to understand how these potentials work.

In order to train it, we chose to select just one molecule: A11. Of course, just one molecule is not enough information to build a neural network by itself, but if we carry some molecular dynamics simulation, we will be able to obtain several values of geometry and energy. This is exactly what was done, and the details of the simulation came here.

The method for obtaining the energies is was Density Functional based Tight Binding (DFTB) [32][33]. First, a simulation was done consisting in 10000 steps and a ramping was

Table 3.1: Number of symmetry functions used between elements, radial and angular. The more bonds between atoms, the more symmetry functions we should use for good description. When there are more than one symmetry function, they differ in the parameters used.

Central atom	Radial			Angular					Total	
	H	C	N	HH	HC	HN	CC	CN		NN
H	4	5	1	0	6	4	3	3	0	26
C	5	5	5	6	4	0	4	4	0	33
N	2	6	0	0	4	0	4	0	0	16

done between 100 K and 4000K. However, at high temperatures, the molecule dissociated, so another simulation consisting on 10000 steps between 100K and 1500 K was done. For the final results we took the first half of the structures of the first experiment and all of the second one.

What we got was a big number of different geometries of molecule A11 with slightly different energies. So we have the input and the output for a Neural Network, and we can train one.

In order to train the model, since the student's code is unable yet to do so, RuNNer was used. It builds the symmetry functions and gives all the files necessary to make calculations with it, including weights for all the elements and the scaling file.

As a test of how good is the training, you can observe figure [3.1 on the next page](#). It represents a fit between the frequencies of the normal modes between the DFTB method and the NN (with the RuNNer program). The NN method does not give the normal nodes by itself, but they can be calculated from the output of the program with a different program, in this case AMS. We can see that the linearity is very good, proof that the network is good for obtaining data for molecule A11.

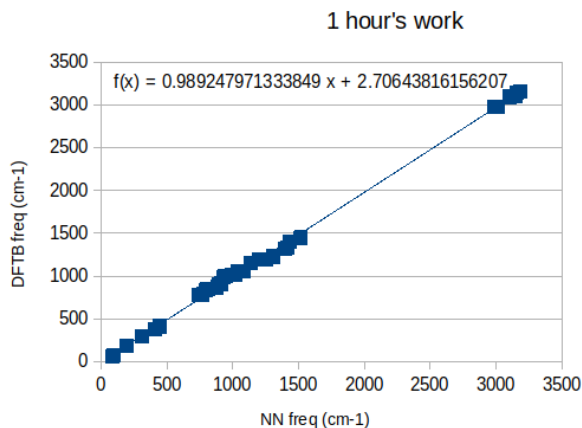


Figure 3.1: Fitting between normal modes from the reference DFTB method and the NN method obtained with RuNNer.

3.2 Testing the model

Now that we have a trained Network, and since the files are compatible between RuNNer and the student's code, we can perform the calculations on the latter, and comment some results. The molecules chosen to get results were the neutral radicals of group A, since we have to use similar molecules as the trained one, for better results.

The energies obtained by themselves mean nothing. We have to choose one of reference to subtract the others. Since it is trained with A11 and the smallest error is going to be it, we can subtract that value. the results obtained with the HDNNP method appear in a much smaller range than the other methods, to solve this we can just divide the difference by the reference energy, and we have all results in the same range of values. So, the formula used in 3.2 and the table is:

$$EF = \frac{E_i - E_{A11}}{-E_{A11}} \quad (3.1)$$

For every method, although for DFT it is not really needed, just the difference is good.

The results obtained appear in table 3.2 on the facing page and we can also see these results in figure 3.2 on the next page.

We can see that the shape of the HDNNP (in full black line, appearing in every individual

Table 3.2: Comparison between a DFT method and the HDNNP one. Energy Fraction involves the operation in equation (3.1), so we can compare the results.

Molecule	B2PLYP-D3			HDNNP		
	Energy (Eh)	Energy Fraction	Calc. Time	Energy (Eh)	Energy Fraction	Calc. Time
A0	-55.8577	0.8068	00:00:05.98	-2.24136	0.8901	00:00:00.10
A1	-95.1457	0.6709	00:00:21.08	-6.49362	0.6816	00:00:00.15
A2	-134.4386	0.5350	00:01:06.98	-10.11698	0.5039	00:00:00.17
A3	-134.4366	0.5350	00:00:58.90	-11.27968	0.4469	00:00:00.18
A4	-213.0239	0.2633	00:03:36.21	-18.35914	0.0998	00:00:00.22
A5	-173.7334	0.3991	00:02:03.95	-14.54355	0.2869	00:00:00.19
A6	-213.0282	0.2632	00:05:53.44	-18.68491	0.0838	00:00:00.21
A7	-172.4995	0.4034	00:06:13.70	-10.58704	0.4809	00:00:00.18
A8	-211.7891	0.2675	00:08:48.75	-13.92879	0.3170	00:00:00.20
A9	-251.1087	0.1315	00:11:18.82	-18.07888	0.1136	00:00:00.22
A10	-290.4097	-0.0044	00:14:57.36	-23.64308	-0.1593	00:00:00.29
A11	-289.1411	0.0000	00:12:02.77	-20.39482	0.0000	00:00:00.24

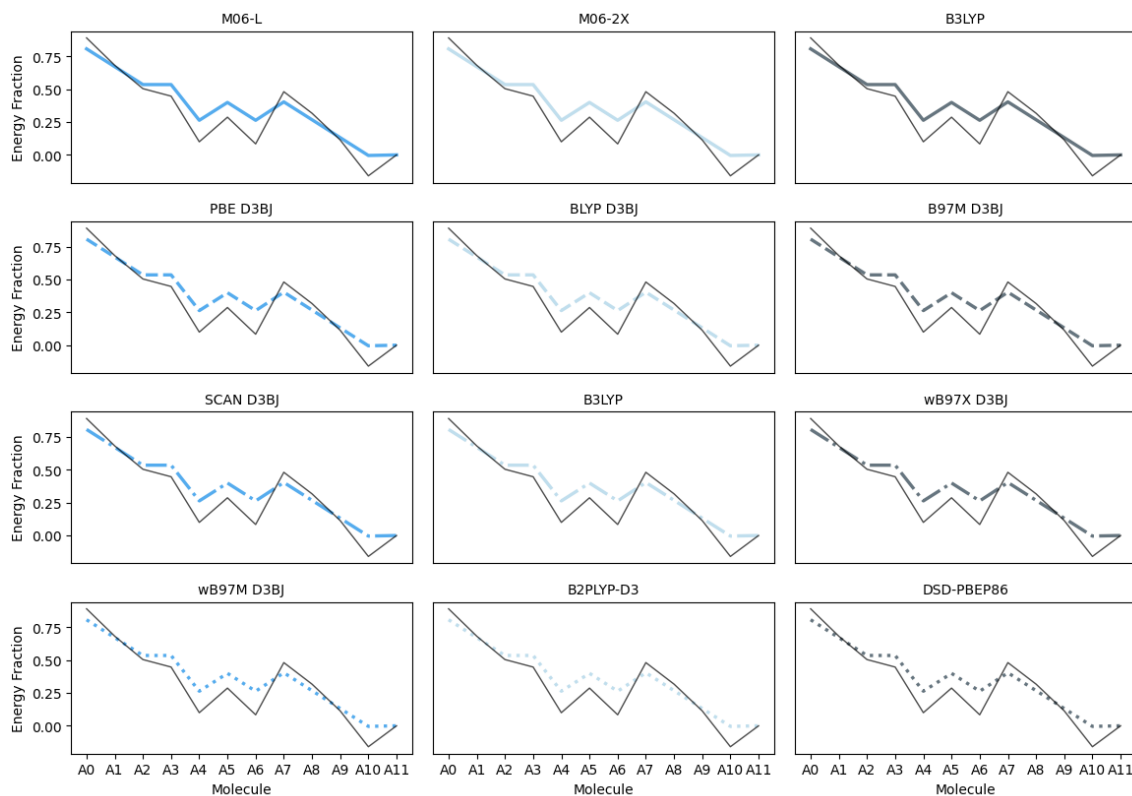


Figure 3.2: The different DFT functionals (in different colors and line styles plotted in each case against the HDNNP results (in black, in every subplot).

plot) resembles that obtained from the other methods (each in its own subplot, in different colors and linestyles), with the same ups and downs. However, we can see that the difference in energy is very big in most cases, whether is by excess or defect. In fact, at this scale we cannot see any difference in the DFT methods, only with HDNNP. The reason is obvious: the model is not trained for these molecules, so it fails.

Furthermore we can see that the more different is the molecule, the biggest is the difference. The molecule A11 does not have any CH_3 or NH group, so it fails in this cases. In order to improve the calculation we would need more conformations of more molecules, for instance 80 of the molecules, and get again some results.

However, this result is worth because it shows how HDNNP emulate the functional used to train the network, in this case DFTB, and to some extent does reproduce the trends.

Now, take a look to the calculation times. Between the DFT method and the HDNNP one there is a huge difference in time. This becomes more impacting given that the calculations have been done with the student code; and it needs yet much more optimization to be commercial. And realize that this time does not depend on the goodness of the values of the energy: for a good method we would have different weights and biases, but the time they take for the calculation would be the same.

This is the main advantage of this method. The results can be better or worse, depending on how good the training is, but the time will always be a lot smaller. For that reason, this potentials are specially suited for situation when we want to obtain a lot of different energies from very similar inputs, such as PES or simulations of liquids, for instance.

4 Conclusions

In this work we have made the following:

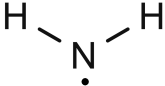
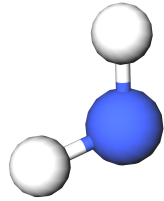
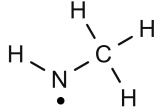
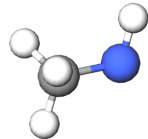
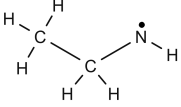
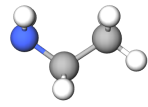
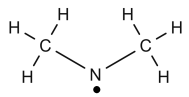
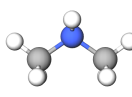
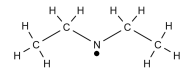
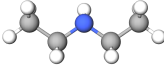
- We have studied a very uncommon kind of calculation that does not rely on physical properties, but in mathematics and regression that uses Artificial Intelligence: High Dimensional Neural Network Potentials.
- The student has programmed from scratch code that is able to obtain energies and forces using these kind of calculations, including reading input information, obtaining and shaping Symmetry Functions and building working Neural Networks.
- This code has been able to be integrated inside the rest of the AMS code, so it can be called in a normal calculation using the appropriate keyword.
- In order to test it, we have build one parametrization of this code, using one molecule as template and we have tested it with the set of molecules, discussing the results obtained.

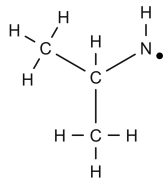
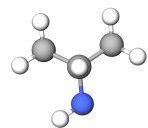
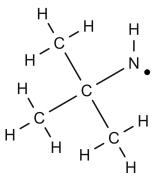
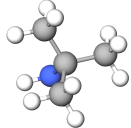
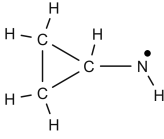
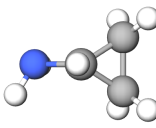
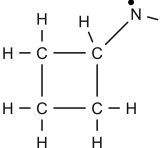
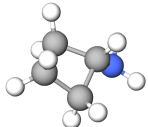
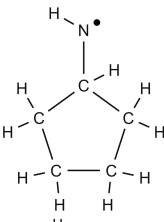
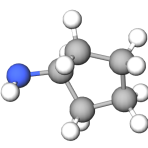
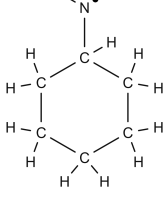
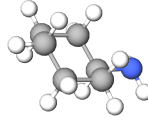
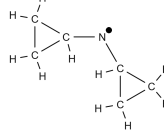
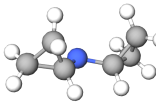
The way of expanding this work in the future includes working on the code: allowing it to train by itself the networks and making it more efficient.

A Molecules

In the following table, all molecules appearing in this thesis.

Table A.1: Molecules used in this thesis. Only the radicals are shown, but the given name is that of the non radical species.

Code	Short	Long	2D Rep	3D Rep	Name
A0	NH ₂	NH ₂			Ammonia
A1	NCH ₃	NHCH ₃			Methylamine
A2	NC ₂ H ₆	NHCH ₂ CH ₃			Ethylamine
A3	NC ₂ H ₆	N(CH ₃) ₂			Dimethylamine
A4	NC ₄ H ₁₀	N(CH ₂ CH ₃) ₂			Diethylamine

Code	Short	Long	2D Rep	3D Rep	Name
A5	NC_3H_8	$\text{NHCH}(\text{CH}_3)_2$			Isopropylamine
A6	NC_4H_{10}	$\text{NHCH}(\text{C}(\text{CH}_3)_3)$			Tert-Butylamine
A7	NC_3H_6	$\text{NHCH}(\text{CH}_2)_2$			Cyclopropylamine
A8	NC_4H_8	$\text{NHCH}(\text{CH}_2)_3$			Cyclobutylamine
A9	NC_5H_{10}	$\text{NHCH}(\text{CH}_2)_4$			Ciclopentylamine
A10	NC_6H_{12}	$\text{NHCH}(\text{CH}_2)_5$			Ciclohexylamine
A11	NC_6H_{10}	$\text{N}(\text{CH}(\text{CH}_2)_2)_2$			Diciclopropylamine

B Python

In this appendix an example of a python script developed for the thesis is shown. In this script appears the table for the basis and those associated to it, in our case we were not interested in it, so we can imagine as if it had only one entry:

```
1 import mysql.connector
2 from mysql.connector import Error
3 import pandas as pd
4
5 def create_server_connection(host_name, user_name, user_password):
6     connection = None
7     try:
8         connection = mysql.connector.connect(
9             host=host_name,
10            user=user_name,
11            passwd=user_password
12        )
13        print("MySQL Database connection successful")
14    except Error as err:
15        print(f"Error: '{err}'")
16
17    return connection
18
19 def create_database(connection, query):
20     cursor = connection.cursor()
21     try:
22         cursor.execute(query)
```

```

23     print("Database created successfully")
24 except Error as err:
25     print(f"Error: '{err}'")
26
27 def create_db_connection(host_name, user_name, user_password, db_name):
28     connection = None
29     try:
30         connection = mysql.connector.connect(
31             host=host_name,
32             user=user_name,
33             passwd=user_password,
34             database=db_name
35         )
36         print("MySQL Database connection successful")
37     except Error as err:
38         print(f"Error: '{err}'")
39
40     return connection
41
42 def execute_query(connection, query):
43     cursor = connection.cursor()
44     try:
45         cursor.execute(query)
46         connection.commit()
47         print("Query successful")
48     except Error as err:
49         print(f"Error: '{err}'")
50
51 def read_query(connection, query):
52     cursor = connection.cursor()
53     result = None
54     try:
55         cursor.execute(query)

```



```

56         result = cursor.fetchall()
57         return result
58     except Error as err:
59         print(f"Error: '{err}'")
60
61 def pandas_query(connection, query, columns):
62     from_db = []
63     results = read_query(connection, query)
64     for result in results:
65         result = list(result)
66         from_db.append(result)
67
68     df = pd.DataFrame(from_db, columns=columns)
69     return df
70
71 def execute_list_query(connection, sql, val):
72     cursor = connection.cursor()
73     try:
74         cursor.executemany(sql, val)
75         connection.commit()
76         print("Query successful")
77     except Error as err:
78         print(f"Error: '{err}'")
79
80 #Create the connection with SQL
81 pw = "PASSWORD"
82 connection = sql.create_db_connection("localhost", "USER", pw, "DATABASE")
83
84 ### MOLECULES ###
85
86 #Headers for molecules
87 column = ["molecule_id", "code", "total_atoms", "multiplicity", "N", "H", "C",
            "mass", "heavy"]

```

```

88 #Molecule(s) to input
89
90 condition = """
91 WHERE """ + column[3] + ' = ' + str(1) +
92 # In this example the ones that are radicals
93
94 #Extract the molecules
95 query = """
96 SELECT *, total_atoms - H AS heavy
97 FROM molecule""" + condition + """
98 ORDER BY family_code, charge;"""
99 #Get heavy atoms.
100
101 m = sql.pandas_query(connection, query, column)
102
103
104 ### FUNCTIONALS ###
105 program="Orca"
106
107 #Headers for functionals
108 column = ["functional_id", "functional_name", "dispersion", "type", "
           category_1", "category_2", "platform"]
109 #functional(s) to input
110 condition = """
111 WHERE """ + column[6] + ' ="' + program + '"' + """
112 """
113 # In this example: (1) Functionals for Orca
114
115 #Extract the functionals
116 query = """
117 SELECT *
118 FROM functional""" + condition + """
119 ORDER BY type, functional_id; """

```

```

120
121 f = sql.pandas_query(connection, query, column)
122
123 #### BASIS ####
124
125 #Headers for basis
126 column = ['basis_id', 'basis_name', 'zeta', 'polarization', 'diffuse',
            'light_AO', 'light', 'heavy_AO', 'heavy', 'superheavy_AO', '
            superheavy', 'ABO']
127 #basis to input
128 condition = ""
129 WHERE "" + column["name"] + ' = ' + "def2-TZVPP"
130 # In this example: specific basis
131
132 #Extract the basis
133 query = ""
134 SELECT *
135 FROM basis"" + condition + ""
136 ORDER BY basis_id; ""
137
138
139 b = sql.pandas_query(connection, query, column)
140
141 #Maximum number of functions we will allow, we might want to use less
142
143 maxFunctions = 999999
144
145
146
147 # Now we have the three tables imported, we can build the experiment
148
149 grid = 'grid4'
150 scf = 'tightscf'

```

```

151 globalQuery = """
152 INSERT INTO experiment (m_id, f_id, b_id, functions, molecule, functional,
      basis, calculation, integral, aux_basis, scf, grid, charge, multiplicity)
153 VALUES (%s, %s, %s, %s, %s,%s, %s, %s, %s, %s,%s, %s, %s, %s)
154 """
155 list = []
156 for i, molecule in m.iterrows():
157     for j, functional in f.iterrows():
158         for k, basis in b.iterrows():
159             query = """
160             SELECT functions
161             FROM NumberFunctions
162             WHERE molecule_id = """ + str(molecule['molecule_id']) + """
163             AND basis_id = """ + str(basis['basis_id']) + ';'
164             functions = sql.read_query(connection, query)[0][0]
165             if functions < maxFunctions:
166                 calculation = 'RKS' if molecule['multiplicity'] == 1 else 'UKS
,
167                 query = """
168                 SELECT t.under20, t.over20
169                 FROM type t
170                 JOIN functional f
171                 ON t.type_id = f.type
172                 WHERE f.functional_id = """ + str(functional['functional_id'])
+ ';'
173                 two_int = sql.read_query(connection, query)[0]
174                 n_int = two_int[0] if molecule['heavy'] <= 20 else two_int[1]
175                 query = """
176                 SELECT i.name, i.fitting
177                 FROM integrals i
178                 WHERE int_id = """ + str(n_int) + ';'
179                 integral = sql.read_query(connection, query)[0]
180                 query = """

```

```

181         SELECT a.J, a.JK
182     FROM ABO a
183     WHERE a.ab_id = """ + str(basis['ABO']) + '>';
184     two_ABO = sql.read_query(connection, query)[0]
185     auxiliary = two_ABO[0] if integral[1] == 'J' else two_ABO[1]
186     list.append((molecule['molecule_id'], functional['
functional_id'], basis['basis_id'], functions, molecule['code'],
functional['functional_name'], basis['basis_name'], calculation, integral
[0], auxiliary, scf, grid, molecule['charge'], molecule['multiplicity']))
187
188 sql.execute_list_query(connection, globalQuery, list)
189
190 if __name__ == "__main__":
191     pass

```

Listing B.1: For generating a correct Experiment table with all the conditions we want to specify.

C MySQL Database

In this appendix a description of the MySQL database used is made in figure C.1.

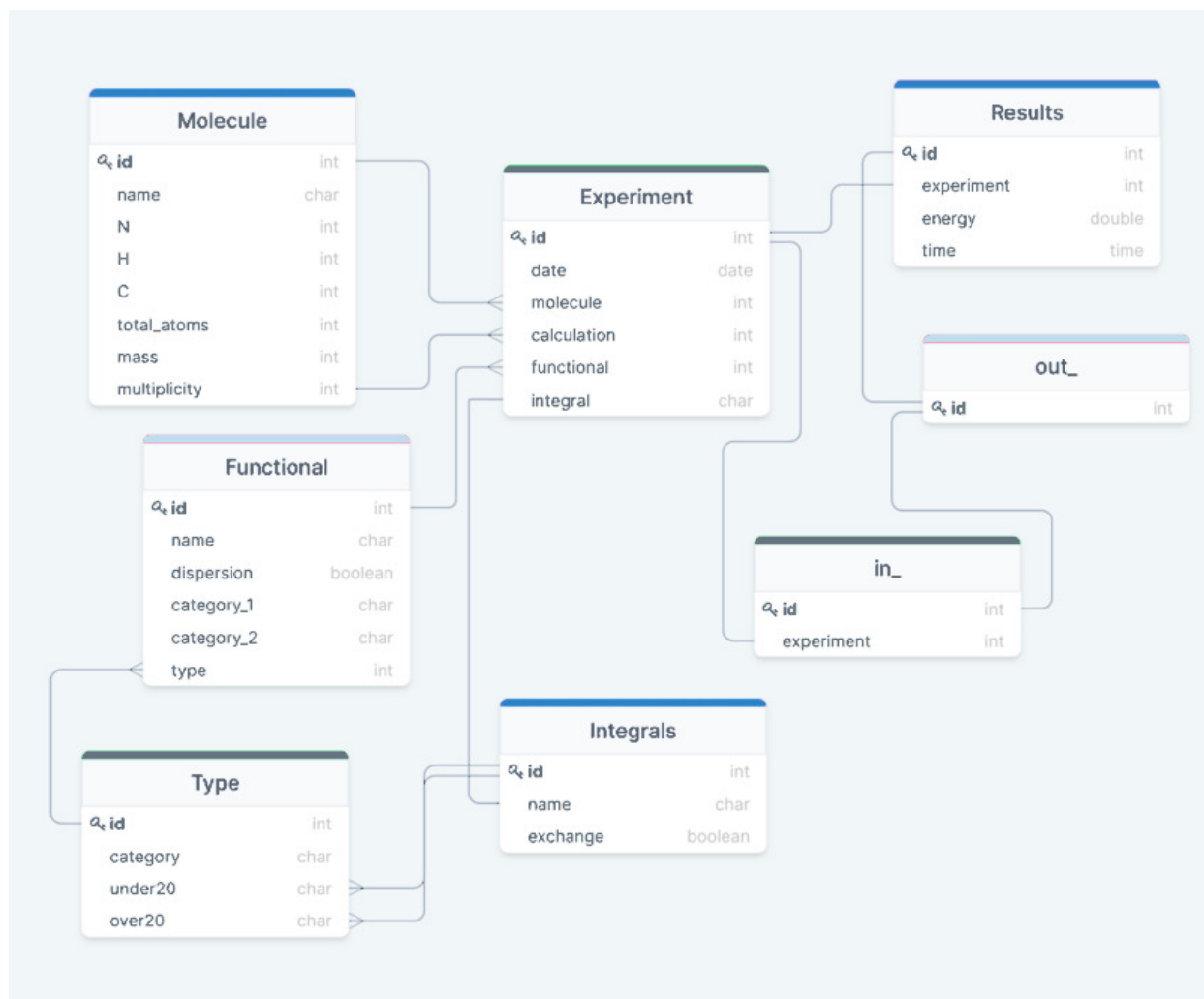


Figure C.1: The MySQL diagram

First of all, this diagram is a shorter version of the original one. The original contained tables for basis sets, charged molecules, many different elements, ... But since finally those

possibilities have not been included in the work, here appears just a description of what has been used.

Table Molecule contains the description for molecules, and table Functional the same for the functionals. Those values are directly inputted inside the Experiment table. But the integral value to choose depends on the type of functional and whether the number of atoms is under 20 or not. So we create a table Integrals with every possible value of it, and a table Types defining for the types of functional which integral should be used in each case. Then, as we know the type of each functional, the correct integral can be called in Experiment.

When we have the table Experiment populated, we can introduce every calculation we want to perform in in_. Then, the bash file will be called and the calculations will be done and the current experiment is transferred to table out_.

Finally, when the calculations are ready, we can call another script to pass the rows from out_ to results, as well as getting and storing the energy and time fields.

As said, this dataframe as shown is just a small part of what can be done. We can introduce basis sets, different kind of calculations such as geometry optimizations or vibrational studies; we can get other results apart from the energy and the calculation time. It would be very easy to expand this database even more if needed, and it's a well suited tool for storing all values.

D Bash

In these appendix appear the bash files for launching and retrieving calculations using MySQL as data storement.

```
1 #!/bin/bash
2 #Launch all the experiments in in table
3 #Some folders
4 BASHDIR='./'
5 XYZDIR='./coords/'
6 CURDIR='pwd'
7 #Data for mysql
8 SQL='mysql -u USER -pPASSWORD -D DATABASE -N -B -e '
9 #Get experiment
10 QUERY="SELECT COUNT(*) FROM in_;"
11 COUNT='eval "${SQL}\ "${QUERY}\ "'
12 echo $COUNT
13 #Start loop
14 for i in $(seq 1 ${COUNT})
15 do
16     echo "Creating input and launching calculation for experiment: " $i
17     #Get experiment
18     QUERY="SELECT i.id FROM in_ i limit 1;"
19     exp='eval "${SQL}\ "${QUERY}\ "'
20     #Delete that row from in_
21     QUERY="DELETE FROM in_ WHERE id=${exp};"
22     eval "${SQL}\ "${QUERY}\ ""
23     #Include in out_
```



```

24 QUERY="INSERT INTO out_ (id) VALUES (${exp});"
25 eval "${SQL}" "${QUERY}"
26 #Launch calculation
27 QUERY="SELECT e.exp, e.molecule, m.group_code, m.family_code, e.functional
, e.basis, e.calculation, e.integral, e.aux_basis, e.scf, e.grid, e.
charge, e.multiplicity FROM experiment e JOIN molecule m on m.molecule_id
= e.m_id WHERE e.exp = ${exp};"
28 NEW=${SQL}" "${QUERY}"
29 #Separate input
30 FOO='eval "${SQL}" "${QUERY}" | sed -e 's/\t/,/g'
31 IFS="," read -r -a T <<< "${FOO}"
32 #Create folder if it does not exist
33 filePath="${CURDIR}/${T[2]}/${T[3]}/${T[11]}/"
34 mkdir -p ${filePath}
35 cd ${filePath}
36 #Input strings
37 filein="e_${T[0]}${IN}"
38 calcLine="! ${T[6]} ${T[4]} ${T[5]} ${T[7]} ${T[8]} ${T[9]} ${T[10]}"
39 commLine="# ${T[0]} ${T[1]} ${T[4]} ${T[5]}"
40 charLine="*xyz ${T[11]} ${T[12]}"
41 coorFile="${XYZDIR}/${T[1]}${XYZ}"
42 #For keeping track of calculations
43 echo $commLine >> z_folder_data.txt
44 #To generate the orca input file .com
45 #${filein} name of input file
46 #${calcLine} command line
47 #${commLine} comment line
48 #${charLine} charge and mult
49 #${coorFile} xyz file
50 #Create file
51 echo ${calcLine} > ${filein}
52 echo ${commLine} >> ${filein}
53 echo ${charLine} >> ${filein}

```

```

54 tail -n+2 ${coorFile} >> ${filein}
55 echo "*" >> ${filein}
56 #Launch the calculation
57 suborca $filein
58 cd ${CURDIR}
59 done

```

Listing D.1: Script for creating and launching calculations with Orca

```

1 #!/bin/bash
2 #Launch all the experiments in in table
3 #Some folders
4 BASHDIR='./'
5 XYZDIR='./coords/'
6 CURDIR=`pwd`
7 #Data for mysql
8 SQL='mysql -u USER -pPASSWORD -D DATABASE -N -B -e '
9 #Get experiment
10 QUERY="SELECT COUNT(*) FROM out_;"
11 COUNT='eval "${SQL}\ "${QUERY}\ "'
12 echo $COUNT
13 #Start loop
14 for i in $(seq 1 ${COUNT})
15 do
16     QUERY="SELECT LPAD(o.id, 6, 0) from out_ o where o.in_order = ${i};"
17     exp='eval "${SQL}\ "${QUERY}\ "'
18     echo $exp
19     #Get experiment
20     QUERY="SELECT e.exp, m.group_code, m.family_code, e.charge FROM experiment
21     e JOIN molecule m on m.molecule_id = e.m_id WHERE e.exp = ${exp};"
22     FOO='eval ${SQL}\ "${QUERY}\ ' | sed -e 's/\t/,/g' '
23     #Separate input
24     IFS="," read -r -a T <<< "${FOO}"
25     #Find path

```

```

25 file="${CURDIR}/${T[1]}/${T[2]}/${T[3]}/e_${T[0]}.log"
26 energy='grep "FINAL SINGLE POINT ENERGY" $file | awk '{print $5}' '
27 declare -a times
28 times=( 0 0 0 0 0 )
29 for value in {0..4}
30 do
31     b=$(( $value*2+4))
32     times[$value]='grep "TOTAL RUN TIME:" $file | awk -v a="$b" '{print $a
33 }' '
34 done
35 time=' echo "scale=3;${times[0]}*86400+${times[1]}*3600+${times[2]}*60+${
36 times[3]}+${times[4]}*0.001" | bc '
37 res= $energy", "$time
38 QUERY="INSERT INTO results (id, energy, mp2, dispersion, TOTAL_ENERGY,
39 time) values($exp,$res); "
40 eval "${SQL}\ "${QUERY}\ "
41 done

```

Listing D.2: Bash script for obtaining the results

Index

A

Activation Function	27
AMS	40
ANI-1ccx	40
ANI-1x	40
ANI-2x	40
Artificial Intelligence	23

B

B2PLYP-D3	7
B3LYP	5
B97M-D3BJ	6
Bash	11
Basis Set	7
Bias	26
BLYP	6
Bond Dissociation Energy	9

C

Classification	24
Cutoff Radius	34

D

D3BJ	6
Data Science	40
Deep Learning	26
def2-TZVPP	7
Density Functional Theory	4, 23, 30
DFTB	44
Dispersion	6
Double Hybrid	7
DSD-PBEP86 D3BJ	7

E

Electron Density	4
Exchange-Correlation potential	5

F

Fortran	40
90	41
2003	40
Functionals	4

G

G3B3	14
GGA	6

H

Hartree-Fock	4, 30
High Dimensional Neural Network Potentials	23
Hybrid Functionals	5

L

layer	26
hidden	27
input	26
output	26

M

Machine Learning	23
Supervised	23
Unsupervised	23
Meta-GGA	6
Minnesota Functionals	5
M06-2X	5
M06-L	5
MP2	7
MUAE	14
MUE	14
MySQL	11

N

Neural Netowrks	26
node	27

O

Object Oriented Programming	40
Orca	14
Overfitting	29

P

PBE	6
post-Hartree-Fock	4
Potential Energy Surface	30
Python	10, 40
python	
Keras	40
TensorFlow	40

R

Radical Stabilization Energy	8
Range Separated	7
Regression	24
Linear	24
Multiple	25
Simple	24
Relational Databases	11
RI-J	8
RI-JCOSX	8
RI-JK	8

RuNNer	41	Training	23
S			
SCAN-D3BJ	6	U	
SCM	40	Underfitting	29
Shell	11		
SQL	11		
Symmetry Functions	34	W	
T			
Testing	24	wB97M-D3BJ	7
		wB97X-D3BJ	7
		Weights	26

Bibliography

- (1) Stephens, P. J.; Devlin, F. J.; Chabalowski, C. F.; Frisch, M. J. *The Journal of Physical Chemistry* **1994**, *98*, 11623–11627.
- (2) Zhao, Y.; Truhlar, D. G. *Theoretical Chemistry Accounts* **2007**, *120*, 215–241.
- (3) Grimme, S.; Ehrlich, S.; Goerigk, L. *Journal of Computational Chemistry* **2011**, *32*, 1456–1465.
- (4) Perdew, J. P.; Burke, K.; Ernzerhof, M. *Physical Review Letters* **1996**, *77*, 3865–3868.
- (5) Tao, J.; Perdew, J. P.; Staroverov, V. N.; Scuseria, G. E. *Physical Review Letters* **2003**, *91*, DOI: [10.1103/physrevlett.91.146401](https://doi.org/10.1103/physrevlett.91.146401).
- (6) Mardirossian, N.; Head-Gordon, M. *The Journal of Chemical Physics* **2015**, *142*, 074111.
- (7) Sun, J.; Ruzsinszky, A.; Perdew, J. *Physical Review Letters* **2015**, *115*, DOI: [10.1103/physrevlett.115.036402](https://doi.org/10.1103/physrevlett.115.036402).
- (8) Halsey-Moore, C.; Jena, P.; McLeskey, J. T. *Computational and Theoretical Chemistry* **2019**, *1162*, 112506.
- (9) Mardirossian, N.; Head-Gordon, M. *The Journal of Chemical Physics* **2016**, *144*, 214110.
- (10) Mardirossian, N.; Head-Gordon, M. *Physical Chemistry Chemical Physics* **2014**, *16*, 9904.

- (11) Jana, S.; Šmiga, S.; Constantin, L. A.; Samal, P. *Journal of Chemical Theory and Computation* **2020**, *16*, 7413–7430.
- (12) Kozuch, S.; Martin, J. M. L. *Physical Chemistry Chemical Physics* **2011**, *13*, 20104.
- (13) Grimme, S. *The Journal of Chemical Physics* **2006**, *124*, 034108.
- (14) Weigend, F.; Ahlrichs, R. *Physical Chemistry Chemical Physics* **2005**, *7*, 3297.
- (15) Burow, A. M.; Sierka, M.; Mohamed, F. *The Journal of Chemical Physics* **2009**, *131*, 214101.
- (16) Neese, F.; Wennmohs, F.; Hansen, A.; Becker, U. *Chemical Physics* **2009**, *356*, 98–109.
- (17) Hioe, J.; Šakić, D.; Vrček, V.; Zipse, H. *Organic & Biomolecular Chemistry* **2015**, *13*, 157–169.
- (18) Neese, F. *WIREs Computational Molecular Science* **2011**, *2*, 73–78.
- (19) Neese, F. *WIREs Computational Molecular Science* **2017**, *8*, DOI: [10.1002/wcms.1327](https://doi.org/10.1002/wcms.1327).
- (20) Behler, J.; Parrinello, M. *Physical Review Letters* **2007**, *98*, DOI: [10.1103/physrevlett.98.146401](https://doi.org/10.1103/physrevlett.98.146401).
- (21) Behler, J. *The Journal of Chemical Physics* **2011**, *134*, 074106.
- (22) Behler, J. *Journal of Physics: Condensed Matter* **2014**, *26*, 183001.
- (23) Behler, J. *International Journal of Quantum Chemistry* **2015**, *115*, 1032–1050.
- (24) Behler, J. *Angewandte Chemie International Edition* **2017**, *56*, 12828–12840.
- (25) Mitchell, T., *Machine Learning*; McGraw-Hill: New York, 1997.
- (26) Wikipedia, I. Neural Network https://en.wikipedia.org/wiki/Neural_network.
- (27) Cybenko, G. *Mathematics of Control, Signals, and Systems* **1989**, *2*, 303–314.
- (28) Hornik, K. *Neural Networks* **1991**, *4*, 251–257.

- (29) Hinton, G. E.; Osindero, S.; Teh, Y.-W. *Neural Computation* **2006**, *18*, 1527–1554.
- (30) Hellström, M.; Behler, J. In *ACS Symposium Series*; American Chemical Society: 2019, pp 49–59.
- (31) Rüger, R.; Franchini, M.; Trnka, T.; Yakovlev, A.; van Lenthe, E.; Philipsen, P.; van Vuren, T.; Klumpers, B.; Soini, T. AMS 2021.1, SCM, Theoretical Chemistry, Vrije Universiteit, Amsterdam, The Netherlands.
- (32) Rüger, R.; Yakovlev, A.; Philipsen, P.; Borini, S.; Melix, P.; Oliveira, A.; Franchini, M.; van Vuren, T.; Soini, T.; de Reus, M.; Asl, M. G.; Teodoro, T. Q.; McCormack, D.; Patchkovskii, S.; Heine, T. AMS DFTB 2021.1, SCM, Theoretical Chemistry, Vrije Universiteit, Amsterdam, The Netherlands.
- (33) Porezag, D.; Frauenheim, T.; Köhler, T.; Seifert, G.; Kaschner, R. *Physical Review B* **1995**, *51*, 12947–12957.